THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'Ecole normale supérieure de Paris

Balancing Energy, Security and Circuit Area in Lightweight Cryptographic Design

Ecole doctorale n° 386

SCIENCES MATHEMATIQUES DE PARIS CENTRE

Spécialité INFORMATIQUE

COMPOSITION DU JURY :

M. GUILLEY Sylvain
Télécom ParisTech, Rapporteur

M. CLAVIER Christophe
Université de Limoges, Rapporteur

M. GOUBIN Louis
Université de Versailles-St-Quentin-en-Yvelines, Membre du jury

M. VUILLEMIN Jean
École normale supérieure, Membre du jury

M. KOCHER Paul
Cryptography Research, Membre du jury

Mme. TRICHINA Elena
Cryptography Research, Membre du jury

Soutenue par Rodrigo PORTELLA DO CANTO
le 27 octobre 2016

Dirigée par **David NACCACHE**

ENS ÉCOLE NORMALE SUPÉRIEURE

PSL RESEARCH UNIVERSITY PARIS

Balancing Energy, Security and Circuit Area in Lightweight Cryptographic Hardware Design

Thèse de Doctorat

en vue de l'obtention du grade de

Docteur de l'École normale supérieure
(spécialité informatique)

présentée et soutenue publiquement le 27 octobre 2016 par

RODRIGO PORTELLA DO CANTO

devant le jury composé de :

Directeur de thèse :	David Naccache	(École normale supérieure)
Rapporteurs :	Christophe Clavier	(University of Limoges)
	Sylvain Guilley	(Télécom ParisTech)
Examinateurs :	Louis Goubin	(University of Versailles-Saint-Quentin-en-Yvelines)
	Paul Kocher	(Rambus Cryptography Research)
	Elena Trichina	(Rambus Cryptography Research)
	Jean Vuillemin	(École normale supérieure)

École doctorale 386: Sciences mathématiques de Paris Centre
Unité de recherche: UMR 8548 - Département d'Informatique de l'École normale supérieure
Laboratoire de recherche affilié au CNRS et a INRIA

Balancing Energy, Security and Circuit Area in Lightweight Cryptographic Hardware Design

Thèse de Doctorat

en vue de l'obtention du grade de

Docteur de l'École normale supérieure
(spécialité informatique)

présentée et soutenue publiquement le 27 octobre 2016 par

RODRIGO PORTELLA DO CANTO

devant le jury composé de :

Directeur de thèse :	David Naccache	(École normale supérieure)
Rapporteurs :	Christophe Clavier	(University of Limoges)
	Sylvain Guilley	(Télécom ParisTech)
Examinateurs :	Louis Goubin	(University of Versailles-Saint-Quentin-en-Yvelines)
	Paul Kocher	(Rambus Cryptography Research)
	Elena Trichina	(Rambus Cryptography Research)
	Jean Vuillemin	(École normale supérieure)

École doctorale 386: Sciences mathématiques de Paris Centre
Unité de recherche: UMR 8548 - Département d'Informatique de l'École normale supérieure
Laboratoire de recherche affilié au CNRS et a INRIA

ACKNOWLEDGEMENTS

First and foremost I want to express my sincere gratitude to Prof. *David Naccache*, for his support, dedication and patience. *David* has always been an incentive to me. He never let me lose focus and always made sure I kept on top of my work, which taught me how to become a better researcher. Without the encouragement of Prof. Naccache, this work would not have been completed.

I am mostly grateful to *Louis Goubin*, *Paul Kocher*, *Elena Trichina*, and *Jean Vuillemin* for agreeing to serve in this thesis committee. I express my particular gratitude to my thesis referees *Christophe Clavier* and *Sylvain Guilley* for their availability and dedication. I am very honored to have such a prestigious committee.

I would like to thank my colleagues at Rambus and Cryptography Research for having always encouraged me to continue with my research. A special thanks to *Elke De Mulder* for creative time off discussing innovative projects and supporting me to pursue this thesis. I would like to show my appreciation to *Roberto Rivoir* and *Pankaj Rohatgi* whose understanding gave me the confidence to finish this work. A very special thank you to *Craig Hampel*, who continuously believed in my work and was always eager to help.

I would also like to show my appreciation to the co-authors of the publications I worked with. All colleagues I had the pleasure to collaborate with were truly cooperative and my exchange with them was very constructive. A special thanks to *Diana Ştefania Maimuţ* and *Roman Korkikian* who were always available to support and contribute to my research. More than colleagues, *Diana* and *Roman* have become dear friends.

I would like to dedicate this work to some very special people who provided endless support and understanding throughout the development of this work. To my mother *Sônia Portella do Canto* who always believed in me and encouraged me to pursue my Ph.D. studies. Your endless dedication made me the man I am today. You are the bravest person I have ever met. The least I could do to pay you back a small fraction of the love, understanding and guidance you gave me was to make you proud. I hope I succeeded in doing so. Thank you for all your dedication and love. To my second mother *Natacha Laniado*, I cannot express how grateful I am of having you as a friend. Without you this thesis would have not been possible to complete. All the support you gave me, and all the issues you dealt with were outstanding. You are the most authentic and energetic person I have ever met and I love sharing good times with you.

To my beautiful and cheerful wife *Daiane Oakes*, I am thankful to eternity and beyond for having you in my life. You make me a better man every day. The grace in your eyes and the delight of your smile makes me the happiest person alive. Thank you for all your love and support, I love you. To my sister-in-law *Rachel Oakes* and to my mother-in-law *Leonilda Ferreira*, thank you for being a family to me and loving me as one of yours.

Last but not the least, I express all my gratitude to *Crégolas*, for being a true companion and for always sharing love and attention.

Rodrigo Portella do Canto

Nada é por acaso.

– Daiane Oakes

To Irma Arenhart and Alci Portella do Canto, in memoriam.

CONTENTS

1 Introduction **13**
- 1.1 Terminology . . . 14
- 1.2 Hardware System Security . . . 17
- 1.3 Notations and Conventions . . . 18
- 1.4 Finite Fields Arithmetic . . . 19
- 1.5 Thesis Outline . . . 22
- 1.6 Publications . . . 23

2 From a Transistor to a Cryptosystem **26**
- 2.1 Integrated Circuit and Logic Design . . . 27
  - 2.1.1 Introduction . . . 27
  - 2.1.2 VLSI Design . . . 27
  - 2.1.3 The CMOS Transistor . . . 30
  - 2.1.4 CMOS Logic . . . 31
    - 2.1.4.1 The Inverter . . . 32
    - 2.1.4.2 The NAND Gate . . . 33
    - 2.1.4.3 Compound Gates . . . 34
    - 2.1.4.4 Tri-state Buffers . . . 34
  - 2.1.5 CMOS I-V Characteristics . . . 35
    - 2.1.5.1 CMOS Electrical Properties . . . 35
    - 2.1.5.2 Non-Ideal I-V Effects . . . 37
- 2.2 Hardware-Based Cryptosystems . . . 41
  - 2.2.1 Introduction . . . 41
  - 2.2.2 Definitions . . . 42
  - 2.2.3 Hardware Design Architecture . . . 42
    - 2.2.3.1 Throughput and Latency . . . 43
    - 2.2.3.2 Area . . . 43
  - 2.2.4 Cryptographic Hardware Design . . . 44
    - 2.2.4.1 Iterative Looping . . . 45
    - 2.2.4.2 Loop Unrolling . . . 45
    - 2.2.4.3 Pipelining . . . 46
    - 2.2.4.4 Sub-Pipelining . . . 48
    - 2.2.4.5 Pseudo-Random Sequences in Hardware . . . 49
- 2.3 Private-Key Cryptosystems . . . 50
  - 2.3.1 The Data Encryption Standard . . . 50
  - 2.3.2 The Advanced Encryption Standard . . . 50
    - 2.3.2.1 AES Rounds . . . 51
    - 2.3.2.2 AES in Hardware (FPGA and ASIC) . . . 53
- 2.4 Cryptographic Hash Functions . . . 57
  - 2.4.1 Introduction . . . 57
  - 2.4.2 Security Requirements of Hash Functions . . . 58
    - 2.4.2.1 Preimage Resistance . . . 58
    - 2.4.2.2 Second Preimage Resistance . . . 58
    - 2.4.2.3 Collision Resistance . . . 59
    - 2.4.2.4 Overview of Hash Algorithms . . . 60
  - 2.4.3 The Secure Hash Algorithm 1 . . . 60

2.4.4 The Secure Hash Algorithm 2 . . . 60
2.4.5 Implementation Tradeoffs and Design Methodologies . . . 62
2.4.6 Known SHA-2 Hardware Optimization Techniques . . . 62
2.4.7 FPGA-Based Cryptography . . . 63
2.4.8 SHA-2 in Hardware (FPGA and ASIC) . . . 63

3 Cryptographic Hardware Acceleration and Power Minimization 66
3.1 BCH with Barrett Polynomial Reduction . . . 67
3.1.1 Introduction . . . 67
3.1.2 Barrett's Reduction Algorithm . . . 67
3.1.2.1 Dynamic Constant Scaling . . . 68
3.1.3 Barrett's Algorithm for Polynomials . . . 69
3.1.3.1 Orders . . . 69
3.1.3.2 Terminology . . . 69
3.1.3.3 Polynomial Barrett Complexity . . . 70
3.1.3.4 Barrett's Algorithm for Multivariate Polynomials . . . 71
3.1.3.5 Dynamic Constant Scaling in $\mathbb{Q}[\vec{x}]$ . . . 73
3.1.4 Application to BCH Codes . . . 75
3.1.4.1 General Remarks . . . 75
3.1.4.2 BCH Preliminaries . . . 75
3.1.4.3 BCH Decoding . . . 76
3.1.4.4 Syndrome . . . 77
3.1.4.5 Error Location . . . 77
3.1.4.6 Peterson's Algorithm . . . 77
3.1.4.7 Chien's Error Search . . . 77
3.1.5 Implementation and Results . . . 78
3.1.5.1 Standard Architecture . . . 78
3.1.5.2 LFSR and Improved LFSR Architectures . . . 79
3.1.5.3 Barrett Architecture (regular and pipelined) . . . 79
3.1.5.4 Performance . . . 80
3.2 Managing Energy on SoCs and Embedded Systems . . . 81
3.2.1 Introduction . . . 81
3.2.2 The Model . . . 81
3.2.3 Optimizing Power Consumption While Avoiding System Malfunction . . . 82
3.2.4 The General Case . . . 84
3.2.5 Probabilistic Strategies . . . 84

4 Side-Channel Attacks and Hardware Countermeasures 86
4.1 An Economical Introduction to Side-Channel Attacks . . . 87
4.2 Differential Cryptanalysis . . . 88
4.3 Differential Power Analysis . . . 89
4.4 Power Scrambling and the Reconfigurable AES . . . 93
4.4.1 Introduction . . . 93
4.4.2 The Proposed AES Design . . . 93
4.4.3 Energy and Security . . . 94
4.4.3.1 Power Analysis . . . 94
4.4.3.2 Power Scrambling . . . 94
4.4.3.3 Transient Fault Detection . . . 97
4.4.3.4 Permanent Fault Detection . . . 97
4.4.3.5 Runtime Configurability . . . 98
4.4.4 Halving the Memory Required for AES Decryption . . . 99
4.4.5 Implementation Results . . . 100
4.5 Cryptographically Secure On-Chip Firewalling . . . 102
4.5.1 Introduction . . . 102
4.5.2 Identifying Attack Surfaces on NoCs . . . 103
4.5.2.1 Request Path . . . 103
4.5.2.2 Firewall Reprogramming Path . . . 103
4.5.2.3 Firewall State at Rest . . . 104

4.5.3 Integration of Security Resources into an SoC . . . 104
4.5.3.1 Securing the Request Path . . . 105
4.5.3.2 Securing the Firewall . . . 105
4.5.4 Access Control Firewalling to On-Chip Resources . . . 105
4.5.4.1 Endpoint versus NoC Firewalling . . . 105
4.5.4.2 Cryptographically Secure Access Control . . . 106
4.5.4.3 CSAC Synthesis Results . . . 109
4.5.4.4 FPGA Implementation . . . 110
4.6 Practical Instantaneous Frequency Analysis Experiments . . . 112
4.6.1 Introduction . . . 112
4.6.2 Preliminaries . . . 113
4.6.2.1 The Hilbert Huang Transform . . . 113
4.6.2.2 AES Hardware Implementation . . . 116
4.6.3 Hilbert Huang Transform and Frequency Leakage . . . 117
4.6.3.1 Why Should Instantaneous Frequency Variations Leak Information? . . . 117
4.6.3.2 Power consumption of one AES round . . . 118
4.6.3.3 Hilbert Huang Transform of an AES Power Consumption Signal . . . 119
4.6.4 Correlation Instantaneous Frequency Analysis . . . 121
4.6.4.1 Correlation Instantaneous Frequency Analysis on Unprotected Hardware 122
4.6.4.2 Correlation Instantaneous Frequency Analysis in the Presence of DVS . . 124

5 Zero-Knowledge Protocols and Authenticated Encryption 126
5.1 Public-Key Based Lightweight Swarm Authentication . . . 127
5.1.1 Introduction . . . 127
5.1.2 Preliminaries . . . 127
5.1.2.1 Fiat-Shamir Authentication Protocol . . . 127
5.1.2.2 Topology-Aware Distributed Spanning Trees . . . 128
5.1.3 Distributed Fiat-Shamir Authentication . . . 129
5.1.3.1 The Approach . . . 129
5.1.3.2 Back-up Authentication . . . 130
5.1.4 Security . . . 130
5.1.4.1 Soundness . . . 130
5.1.4.2 Zero-knowledge . . . 130
5.1.4.3 Security Analysis . . . 131
5.2 The Offset Merkle-Damgård Authenticated Cipher . . . 133
5.2.1 Introduction . . . 133
5.2.2 Preliminaries . . . 135
5.2.2.1 Security Definitions and Goals . . . 137
5.2.2.2 Quantitative Security Level of OMD-SHA256 . . . 139
5.2.2.3 Quantitative Security Level of OMD-SHA512 . . . 139
5.2.2.4 Security Proofs . . . 140
5.2.2.5 Generalization of OMD Based on Tweakable Random Functions . . . 140
5.2.2.6 Instantiating Tweakable RFs with PRFs . . . 143
5.2.3 Specification of OMD . . . 144
5.2.3.1 The OMD Mode of Operation . . . 145
5.2.3.2 OMD-SHA256: Primary Recommendation for Instantiating OMD . . . 146
5.2.3.3 OMD-SHA512: Secondary Recommendation for Instantiating OMD . . . 148
5.2.3.4 Compression Functions of SHA-256 and SHA-512 . . . 149

6 Conclusion 154

A Code: Barrett's Algorithm for Polynomials 179

B Compression Functions 181
B.1 Compression Functions of SHA-256 and SHA-512 . . . 181
B.1.1 The Compression Function of SHA-256 . . . 182
B.1.2 The Compression Function of SHA-512 . . . 182

List of Figures

1.1 The hierarchical diagram of the different fields of cryptography. . . . 14
1.2 The broad fields of cryptography [MvV97]. . . . 15
1.3 The hierarchical diagram of the different fields of cryptanalysis [PP09]. . . . 17

2.1 Gajski-Kuhn $\mathcal{Y}$-chart [GK83]. . . . 28
2.2 The different design styles of a digital VLSI circuit [GBC+96]. . . . 29
2.3 The two types of CMOS devices. . . . 31
2.4 CMOS transistor symbols and switch levels. . . . 32
2.5 General logic gate using *pull-up* and *pull-down* networks. . . . 32
2.6 CMOS inverter gate schematic and symbol. . . . 33
2.7 CMOS NAND gate schematic and symbol. . . . 33
2.8 CMOS AND-OR-INVERTER-22 schematic and symbol. . . . 34
2.9 CMOS transmission gate. . . . 34
2.10 CMOS tri-state buffer schematic and symbol. . . . 35
2.11 I-V characteristics of ideal nMOS (upper graph) and pMOS (lower graph) transistors [WH10]. . . . 38
2.12 General Iterative Looping architecture. . . . 45
2.13 General architecture of Loop Unrolling. . . . 46
2.14 General architecture of Pipelining. . . . 47
2.15 General architecture of Sub-Pipelining. . . . 48
2.16 General architecture of a Feedback Shift Register (FSR). . . . 49
2.17 General architecture of a Linear Feedback Shift Register (LFSR). . . . 49
2.18 Secret-key cryptography (overview). . . . 50
2.19 The AES *state*. . . . 51
2.20 The AES encryption flowchart. . . . 52
2.21 The AES decryption flowchart. . . . 52
2.22 Application of *SubBytes* transformation to the *state*. . . . 53
2.23 Application of *ShiftRows* transformation to the *state*. . . . 53
2.24 Application of *MixColumns* transformation to the *state*. . . . 53
2.25 Decomposition of key scheduling into *KeyExpansion* and *RoundKeySelection* for $N_k = 6$ (192-bit key) and $N_b = 4$ (128-bit data block). In the figure, k_i is a word of 32 bits. . . . 54
2.26 *KeyExpansion* formula for $i \bmod N_k \neq 0$. . . . 54
2.27 *KeyExpansion* formula for $i \bmod N_k = 0$. . . . 54
2.28 Basic protocol for digital signatures with a hash function. . . . 57
2.29 Substitution attack on a hash scheme without second preimage resistance. . . . 59
2.30 Attack on a hash scheme without second collision resistance. . . . 59
2.31 General block diagram of the hardware SHA-2 implementation. . . . 61

3.1 Standard LFSR architecture block diagram. (Design `BCH-LFSR`). . . . 79
3.2 Improved LFSR architecture block diagram. `In` denotes the module's serial input. (Design `BCH-LFSR-improved`). . . . 79
3.3 Example of a request function $x(t)$ with a threshold x_0. When $x(t) \geq x_0$, it is more advantageous to be in $\mathcal{A}$-mode. Otherwise $\mathcal{S}$ should better go into $\mathcal{B}$-mode. . . . 83
3.4 Example of $v(t)$ (purple) for an example function $x(t)$ (blue). . . . 85
3.5 Example of $v(t)$ (purple) for $x(t) = |\sin(x) + \sin(2x)|$ (blue). . . . 85

4.1 Power trace of an RSA exponentiation. . . . 89
4.2 AES encryption flowchart. . . . 93

4.3 AES decryption flowchart. . . . 94
4.4 Flow of computation in time. . . . 95
4.5 Unprotected implementation: Pearson correlation value of a correct (red) and an incorrect (green) key byte guess. 500,000 power traces. . . . 95
4.6 Power scrambling with a PRNG. . . . 96
4.7 LFSR implementation: Pearson correlation value of a correct (red) and an incorrect (green) key byte guess. 1,200,000 power traces. . . . 96
4.8 Power scrambling with tri-state buffers. . . . 97
4.9 Tri-state buffers implementation: Pearson correlation value of the correct key byte (green) and a wrong key byte guess (red). 800,000 power traces. . . . 97
4.10 Transient fault detection scheme for AES. . . . 98
4.11 Permanent fault detection scheme for AES. . . . 98
4.12 Memory halving for AES decryption when $N_r = 10$. . . . 100
4.13 AES design's inputs and outputs. . . . 100
4.14 Firewalling in an SoC based on NoC interconnect. . . . 102
4.15 Evolution of NoC integration services. . . . 103
4.16 Different attack surfaces on a NoC firewall. . . . 104
4.17 Simple firewall partitioning of an address space covering two targets. . . . 105
4.18 Endpoint firewall controlling access from initiators to a targets. . . . 106
4.19 Content of a complex CSAC region. . . . 107
4.20 HMAC intermediate used for key integrity checks. . . . 108
4.21 Timing diagram of reset-on-scan block. . . . 109
4.22 Illustration of the EMD: (a) is the original signal $u(t)$; (b) $u(t)$ in thin solid black line, upper and lower envelopes are dot-dashed with their mean $m_{i,j}$ in thick solid red line; (c) shows the difference between $u(t)$ and the envelope's mean. . . . 114
4.23 The increasing frequency function $\cos((a+bt)t)$. . . . 116
4.24 Analysis of the function $\cos((a+bt)t)$: Marginal Hilbert spectrum of Fig. 4.23. . . . 116
4.25 Analysis of the function $\cos((a+bt)t)$: Hilbert's amplitude spectrum contour of Fig. 4.23. 116
4.26 Analysis of the function $\cos((a+bt)t)$: Hilbert's amplitude spectrum contour of Fig. 4.23. 116
4.27 Inverters switch simulation. . . . 117
4.28 Four AES last rounds. . . . 119
4.29 AES last round power consumption for 55 (red), 65 (blue) and 75 (black) register's flip-flops: Full voltage range. . . . 120
4.30 AES last round power consumption for 55 (red), 65 (blue) and 75 (black) register's flip-flops: Zoomed voltage range. . . . 120
4.31 AES last round power consumption for 55 (red), 65 (blue) and 75 (black) register's flip-flops: Power spectra density for the signals shown on Fig. 4.29. . . . 120
4.32 Power consumption of our experimental AES-128 implementation: initial signal $u(t)$. . . . 121
4.33 Power consumption of our experimental AES-128 implementation: The Empirical Mode Decomposition of signal $u(t)$. . . . 121
4.34 Power consumption of our experimental AES-128 implementation: IF distribution over time for the different IMFs of Fig. 4.33. . . . 122
4.35 Fourier and Hilbert power spectrum density of Fig. 4.32. . . . 122
4.36 Dependency between the Hamming distance of 9th and 10th AES round states and the IF of the first IMF component at time 276 ns (corresponding to the beginning of the last AES round). . . . 123
4.37 Maximum correlation coefficients for a byte of the last round AES key in an unprotected implementation. Although the three attacks eventually succeed CPA>CSBA>CIFA. (a) CPA (b) CSBA (c) CIFA. . . . 123
4.38 Power traces of the FPGA AES implementation. The unprotected signal is shown in red. The DVS-protected signal is shown in black. . . . 124
4.39 Maximum correlation coefficient for a byte of the last round AES key with simulated DVS. (a) CPA (b) CSBA (c) CIFA. . . . 125

5.1 Fiat-Shamir authentication round. . . . 128
5.2 The proposed algorithm running on a network. Each parent node aggregates the values computed by its children before transmitting it upwards to the base station. . . . 130

5.3 The $\mathbb{OMD}[\widetilde{R}, \tau]$ scheme using a tweakable random function $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ (i.e., $\widetilde{R} \xleftarrow{R} \mathrm{Func}^{\mathcal{T}}(n+m, n)$). The tweak space $\mathcal{T}$ consists of five mutually exclusive sets of tweaks, namely $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathbb{N} \times \{0\} \cup \mathbb{N} \times \{1\}$, where $\mathcal{N} = \{0,1\}^{|N|}$ is the set of nonces, $\mathbb{N}$ is the set of positive integers. 141
5.4 Building a tweakable PRF $\widetilde{F}_K^{\langle T \rangle} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ using a PRF $F_K : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. There are several efficient ways to define the masking function $\Delta(T)$ [Rog04a, CS07, KR11]. We use the method of [KR11]. 144
5.5 The encryption process of OMD$[F, \tau]$ using a keyed compression function $F_K : (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ with $m \leq n$. (**Top**) The encryption process when the message length is a multiple of the block length m and no padding is required. (**Middle**) The encryption process when the message length is not a multiple of the block length and the final block M_* is padded to make a full block $M_* || 10^{m-|M_*|-1}$. (**Bottom, Left**) Computing the intermediate value T_a when the bit length of the associated data is a multiple of the **input** length $n+m$. (**Bottom, Right**) Computing T_a when the bit length of the associated data is not a multiple of $n+m$ and the final block is padded to make a full block $A_* || 10^{n+m-|A_*|-1}$ is needed. The output ciphertext is $C||\mathsf{Tag}$. For operation $\oplus$ see our convention in Section 5.2.2. Five types of key-dependent masking values (corresponding to five mutually exclusive tweak sets) are used; these are denoted by $\Delta_{N,i,0}, \Delta_{N,i,1}, \Delta_{N,i,2}, \bar{\Delta}_{i,0}$ and $\bar{\Delta}_{j,1}$, for $i \geq 1$ and $j \geq 0$, where N is the nonce. Note that the masks used in computing T_a do not depend on the nonce. 147
5.6 Definition of OMD$[F, \tau]$. The function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ is a keyed compression function with $\mathcal{K} = \{0,1\}^k$ and $m \leq n$. The tag length is $\tau \in \{0, 1, \cdots, n\}$. Algorithms $\mathcal{E}$ and $\mathcal{D}$ can be called with arguments $K \in \mathcal{K}$, $N \in \{0,1\}^{\leq n-1}$, and $A, M, \mathbb{C} \in \{0,1\}^*$. ℓ_{max} is the bound on the maximum number of blocks in any input to the encryption or decryption algorithms. 148

6.1 Tradeoffs in cryptography. 155

List of Tables

2.1 Trends in microelectronics in the past decades [WH10]. . . . 27
2.2 Comparison of VLSI design methodologies [WH10]. . . . 30
2.3 Comparison of different AES implementations. . . . 55
2.4 Comparison of different pipelined AES implementations. . . . 56
2.5 Comparison of different pipelined AES implementations. . . . 65
2.6 Comparison of different SHA-2 implementations. . . . 65

3.1 Synthesis results of the four BCH designs. . . . 80
3.2 Increase ΔP of the power consumption function in a short time period ΔT. . . . 82
3.3 Increase ΔP of the power consumption function in a short time period ΔT. $(r_{\mathcal{A}} + r_{\mathcal{B}})/2$ represents the average rate due to the alternation of modes $\mathcal{A}$ and $\mathcal{B}$ during the request. . . 84

4.1 29 possible configurations. . . . 99
4.2 Number of configurations. . . . 99
4.3 Unprotected AES, LFSR and tri-state buffer designs synthesized to the 45nm *FreePDK* open-cell library. . . . 101
4.4 Spartan3E-500 utilization summary report. . . . 101
4.5 Synthesis results of five CSAC designs (4 regions, 6 initiators, 4GB of address space) on a $45nm$ technology node. . . . 109
4.6 Synthesis results of five CSAC designs (8 regions, 14 initiators, 64GB of address space) on a $45nm$ technology node. . . . 110
4.7 CSAC (4 regions, 6 initiators, 4GB of address space) synthesis on Zynq-7000 board. . . . 110
4.8 CSAC (8 regions, 14 initiators, 64GB of address space) synthesis on Zynq-7000 board. . . . 111

List of Algorithms

1 AES algorithm description. . . . 55
2 Barrett's algorithm. . . . 68
3 Polynomial Barrett Algorithm. . . . 72
4 Peterson's algorithm. . . . 78
5 Standard modular division (`BCH-standard`). . . . 78
6 Computation of an RSA signature (modular exponentiation). . . . 89
7 Dynamic Voltage Scrambling (DVS) simulator. . . . 124
8 Mooij-Goga-Wesselink algorithm, basic part. . . . 129
9 Compression function of SHA-256 183
10 Compression function of SHA-512 183

CHAPTER 1

INTRODUCTION

Cryptography is an ancient art. The word comes from the Greek $\kappa\rho\pi\tau\acute{o}\varsigma$, that means hidden, and $\gamma\rho\acute{\alpha}\phi\epsilon\iota\nu$, that means writing. It is therefore the science of writing secrets, and has started thousands of years ago. The advent of cryptography is sometimes linked with the origin of written language [Vau05]. An example of that are the hieroglyphs, a written language based on symbols, created by the Egyptians. The scribes were usually the only ones who mastered it. They used to transmit their knowledge from father to son [1] until the Egyptian society fell apart and collapsed. For this reason, hieroglyphs remained a secret to humanity until Jean-François Champollion broke the code, in 1822 [Cha24].

Many historical personalities like Julius Caesar, Francis Bacon, Louis XIV, and Napoleon invented their own ciphers and used them for their secret correspondence [MM03]. During the American War of Independence, spies employed a code system where words were replaced with numbers extracted from a codebook. Around 1900, the French military employed a ciphering machine called the *Bazeries cylinder*. In the World War II, the German military bought the *Enigma machine*, perhaps the most famous cryptosystem.

Early cryptographic schemes focused on converting language into symbols, or mixing language characters, to carry the message from one place to another. Until late 19^{th} century, cryptographic algorithms lacked rigorous and formal security proofs, and were seen as an art rather than science. It was only in the early 20^{th} century that more sophisticated encryption means, using mechanical and electromechanical machines, such as the *Enigma machine*, were created, and cryptography increased in complexity. This development happened in parallel with cryptanalysis, the study of breaking such codes and ciphers. The *Enigma*, for example, was reversed engineered several times by Polish, French and British military intelligence, changing the course of World War II [Vau05].

A broader concept, cryptology, is the science of secure communications or, in other words, the study of cryptosystems. The word stems from Greek roots $\kappa\rho\pi\tau\acute{o}\varsigma$ and $\lambda o\gamma\acute{\iota}\alpha$, meaning hidden and word, respectively. Cryptology basically subdivides into two introduced disciplines, cryptography and cryptanalysis, since these two topics are closely related. When implementing a cryptosystem, one needs to prove its strength by actually attacking it (or proving the infeasibility of the attack). In other words, cryptanalysis assures that a cryptosystem is secure.

Fig. 1.1 depicts the three main building blocks of cryptography [PP09]:

Symmetric Algorithms Two parties have an encryption and decryption method for which they share a secret key. All cryptography from ancient times until 1976 was exclusively based on symmetric methods. Symmetric ciphers are still in widespread use, especially for data encryption and message integrity checks;

Asymmetric (or Public-Key) Algorithms In 1976 an entirely different type of cipher was introduced by Whitfield Diffie, Martin Hellman and Ralph Merkle [DH76, Mer78]. In public-key cryptography, a user possesses a secret key as in symmetric cryptography but also a public key. Asymmetric

1. Although some craftsmen were able to get their sons into the school for scribes, it was very rare.

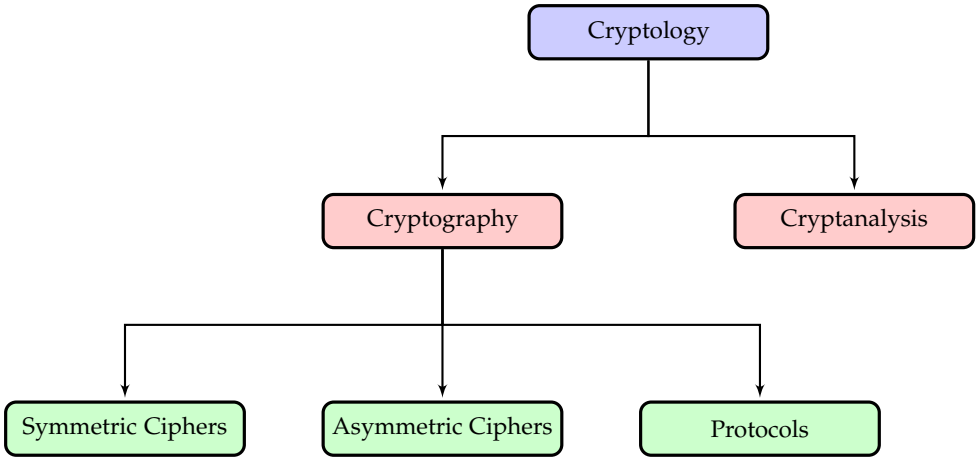


Figure 1.1: The hierarchical diagram of the different fields of cryptography.

algorithms can be used for applications such as digital signatures and key establishment, and also for classical data encryption;

Cryptographic Protocols Also called security protocols, they are abstract or concrete protocols that perform security-related functions applying cryptographic methods or primitives. Cryptographic protocols describe how the algorithms should be used, with details about data structures, key establishment, authentication between parties, secret sharing methods, secure multi-party computation, etc.

The early usage of cryptography was to enable governmental and military applications. Nowadays it is used everywhere and is taken for granted by most people. We use underlying cryptography embedded in cryptosystems to surf the internet, purchase online and communicate using the cellphone. It is a powerful and important tool that enables fundamental operations in our modern society. Modern cryptology appeared somewhere in the mid-1970s, when academic research started to mature this discipline. Although before, in 1949, Claude Shannon published a paper [Sha49] linking cryptography to information theory, discussing the foundations of modern cryptography and providing a comprehensive theory of secrecy systems. His work proved the unconditional security of the *one-time pad* cipher, earlier published by Gilbert Vernam in 1926 [Ver].

Apart from the groundbreaking work of Claude Shannon, two other contributions mark the 1970s as the beginning of a new era: First, in 1973, Horst Feistel published a work [Fei73, Fei74] of an iterative symmetric block-cipher called Lucifer, based on the Feistel construction. This algorithm was submitted to the National Bureau of Standards and was accepted in 1976 as the Data Encryption Standard (DES), becoming the first standardized cryptographic algorithm. Secondly, in 1976, the concept proposed by Whitfield Diffie and Martin Hellman [DH76] defined the basis of a new paradigm: public-key cryptography. They showed that secret communication was possible without any transfer of secrets.

1.1 Terminology

Cryptography has developed an extensive vocabulary. This section lists some definitions that will be encountered in the rest of this thesis.

The basic security requirements are defined as the following properties [Til99]:

Confidentiality is a service used to keep the content of information hidden from all but those authorized to have it. *Secrecy* is a term synonymous to confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.

Data integrity is a service which addresses the unauthorized alteration of data. To ensure data integrity, one must have the ability to detect data manipulation by unaccredited parties. Data manipulation includes such operations as insertion, deletion, and substitution.

Authentication is a service related to identification. This function applies both to entities and to information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

Nonrepudiation is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny that such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

Encryption is the first basic cryptographic operation to ensure *secrecy* or *confidentiality* of data transmitted across an insecure communication channel. Basically, encryption takes an information element (often called the *message*, *message block*, or *plaintext*) and translates it into a *cryptogram* (usually referred of as the *codeword* or *ciphertext*) using a cryptographic *secret key*. Decryption is the reverse operation, where the *ciphertext* is translated to a *plaintext* under a *secret key*. The step-by-step description of an encryption (or decryption) scheme is called the *encryption algorithm* (or *decryption algorithm*). Denominations such as *ciphers*, *cryptoalgorithms* or *cryptosystems* are often used without the need to differentiate encryption from decryption.

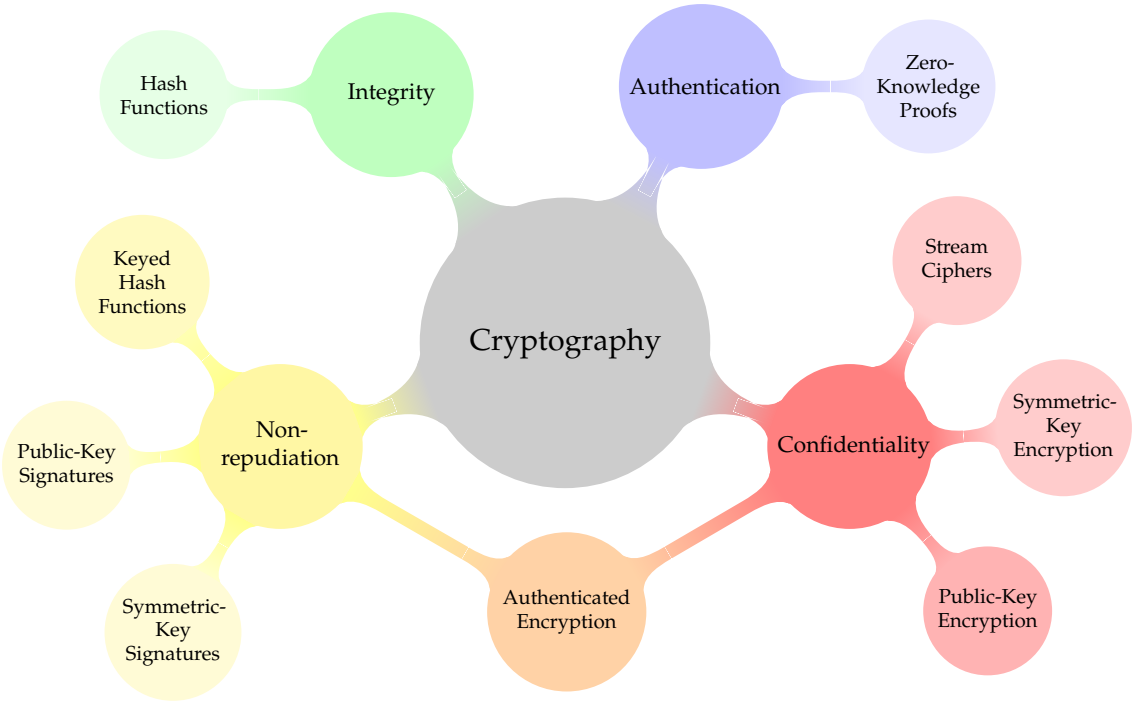


Figure 1.2: The broad fields of cryptography [MvV97].

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities. Many of these will be briefly introduced in this chapter, with the detailed discussion being left to later chapters. These primitives should be evaluated with respect to various criteria such as:

Security Level. This is usually difficult to quantify. Often it is given in terms of the number of operations required (using the best methods currently known) to defeat the intended target. Typically the security level is defined by an upper bound on the amount of work necessary to defeat the objective. This is sometimes called the *work factor*.

Functionality. Primitives will need to be combined to meet various information security objectives. Which primitives are most effective for a given objective will be determined by the primitives' basic properties.

Methods of operation. Primitives, when applied in various ways and with various inputs, will typically exhibit different characteristics; thus, one primitive could provide very different functionalities depending on its mode of operation or usage. In other words, *methods of operation* means how different characteristics the protocol exhibits when applied in various ways and with various inputs.

Performance. Refers to the efficiency of a primitive in a particular mode of operation. (For example, an encryption algorithm may be rated by the number of bits per second which it can encrypt.)

Ease of implementation. Refers to the difficulty of realizing the primitive in a practical instantiation. This might include the complexity of implementing the primitive in either software or hardware.

Cryptanalysis Cryptanalysis is the art of breaking cryptosystems. Breaking a cipher does not necessarily mean finding a practical way to recover the plaintext from the ciphertext. Instead, breaking a cipher involves finding weaknesses in the cipher that can be exploited with complexity less than a brute-force attack [Sch00], in which every possible key is tested in sequence until the correct one is found. A cryptanalysis is also considered successful when it breaks a reduced-round variant of the cipher — 8-round AES versus the full 10-round AES (for 128-bit key), for example. Academic publications usually start out with a reduced-round variant cryptanalysis that eventually escalates to all the cipher rounds.

Modern cryptanalysis is fundamentally based on *Kerckhoffs's principle* [Ker83], which states that a cryptosystem should remain secure even if everything about the system, except the key, is public knowledge. Kerckhoffs's article, published in 1883 and entitled "The Military Cryptography", states the following design principles:

— The system must be practically, if not mathematically, indecipherable.
— The system must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.
— The system's key must be communicable and retainable without the help of written notes, and changeable or modifiable at the will of the correspondents.
— The system must be applicable to telegraphic correspondence.
— Apparatus and documents must be portable, and its usage and function must not require the concourse of several people.
— Finally, it is necessary, given the circumstances that command its application, that the system be easy to use, requiring neither mental strain nor the knowledge of a long series of rules to observe.

We can depict the field of cryptanalysis as shown in Fig. 1.3. The first subfield of cryptanalysis, Classical cryptanalysis, is the study of how to break the cryptosystem by means of recovering the secret key from the ciphertext or, alternatively, recovering the plaintext from the ciphertext. It is divided in Mathematical Analysis and Brute-Force Attacks. The first exploits algorithmic weaknesses to break the cipher and therefore discovers the secret key; the second treats the cipher as a black-box and tries all the possible key combinations, until the correct one is found.

Implementation attacks are a subfield of cryptanalysis exploiting implementation weaknesses rather than algorithmic mathematical properties. An unprotected implementation of a block cipher may leak information through the power being consumed or the time necessary to perform internal operations, and this side-channel information can be used by a cryptanalyst to break the cipher. Implementation attacks are most common when the attacker has physical access to cryptosystem.

The third cryptanalysis subfield, social engineering, uses blackmailing, espionage, bribing or other illegal practices to retrieve information that can lead to the discovery of the secret key. The more cryptosystem and secure protocols evolve, the more social engineering is used to trick people into disclosing sensitive information.

Cryptanalysis attacks are alternatively divided into passive and active attacks. A passive attack occurs when the communication is being eavesdropped and the information confidentiality is therefore

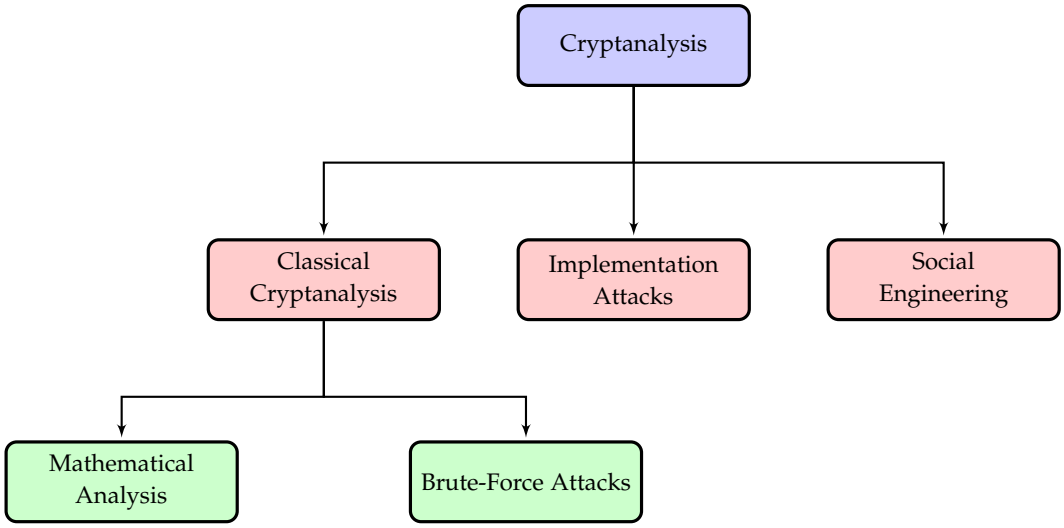


Figure 1.3: The hierarchical diagram of the different fields of cryptanalysis [PP09].

threatened. An active attack, on the other hand, occurs when the attacker attempts to modify the communication information, thus compromising the message's confidentiality, integrity and authenticity. In either way, the attack objective is to determine the key [Sti95].

The most common passive attacks include:

Ciphertext-Only The attacker has only access to the ciphertext.

Known-Plaintext The attacker has access to both the plaintext and the corresponding ciphertext.

Chosen-Plaintext The attacker has gained access to the encryption process of a cryptosystem and therefore, has the ability to input plaintext and observe the corresponding ciphertext.

Chosen-Ciphertext This is the reverse of a chosen plaintext attack. The attacker has access to the decryption process and is able to input ciphertexts and observe the original plaintext.

Three well-known active attacks are:

Man-in-the-Middle (MITM) Here the attacker controls the communication channel between two parties. The attacker can then retrieve information and send on altered messages without the other party being aware of it. Passwords are easily compromised in a MITM attack. Typically, hash functions are used to thwart such attacks.

Timing Attacks Cryptographic algorithms' inputs and internal data vary in the time it takes to process. By carefully measuring the amount of time required to perform certain operations, information can be retrieved and indeed, secret keys can even be uncovered [Koc96].

Power Analysis This technique involves interpreting power consumption measurements of various cryptographic operations to retrieve information [KJJ99]. Features such as DES permutation and shift operations can be easily spotted, as their power consumption traces are visibly different. Differential Power Analysis (DPA) is an even more powerful method of attack, in which statistical analysis and error correction techniques are also used to deduce information.

The science of cryptology is continually driven forward by the constant battle between cryptographers trying to secure information and cryptanalysts attempting to break cryptosystems.

1.2 Hardware System Security

Cryptographic engineering blends theory and practice of engineering a cryptosystem. A cryptographic engineer is thus responsible for translating the mathematical and formal descriptions of cryptographic algorithms to hardware or software systems. After designing and coding the encryption and

decryption modules, authentication blocks, digital signature schemes or any other cryptographic methods, a cryptography engineer is, nowadays, interested in cryptanalyzing the system for the purpose of checking its robustness and strength against attacks. In the past, cryptographic engineering relied on the mathematical strength of a cryptographic algorithm as a blind proof that the cryptosystem was therefore unbreakable. New cryptanalysis techniques have completely torn apart this belief.

Classical and theoretical cryptanalysis consider attack scenarios where adversaries access the cryptosystem as a black box, i.e., its inputs and outputs. For example, in a chosen ciphertext attack, it is assumed that the attacker can choose the ciphertext input into the decryption black-box and also that he can read the plaintext at the black box output. In real life, though, attackers might be even more powerful. For example, an adversary may monitor side-channel information that leaks out of the black-box, such as execution time or power consumption. The basic idea behind side-channel analysis is to infer secret information from this extra information.

Secure protocols, cryptographic algorithms and primitives do not specify how they should be implemented in hardware or software. Instead, they focus on describing their mathematical operations and transformations. The specification of a secure protocol disregards by which physical device it will be executed. For example, the target application could be a software running on a general purpose processor or a custom integrated circuit. Besides, each target platform leaks side-channel information in a different way.

The first attack of this kind was published by Paul Kocher in 1996 [Koc96]. Kocher showed how different RSA [Jr.96] operations could be tracked down by the actual time they take to compute. By doing so, Kocher demonstrated that it was possible to differentiate the multiplication and the squaring steps, therefore exposing the secret key. In 1999, Kocher *et al.* also presented a technique to infer the DES secret key by recording the power consumption of the device [KJJ99], called Differential Power Analysis (DPA). Power attacks proved to be so powerful that new academic publications rapidly demonstrated how to apply DPA to break the newly symmetric cipher approved by NIST, the Advanced Encryption Standard (AES). Although AES had been selected in the early 2000 to overcome security issues with DES, presenting a bigger key space and a more complex mathematical structure, it was not less vulnerable to DPA attacks than any other block-cipher.

Other side-channel attacks exploit electromagnetic emanations [AARR03,GMO01,QS01], optical [Kuh02] and acoustic [LU02, BWY06] leakage. In fact, side-channel analysis proved to be so powerful that the majority of techniques applied use only a portion of the side-channel information. Even if the acquired power traces present a considerable amount of noise, the idea behind differential analysis is to run and record several iterations of the device's power profile such that information can be averaged over a large number of samples.

1.3 Notations and Conventions

In this thesis we introduce the following notations and conventions:

If $\mathcal{S}$ is a finite set, $x \xleftarrow{\$} S$ means that x is chosen from S uniformly at random. $X \leftarrow Y$ is used for denoting the assignment statement where the value of Y is assigned to X. The set of all binary strings of length n bits (for some positive integer n) is denoted as $\{0,1\}^n$, the set of all binary strings whose lengths are variable but upper-bounded by L is denoted by $\{0,1\}^{\leq L}$ and the set of all binary strings of arbitrary but finite length is denoted by $\{0,1\}^*$. For two strings X and Y we use $X||Y$ and XY equivalently to denote the string obtained by concatenating Y and X. For an m-bit binary string $X = X_{m-1} \cdots X_0$ we denote the left-most bit by $\mathsf{msb}(X) = X_{m-1}$ and the right-most bit by $\mathsf{lsb}(X) = X_0$; let $X[i \cdots j] = X_i \cdots X_j$ denote a substring of X, for $0 \leq j \leq i \leq (m-1)$. Let $1^n 0^m$ denote concatenation of n ones by m zeros. For a non-negative integer i let $\langle i \rangle_m$ denote binary representation of i by an m-bit string.

For a binary string $X = X_{m-1} \cdots X_0$, let $X \ll n$ denote the left-shift operation, where the n left-most bits are discarded and the n vacated right bits are set to 0; that is, $X \ll n = X_{m-n-1} \cdots X_0 0^n$. We let $X \gg n$ denote the (unsigned) right-shift operation where the n right-most bits are discarded and the n vacated left bits are set to 0, i.e., $X \gg n = 0^n X_{m-1} \cdots X_n$. We let $X \gg_s n$ denote the *signed* right-shift

operation where the n right-most bits are discarded and the n vacated left bits are filled with the left-most bit (which is considered as the sign bit); for example, $1001100 \gg_s 3 = 1111001$. If the left-most bit of X is 0 then we have $X \gg_s n = X \gg n$.

$\neg X$ means bitwise complement of X. For two binary strings X and Y, let $X \wedge Y$ and $X \vee Y$ denote, respectively, bitwise AND and bitwise OR of the strings.

The special symbol $\perp$ means that the value of a variable is undefined; we also overload this symbol and use it to signify an error. Let $|Z|$ denote the number of elements of Z if Z is a set, and the length of Z in bits if Z is a binary string. The empty string is denoted by ε and we let $|\varepsilon| = 0$. For $X \in \{0,1\}^*$ let $X[1]||X[2]\cdots||X[m] \xleftarrow{b} X$ denote partitioning X into blocks $X[i]$ such that $|X[i]| = b$ for $1 \leq i \leq m-1$ and $|X[m]| \leq b$; let $m = |X|_b$ denote length of X in b-bit blocks.

For two binary strings $X = X_{m-1}\cdots X_0$ and $Y = Y_{n-1}\cdots Y_0$, the notation $X \oplus Y$ denotes bitwise XOR of $X_{m-1}\cdots X_{m-1-\ell}$ and $Y_{n-1}\cdots Y_{n-1-\ell}$ where $\ell = \min\{m-1, n-1\}$. That is, $X \oplus Y$ is a binary string whose length is equal to the length of the shorter operand and is obtained by XORing the shorter operand with an equal length left-most substring of the longer operand consisting of its left-most bits. Clearly, if X and Y have the same length then $X \oplus Y$ simply means their usual bitwise XOR. For any string X, define $X \oplus \varepsilon = \varepsilon \oplus X = \varepsilon$.

1.4 Finite Fields Arithmetic

The theory of finite fields is a branch of modern algebra and can be applied to multiple applications, from coding theory to cryptology. Having an elegant algebraic structure, finite fields are used to describe the underlying operations on a cryptosystem or on an error-correcting code, for example. It is therefore important to define its basic operations and rationale.

The concept of a *field* is introduced as follows:

Definition 1.1 *A field F is a non-empty set of elements with well defined* addition *and* multiplication *operations, denoted by $+$ and $*$, respectively. Moreover, for F to be a finite field, it must ensure the following conditions:*

Closure $\forall\, a, b \in F^2$,
$c = a + b$ $d = a * b$, *where* $c, d \in F$.

Associative $\forall\, a, b, c \in F^3$
$a + (b + c) = (a + b) + c$ *and* $a * (b * c) = (a * b) * c$.

Identity *There exists an identity element* $'0'$ *for addition and* $'1'$ *for multiplication that satisfies*
$0 + a = a + 0 = a$ *and* $a * 1 = 1 * a = a$ *for every* a *in* F.

Inverse *If* a *is in* F, *there exists elements* b *and* c *in* F *such that*
$a + b = 0$ *and* $a * c = 1$.

Commutative $\forall\, a, b \in F^2$
$a + b = b + a$ *and* $a * b = b * a$.

Distributive $\forall\, a, b \in F^2$
$(a + b) * c = a * c + b * c$.

The existence of a multiplicative inverse a^{-1} enables the existence of division. This is because for $a, b, c \in F$, $c = b/a$ is defined as $c = b * a^{-1}$. Similarly the existence of an additive inverse $(-a)$ enables the existence of subtraction. In this case for $a, b, c \in F$, $c = b - a$ is defined as $c = b + (-a)$.

It can be shown that the set of integers $\{0, 1, 2, ..., p-1\}$ where p is a prime, together with modulo p addition and multiplication forms a field. Such a field is called the finite field of order p, or $GF(p)$, in honor of Evariste Galois. In this thesis only binary arithmetic is considered, where p is constrained to equal 2. This is because, as shall be seen, by starting with $GF(2)$, the representation of finite field elements maps conveniently into the digital domain. Arithmetic in $GF(2)$ is therefore defined modulo 2. It is from the base field $GF(2)$ that the extension field $GF(2^n)$ is generated.

A polynomial $p(x)$ of degree n over $GF(2)$ is a polynomial of the form $p(x) = p_0 + p_1x + p_2x^2 + ... + p_nx^n$ where the coefficients p_i are elements of $GF(2) = \{0, 1\}$. Polynomials over $GF(2)$ can be added, subtracted, multiplied and divided in the usual way. A useful property of polynomials over $GF(2)$ is that $p^2(x) = (p_0 + p_1x + ... + p_nx^n)^2 = p_0 + p_1x^2 + ... + p_nx^{2n} = p(x^2)$. A polynomial $p(x)$ over $GF(2)$ of degree n is irreducible if $p(x)$ is not divisible by any polynomial over $GF(2)$ of degree less than n and greater than zero.

To generate the extension field $GF(2^n)$, an irreducible, monic polynomial $p(x)$ of degree n over $GF(2)$ is chosen. Then the set F of 2^n polynomials of degree less than n over $GF(2)$ is formed. It can then be proven that when addition and multiplication of these polynomials is taken modulo $p(x)$, the set F forms a field of 2^n elements, denoted $GF(2^n)$. Note that $GF(2^n)$ is extended from $GF(2)$ in an analogous way that the complex numbers $\mathbb{C}$ are formed from the real numbers $\mathbb{R}$ where in this case, $p(x) = x^2 + 1$.

THE FIELD WITH 2^n POINTS. Let $(GF(2^n), \oplus, \cdot)$ denote the Galois Field with 2^n points. When considering a point $\alpha \in GF(2^n)$, α can be represented in any of the following equivalent ways:

1. as an integer between 0 and 2^n,
2. as a binary string $\alpha_{n-1} \cdots \alpha_0 \in \{0, 1\}^n$, or
3. as a polynomial $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1X + \alpha_0$ with binary coefficients.

Example 1.1 *In $GF(2^{256})$: the string $0^{254}10$, the number 2 and the polynomial X are different representations of the same field element; the string $0^{254}11$, the number 3 and the polynomial $X + 1$ represent the same field element, and so forth.*

The addition $\oplus$ and multiplication $\cdot$ of two elements in $GF(2^n)$ are defined as follows:

Definition 1.2 *The addition of two elements $\alpha, \beta \in GF(2^n)$ simply means the element obtained by bitwise XORing their representations as binary strings.*

Example 1.2 $2 \oplus 1 = 0^{n-2}10 \oplus 0^{n-2}01 = 0^{n-2}11 = 3$, $2 \oplus 3 = 1$, $1 \oplus 1 = 0$, *and so forth. (Note that addition in $GF(2^n)$ is* different *from the addition of integers modulo 2^n.)*

To multiply two elements, first choose and fix an irreducible polynomial $P_n(X)$ of degree n over $GF(2)$. For example, choose the lexicographically first polynomial among the irreducible polynomials of degree n over $GF(2)$ with a minimum number of nonzero coefficients.

Example 1.3 *For $n = 256$ we use $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, for $n = 512$ we use $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$.*

To multiply two elements α and β in $GF(2^n)$ denoted by $\alpha \cdot \beta$ consider them as polynomials $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1X + \alpha_0$ and $\beta(X) = \beta_{n-1}X^{n-1} + \cdots + \beta_1X + \beta_0$, form their product in $GF(2)$ to get $\gamma(X)$ and take the remainder of dividing $\gamma(X)$ by the irreducible polynomial $P_n(X)$.

It is easy to multiply an arbitrary field element α by the element 2 (i.e., X). We describe this for $GF(2^{256})$and $GF(2^{512})$. Let $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1X + \alpha_0$ then multiplying by X we get $\alpha_{n-1}X^n + \alpha_{n-2}X^{n-1} \cdots + \alpha_1X^2 + \alpha_0X$; so if $\mathsf{msb}(\alpha) = 0$ then $2.\alpha = X.\alpha = \alpha \ll 1$. If $\mathsf{msb}(\alpha) = 1$ then we need to reduce the result by module $P_n(X)$, i.e., we have to add X^n to $\alpha \ll 1$. For $n = 256$ using $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, we have $X^{256} = X^{10} + X^5 + X^2 + 1 = 0^{245}10000100101$, so adding X^{256} means XORing with $0^{245}10000100101$. For $n = 512$ using $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$, we have $X^{512} = X^8 + X^5 + X^2 + 1 = 0^{503}100100101$, so adding X^{512} means XORing with $0^{503}100100101$.

In summary, for $GF(2^{256})$

$$
\begin{aligned}
2.\alpha &= \begin{cases} \alpha \ll 1 & \text{if } \mathsf{msb}(\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{245}10000100101 & \text{if } \mathsf{msb}(\alpha) = 1 \end{cases} \\
&= (\alpha \ll 1) \oplus ((\alpha \gg_s 255) \wedge 0^{245}10000100101)
\end{aligned}
$$

and for $GF(2^{512})$

$$
\begin{aligned}
2.\alpha &= \begin{cases} \alpha \ll 1 & \text{if } \mathsf{msb}(\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{503}100100101 & \text{if } \mathsf{msb}(\alpha) = 1 \end{cases} \\
&= (\alpha \ll 1) \oplus ((\alpha \gg_s 511) \wedge 0^{503}100100101)
\end{aligned}
$$

1.5 Thesis Outline

As depicted in Fig. 1.2, cryptography is a broad field. This thesis focuses on presenting lightweight hardware design and countermeasures to *improve* cryptographic computation. Because cryptography (and cryptanalysis) are nowadays becoming more and more ubiquitous in our daily lives, it is crucial that newly developed systems are robust enough to deal with the increasing amount of processing data without compromising the overall security.

This work addresses several different topics related to lightweight cryptographic implementations. The main contributions of this thesis are:

— a new cryptographic hardware acceleration scheme applied to BCH codes;
— hardware power minimization applied to SoCs and embedded devices;
— timing and DPA power scrambling hardware lightweight countermeasures applied to the AES;
— a cryptographically secure on-chip firewall;
— frequency analysis attack experiments;
— a new zero-knowledge protocol applied to wireless sensor networks;
— a new authenticated encryption scheme.

Organization. Chapter 2 introduces integrated circuit and logic design, from a transistor to a more formal definition of a hardware-base cryptosystem. Two of the most important cryptographic algorithms nowadays, the AES and the SHA-2 are presented in detail in Section 2.3.2 and 2.4.4, respectively. This chapter also discusses efficient ways of implementing a cryptographic hardware design, in particular focusing on AES and SHA-2.

Chapter 3 introduces a new way of computing BCH error-correcting codes by using the Barrett's Polynomial Reduction Algorithm. This algorithm is presented in detail and its application to BCH is further discussed. Hardware implementation results are given for three different architectures and compared. In addition in Chapter 3, a smart energy management system focusing on cryptographic computation requests on SoCs is presented. This scheme allows saving energy when incoming request probability is low enough to take such a risk, reducing unresponsiveness penalties whenever possible.

Chapter 4 is devoted to side-channel attacks and hardware countermeasures. In Section 4.4 a reconfigurable AES implementation technique is presented, in which unused modules are used to either scramble the power consumption, compute previous blocks to check for faults or are shut down to save energy. The DPA analysis is then further discussed, and implementation details and performance are shown in both ASIC and FPGA targets. Section 4.5 presents CSAC, a hardware protection tailored for network-on-chips that make use of firewalls to protect the master-target privileges and permissions. While remaining reprogrammable and flexible, CSAC authenticates the programming entity using an HMAC based on the SHA-2 protocol. Besides, CSAC is able to detect changes on the internal look-up table due to fault attacks on the privilege rules, applying a smart error detecting mechanism. ASIC and FPGA are then further discussed. Closing Chapter 4, Section 4.6 discusses the use of instantaneous frequency instead of power amplitude and power spectrum in side-channel analysis. By opposition to the constant frequency used in Fourier Transform, instantaneous frequency reflects local phase differences and allows to detect frequency variations. These variations depend on the processed binary data and are hence cryptanalytically useful.

Chapter 5 presents two advancements in theoretical cryptography. Section 5.1 introduces an authentication protocol based on the zero-knowledge paradigm that establishes network integrity, and leverages the distributed nature of computing nodes to alleviate individual computational effort. In that work we describe a distributed Fiat-Shamir 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. Section 5.2 details our proposal of a new authenticated cipher for consideration in the CAESAR competition. Our scheme, called Offset Merkle-Damgård (OMD), is a keyed compression function mode of operation for nonce-based AEAD. We believe that an AE scheme whose security is proved by a modular and easy to verify security reduction, only relying on some widely-verified standard assumption(s) on its underlying primitive(s), can

get more confidence on its security compared to a scheme that demands strong and idealistic properties from its underlying primitive(s) or is not supported by a formal security proof. Provable security helps cryptanalysis efforts to be focused on analyzing the simpler underlying primitives rather than the whole scheme; hence, building confidence on the security of the scheme becomes easier if it uses well-analyzed and verified primitives.

Chapter 6 concludes this thesis and sets future works and advancements.

1.6 Publications

This thesis is primarily based on the publications described in this section. Among the developed work are published papers, e-prints, a book article, two patents, and the submission to the cryptographic competition CAESAR.

Cryptographically Secure On-Chip Firewalling [CHdAdC15, HCPdCOdA15]

with Jean-Michel Cioranesco, Craig Hampel, and Guilherme Ozari de Almeida

Abstract. As SoCs become more complex, on-chip interconnect emerges as the point of integration for a variety of system level functions, including security. Integrators are beginning to rely on distributed access control hardware to protect resources that are shared between IP cores executing both trusted and untrusted software. Existing solutions cover enforcement of on-chip access control policies but they don't secure the programming interface or the hardware against possible attacks. As the embedded content increases in theft value, the on-chip access enforcement will need to consider both software- and hardware-directed attacks. We introduce a secure on-chip access device that enables secure and programmable allocation of resources in an SoC by offering cryptographically-signed programming, fault detection and key integrity. Synthesis results are shown in both ASIC and FPGA implementations.

Note. Published in the 9th International Conference of Network and System Security. This work is presented in detail in Chapter 4.5.

How to (Carefully) Breach a Service Contract? [To appear]

with Céline Chevalier, Damien Gaumont, and David Naccache

Abstract. Consider a firm $\mathcal{S}$ providing support to clients $\mathcal{A}$ and $\mathcal{B}$. The contract $\mathcal{S} \leftrightarrow \mathcal{A}$ stipulates that $\mathcal{S}$ must continuously serve $\mathcal{A}$ and answer its calls immediately. While servicing $\mathcal{A}$, $\mathcal{S}$ incurs two costs: personnel fees (salaries) that $\mathcal{A}$ refunds on a per-call-time basis and technical fees that are not refunded. The contract $\mathcal{S} \leftrightarrow \mathcal{B}$ is a pay-per-call agreement where $\mathcal{S}$ gets paid an amount proportional to $\mathcal{B}$'s incoming calls duration. We consider that the flow of incoming $\mathcal{B}$ calls is unlimited and regular. $\mathcal{S}$ wishes to use its workforce for both tasks, switching from $\mathcal{A}$ to $\mathcal{B}$ if necessary. As $\mathcal{S} \leftrightarrow \mathcal{B}$ generates new benefits and $\mathcal{S} \leftrightarrow \mathcal{A}$ is the fulfilling of a contracted obligation, $\mathcal{S}$ would like to devote as little resources as necessary to support $\mathcal{A}$ and divert as much workforce as possible to serve $\mathcal{B}$. Hence, $\mathcal{S}$'s goal is to minimize its availability to serve $\mathcal{A}$ without incurring too high penalties. This paper models $\mathcal{A}$ as a naïve player. This captures $\mathcal{A}$'s needs but not $\mathcal{A}$'s game-theoretic interests which thorough investigation remains an open question.

Note. Although this paper has an economic background, we have reworked its underlying theory applied to mitigating power consumption on system-on-chips and embedded systems. Here the system $\mathcal{S}$ can choose whether it goes idle to save energy (and be therefore unavailable to answer a request, causing a penalty) or it stays active and drains energy faster. This work is presented in detail in Chapter 3.2.

Applying Cryptographic Acceleration Techniques to Error Correction [GMN+15]

with Rémi Géraud, Diana-Ştefania Maimuţ, David Naccache, and Emil Simion

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 reduction 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 times faster than an improved BCH LFSR design.

Note. Published in the proceedings of *SECITC 2015*. This paper is described in Chapter 3.1.

Public-Key Based Lightweight Swarm Authentication [To appear]

with Simon Cogliani, Rémi Géraud, Diana-Ştefania Maimuţ, and David Naccache

Abstract. This work describes an authentication protocol based on the zero-knowledge paradigm 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.

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.

Note. A detailed version of this paper is presented in Section 5.1.

Buying AES Design Resistance with Speed and Energy [PdCK14, PdCKN16]

with Roman Korkikian, and David Naccache

Abstract. Fault and power attacks are two common ways of extracting secrets from tamper-resistant chips. Although several protections have been proposed to thwart these attacks, resistant designs usually claim significant area or speed overheads. Furthermore, circuit-level countermeasures are usually not reconfigurable at runtime. This paper exploits the AES' algorithmic features to propose low-cost and low-latency protections. We provide Verilog and FPGA implementation details. Using our design, real-life applications can be configured during runtime to meet the user's needs and the system's constraints.

Note. This article is presented in detail in Section 4.4. This work has served as basis of a patent publication [PdCK14] and a book chapter, volume 9100 of the Lecture Notes in Computer Science series [PdCKN16].

Practical Instantaneous Frequency Analysis Experiments [KNdAdC13]

with Roman Korkikian, David Naccache, and Guilherme Ozari de Almeida

Abstract. This paper investigated the use of instantaneous frequency (IF) instead of power amplitude and power spectrum in side-channel analysis. By opposition to the constant frequency used in Fourier Transform, instantaneous frequency reflects local phase differences and allows detecting frequency variations. These variations reflect the processed binary data and are hence cryptanalytically useful. IF exploits the fact that after higher power drops more time is required to restore power back to its nominal value. Whilst our experiments reveal IF does not bring specific benefits over usual power attacks when applied to unprotected designs, IF allows to obtain much better results in the presence of amplitude modification countermeasures.

Note. This novel type of attack is presented in Section 4.6.

Offset Merkle-Damgård (OMD) v.1: A CAESAR Proposal

OMD: A Compression Function Mode of Operation for Authenticated Encryption [CMN+14, CAE]

with Simon Cogliani, Diana-Ştefania Maimuţ, David Naccache, Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár

Abstract. This work describes a cipher for nonce-based authenticated encryption with associated data (AEAD), submitted to the CAESAR competition. Our proposal, called Offset Merkle-Damgård (OMD), is a mode of operation for a keyed compression function. If the compression function at hand does not have a dedicated input for the key then it must first be keyed by some conventional method, for example, by prepending the key to the message in the input of the compression function.

Compression functions are among the most widely-used and well-analyzed cryptographic primitives. We have a rich source of secure and efficient compression functions thanks to more than two decades of public research and standardization activities on hash functions including the recently completed NIST competition for SHA-3. This motivates one to build an AEAD scheme based on a compression function. The recent announcement by Intel in July 2013 about introduction of new instructions, supporting performance acceleration of the Secure Hash Algorithm (SHA) on Intel processors (in particular, for SHA-1 and SHA-256), further encourages the decision to design a compression function based scheme.

OMD takes advantage of the aforementioned facts about compression functions: OMD is provably secure in the standard model based on a well-established security property of the underlying compression function, and it has promising performance using a compression function from the SHA family, thanks to the new Intel SHA Extensions. As specific instantiations of OMD, we recommend two specific compression functions to be keyed and used in OMD, namely, the compression functions of SHA-256 and SHA-512. OMD parameterized with these two compression functions is called OMD-SHA256 and OMD-SHA512, respectively. The former is intended for 32-bit implementations and is our main recommended cipher for CAESAR, while the latter could be used specifically for 64-bit machines.

OMD achieves nearly optimal performance in terms of the number of compression function calls that one can expect from any AEAD scheme *solely* using a compression function. OMD has several attractive features: (i) unlike the permutation-based schemes whose security proof relies on idealistic assumptions about their underlying permutation, the security of OMD is proved in the standard model based on merely the classical PRF assumption on the compression function, (ii) one can easily get a high quantitative level of security using OMD with the compression function of a standard hash function with a large hash size (e.g. 256 bits or 512 bits) while in block-cipher based schemes using AES the block length (affecting the security level) is limited to 128 bits, (iii) the only operations that OMD needs in addition to its core compression function are the basic bitwise XOR and bitwise AND operations on two binary strings and (left and right) shift operation on a binary string, (iv) selecting the core compression function to be that of SHA-256, our primary algorithm OMD-sha256 can take advantage of the new Intel instructions for a highly efficient software implementation.

Note. The OMD authenticated cipher is described in detail in Chapter 5.2. The compression functions of SHA-256 and SHA-512 are recalled in Appendix B.

CHAPTER 2

FROM A TRANSISTOR TO A CRYPTOSYSTEM

Summary

The simplest and most important semiconductor element is the single-crystal silicon, Si. Silicon is the 14th element in the periodic table and has four outer bounding electrons. It forms a covalent structure in a shape of a tetrahedron. Silicon is the basis of the transistor technology, and therefore, electronics.

This chapter draws a brief introduction to modern CMOS, from logic design and the physical characteristics of transistors, to the formal definition of a hardware cryptosystem and techniques on how to implement them efficiently. This chapters also studies the AES cryptosystem and SHA-2 hash function family in further detail, since these two cryptographic blocks will be an important part of the next chapters.

Section 2.1 introduces the digital integrated circuit technology and explains the CMOS physical properties, focusing on the transistor power dissipation. Section 2.2 presents the formal definition of a hardware-based cryptosystem, as well as hardware design architecture metrics and logic design styles. Section 2.3 discusses the AES public-key block cipher in detail. Section 2.4 presents the SHA-2 cryptographic hash function and its implementation tradeoffs.

2.1 Integrated Circuit and Logic Design

2.1.1 Introduction

In 1948, the most significant step in modern electronics happened with the invention of the *transistor* by Bell Laboratories. Together with the solid-state diode, the *transistor* opened the door to microelectronics. The term *microelectronics* itself is defined as the area of technology applied to the realization of electronic systems made of extremely small electronic parts or components, and it is often related to the term *integrated circuits* (IC).

Gordon Moore, co-founder of Intel, published an article in 1965 [Moo00] stating that the number of components per integrated circuit would double approximately every two years. This observation is broadly known as *Moore's Law*. This law proved valid since 1965 and, equally importantly, has guided the microelectronics industry's research and development.

Many other engineering fields involve tradeoffs between performance, power and price. Nevertheless, as digital circuit technology nodes shrink, transistors become faster, consume less power and are typically cheaper to manufacture [WH10]. This unique characteristic of integrated circuits has not only revolutionized electronics, but also shaped our lives. The fast-growing pattern of integrated circuits is depicted in Table 2.1.

Table 2.1: Trends in microelectronics in the past decades [WH10].

Year	1985	1993	2004	2010
Transistor Counts	10^5 - 10^6	10^6 - 10^7	10^8 - 10^9	10^9 - 10^{10}
Clock Frequencies	10^7	10^8	10^9	10^9
Worldwide Market	$25B	$60B	$170B	$250B

More than 80% of the microelectronics industry is composed by digital circuits [Raz06]. Commercially-wise, the majority of the cryptographic systems are made of digital chips. This chapter draws the line from the basic transistor and its power characteristics until the theory behind logic gates that constitute a cryptosystem.

2.1.2 VLSI Design

Digital VLSI design is commonly divided into five abstraction levels [WH10]:

— *architecture* design, that describes how the system operates;
— *microarchitecture* design, that details how the system architecture is partitioned into registers and functional units;
— *logic* design, that specifies how functional units and sub-modules are constructed
— *circuit* design, that describes how transistors are used to implement the logic; and
— *physical* design, that contains the final layout of the chip.

An alternative way of seeing the digital design partitioning is shown in Fig. 2.1. The radial lines on the $\mathcal{Y}$-chart represent three design domains: behavioral, structural and physical. These domains can be used to describe the different phases of a digital design that create a level of design abstraction that starts at a very high level and descend eventually to the individual components that form the top level abstraction.

The Behavioral Domain describes the system's functionality and contains static and dynamic components. The static component describes the operations, while the dynamic portion describes their sequencing and timing. Thus, differences in algebraic functionality, pipelining, and timing are all changes in behavior. For example, at the Functional Block Level, the Behavioral Domain describes the design in terms of register transfers, although the behavioral description might also contain timing information.

The Structural Domain describes the design's abstract implementation, usually represented by a structural interconnection of a set of abstract modules. For example, at the Functional Block Level, the

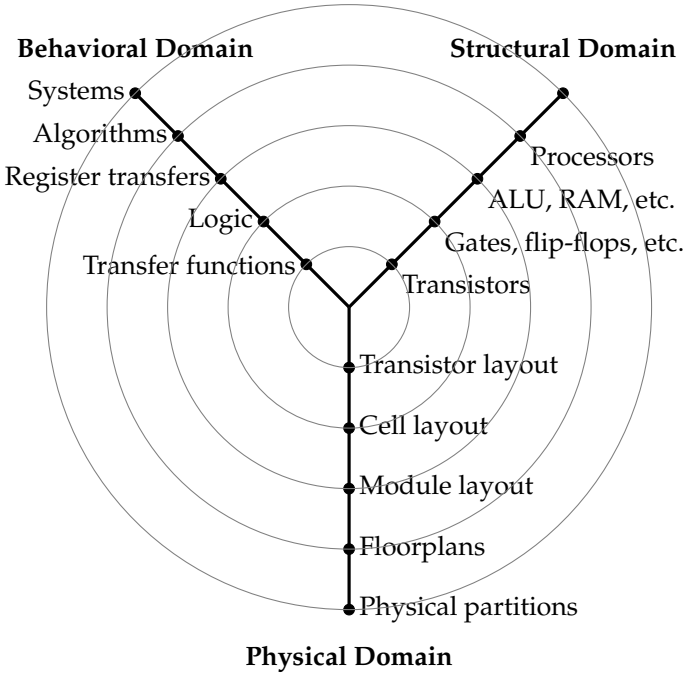


Figure 2.1: Gajski-Kuhn $\mathcal{Y}$-chart [GK83].

Structural Domain describes the design in terms of ALUs, MUXes, and registers. Nevertheless, this domain also describes the digital system's control logic portion.

The Physical Domain describes the design's physical implementation, changing the abstract structural components into physical components. The domain deals with constraints on the physical partitioning and design's layout, as well as physical component geometry. As an example, at the Functional Block Level, the Physical Domain describes the floorplanning that the circuit must have in order to implement the required logic from upper level descriptions. In this domain, speed, power and area constraints are measured and better evaluated than in higher domains.

The growing adoption of higher level of abstraction and complexity of a digital design pushed academia and industry to create Hardware Description Language (HDL) tools to describe both the behavior and the structure of the design, including physical and geometrical information as well [WT85].

Another way of structuring the different types of integrated circuit design is by categorizing according to the design methodology (also known as design styles) [GBC+96]. These styles are primarily based on the viability of the IC design and its target application. The following parameters typically define the choice of a particular design style [WH10]:

— performance – speed, power, system flexibility;
— size of die (hence, die cost);
— time to design (hence, engineering cost);
— ease of verification and test generation (hence, engineering cost).

Since the whole IC design process is complex and long, VLSI designers have to choose the most suitable design style to match its requirements and specifications. The use of design constraints as listed above helps the designer to opt for the best silicon approach. The design methodology of a digital VLSI is commonly divided in two main categories: *Standard Circuit* and *Application-Specific Integrated Circuit* (ASIC). The most important (and commonly used) design styles are listed as a tree in Fig. 2.2.

For several reasons, the best approach to solve a system design problem might be to use a standard architecture, such as a microprocessor or digital signal processor (DSP). Market solutions with built-in RAM or EPROM are broadly available in the market. Microprocessors provide a great level of flexibility, basically transforming the hardware into software design. This methodology is called *hardwired* and

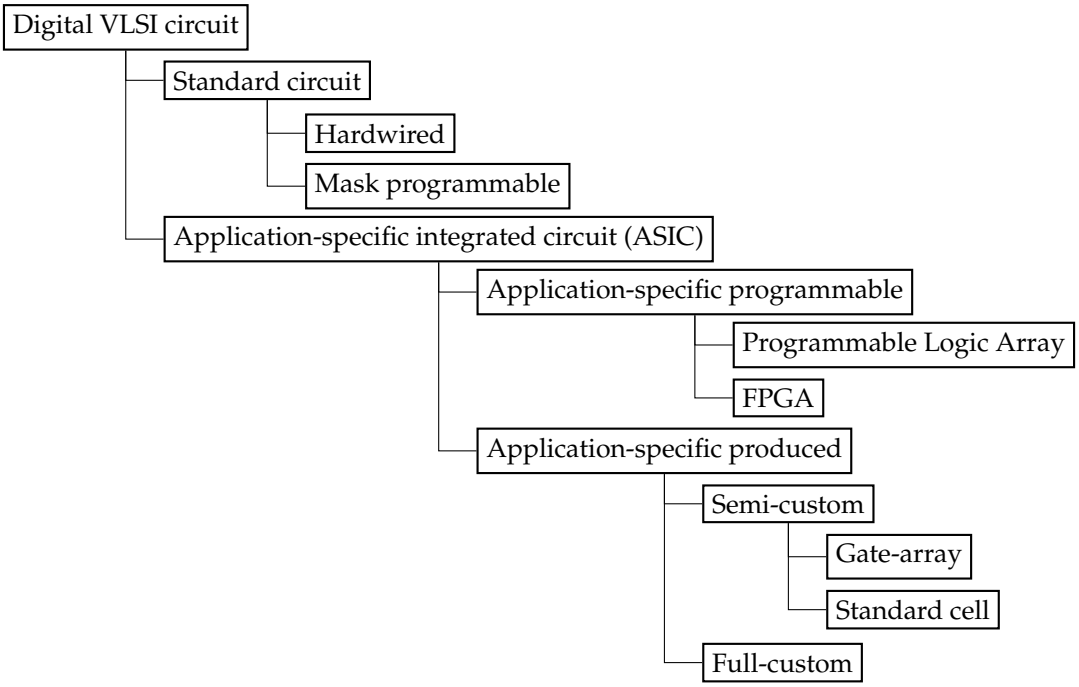


Figure 2.2: The different design styles of a digital VLSI circuit [GBC+96].

represents one type of *standard circuit* design style. The second *standard circuit* type is called *mask programmable* logic and represents the type of read-only memory whose contents are programmed by the IC manufacturer rather than the VLSI designer. This design style typically uses rewritable non-volatile memory (such as EPROM) for the project's development phase, switching to the masked version of the design when the code is finalized.

Often, the cost, performance or power consumption of a microprocessor do not achieve the system's requirement. The next VLSI circuit family that meets a more dense, faster circuit is the *application-specific programmable* logic. In this family, the most important design style is represented by the *Programmable Logic Array* (PLA). This type of chip implements two-level sum-of-product programmable logic arrays with limited routing capability, consisting of an AND and an OR logic gate to compute any function expressed as a sum of products. Any transistor of the AND and the OR gates are capable of being programmed to be present or not. Besides that, a PLA cell contains a NOR structure programmable by a floating-gate transistor, a fusible link, or a RAM-controlled transistor.

The second type of *application-specific programmable* logic family is the *Field Programmable Gate Arrays* (FPGA). FPGAs are a completely reprogrammable IC even after the chip is shipped to final customer. They consist of an array of logic cells, or blocks, surrounded by programmable routing resources. Although first generation FPGAs used fuses or anti-fuses to program the internal blocks and personalize their logic, second generation FPGAs use static RAM or flash memory to configure the routing and logic functions. The first generation is one-time programmable, while the latest FPGA family allows reprogrammability.

Second generation FPGAs are composed of configurable logic blocks (CLB). Among consecutive CLBs, metal tracks are placed vertically and horizontally, forming the configurable routing paths between blocks. CLBs use programmable lookup tables to compute any logic function of several inputs and outputs. Configurable I/O cells surround the core array of CLBs. The main benefit of choosing an FPGA is that it provides the latest technology node with millions of logic gates that easily operate at rates of gigabits per second. They frequently embed extra components for system integration, such as microprocessors, external memory, and hardware accelerators.

The logic styles presented so far do not require a fabrication run. Instead, the VLSI designer can focus on design only, diminishing the non-recurring engineering (NRE) cost. Another way of doing so is to make use of a *semi-custom* logic called *gate-array* (or *sea-of-gates*). This method consists of constructing a chip with a common base array of nMOS and pMOS transistors and later personalizing it by altering

the metallization (metal and via masks) placed on top of the transistors. Differently than *logic array* style, the *gate-array* logic is reprogrammable (like FPGAs). This allows the VLSI designer to correct logic errors and avoid process variability that might cause late bugs and delay the project.

Standard cell logic style represents another *semi-custom* IC type. In this design, a technology IC digital library provided by a foundry or library vendors is available as the basic building blocks of the chip. Also called *cell-based design*, this design methodology achieves smaller, faster and low-power chips than FPGAs, but incurs higher NRE costs to produce the custom mask set. *Standard cell* design is therefore only suitable for high volume fabrication.

Finally, a *full-custom* design is another type of *application-specific produced* IC, where all circuit details have to be designed, all transistors have to be dimensioned and the layout have to be carefully drawn and verified by an analog simulator afterwards. In other words, a *full-custom* design and production are fully controlled by the VLSI designer. Even though it allows optimal results, the design effort is highly prohibitive for large ICs. A classical example of a *full-custom* design is the commercial microprocessor, in which top-notch performance is a requirement.

Given all the IC families described above, a VLSI design choice must be based on the most cost-effective approach taking into account speed, power and area figures. If off-the-shelf solutions (microprocessors, microcontrollers, for example) meet the designer's requirement targets, they should always be preferred. If not, FPGA is the next logic candidate, especially for low-volume ($< 100,000$) applications. When the production volume is high enough to justify the costs, or when low power is at stake, *standard cell* design is the preferable option. Mixed-signal, radio-frequency (RF), and high speed digital designs are examples of IC that require a cell-based design approach. Table 2.2 summarizes the design methodologies cost in terms of several different criteria, from low to very high.

Table 2.2: Comparison of VLSI design methodologies [WH10].

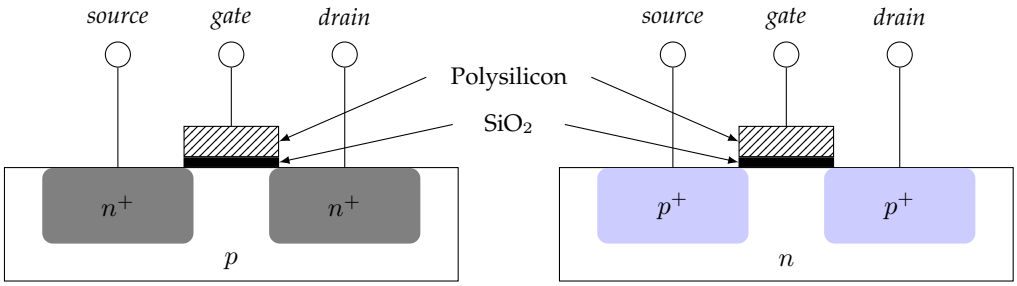
Design style	NRE	Unit cost	Power	Design complexity	Time to market	Performance	Flexibility
Hardwired & mask programmable	low	medium	high	low	low	low	high
PLA	low	medium	medium	low	low	medium	low
FPGA	low	medium	medium	medium	low	high	high
Gate-array	medium	low	low	high	medium	high	high
Standard cell	high	low	low	high	high	high	low
Full-custom	high	low	low	high	high	very high	low

2.1.3 The CMOS Transistor

A transistor can be viewed as a device with 3 terminals, or a 2-input/1-output switch. One input, the control, acts by enabling or disabling the current transfer from the other input to the output, depending on the voltage applied to the control terminal. Early integrated circuits first adopted the bipolar junction transistor, developed by Bell Labs. The next generation came to production in 1960, replacing the bipolar transistor: the Metal Oxide Semiconductor Field Effect Transistor (MOSFET). MOSFETs have three major advantages:

— draining almost zero current while idle;
— their miniaturization could be more easily achieved; and
— consuming only nanowatts of power, six orders of magnitude less than their bipolar counterparts.

MOSFETs belong to two sub-types, differing by the silicon substrate type: n-type (called nMOS transistors) and p-type (called pMOS transistors). That is the reason that they became to be named Complementary Metal Oxide Semiconductors, or CMOS.



(a) The nMOS transistor.

(b) The pMOS transistor.

Figure 2.3: The two types of CMOS devices.

Each CMOS transistor consists of a conducting *gate*, an insulating layer of silicon dioxide (SiO_2, also called *glass*) and the silicon *substrate*, also known as *body* or *bulk*, that is usually grounded. An nMOS transistor, depicted in Fig. 2.3a, is built on a p-type *body* and has regions of n-type semiconductor adjacent to the *gate* called the *source* and *drain*. A pMOS transistor, as shown in Fig. 2.3b, is the opposite, having p-type *source* and *drain* regions on a n-type *body*. In a CMOS technology where both transistor types are present, the substrate is either n- or p-type. To build the other transistor type, a special *well* with dopant atoms has to be added to form the opposite body type. The n^+ and p^+ regions indicate heavily doped n- or p-type silicon.

A voltage applied to the *gate* of the CMOS device controls the current flowing between *source* and *drain*. More specifically, the nMOS transistor creates a current flow from *source* to *drain* when a positive voltage is applied to the *gate*. Otherwise it acts as an open switch, i.e., no current flows through the two opposite terminals, and the nMOS transistor is OFF. The *gate* functions as a control input: it acts on the electrical current flow between the *source* and *drain*. As the *gate* voltage raises, it creates an electric field that attracts free electrons to the underside of the Si–SiO_2 interface. If the voltage raises enough, the electrons outnumber the holes and a thin region between the two terminal is created, transformed into an n-type semiconductor. A conducting path of electrons flow from *source* to *drain*, and we say that the transistor is ON.

In the pMOS, the behavior is the again opposite. By applying a positive voltage, the body is held reverse-biased and no channel is created, therefore the transistor remains OFF. When the *gate* voltage is lowered, positive charges are attracted to the underside of the Si–SiO_2 interface. A sufficiently low gate voltage inverts the pMOS channel and a conducting channel is formed from *source* to *drain*, and the transistor is ON.

The voltage value at which a conducting channel is created between *source* and *gate* is called the threshold voltage. The positive voltage is usually called V_{DD} or *POWER* and its value varies depending on the technology library. In popular logic families of the 1970s and 1980s, V_{DD} was usually set to 5 volts, whereas nowadays voltages around 1 volt predominate. The low voltage is called *GROUND* or V_{SS} and its value is usually 0 volts. Bringing it to the digital world, V_{DD} is referred to a logic (or bit) 1, and V_{SS} represents the logic 0. Fig. 2.4 (where *g*, *s*, *d* stand for *gate*, *source* and *drain*, respectively) shows the symbol of each CMOS transistor and their "switch" behavior depending on the voltage applied to the *gate*.

2.1.4 CMOS Logic

Basic and complex CMOS logic gates are constructed with two networks: the upper part, called *pull-up* network, connects to the power supply (V_{DD}); the lower part, called *pull-down* network, connects to ground (V_{SS}). Both are connected together to the output of the logic gate. For the logic to work as expected, the *pull-up* network is composed by pMOS transistors only, while the *pull-down* counterpart contains only nMOS transistors. Fig. 2.5 depicts the general schematic of a general CMOS logic gate.

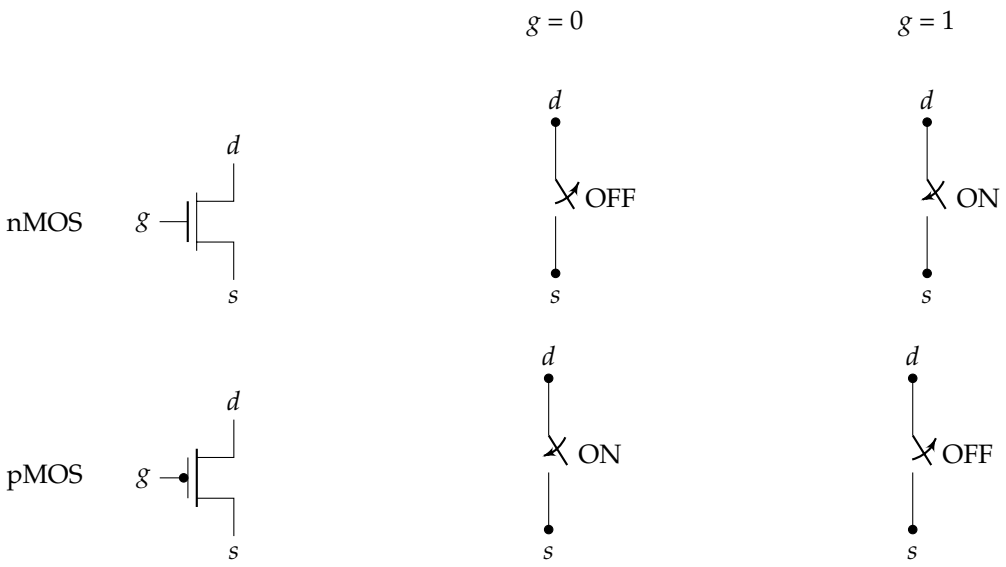


Figure 2.4: CMOS transistor symbols and switch levels.

The *pull-up* and *pull-down* networks in the inverter each consist of a single transistor. The NAND gate uses a series *pull-down* network and a parallel *pull-up* network. More elaborate networks are used for more complex gates. Two or more transistors in series are ON only if all of the series transistors are ON. Two or more transistors in parallel are ON if any of the parallel transistors are ON. By using combinations of these constructions, CMOS combinational gates can be constructed.

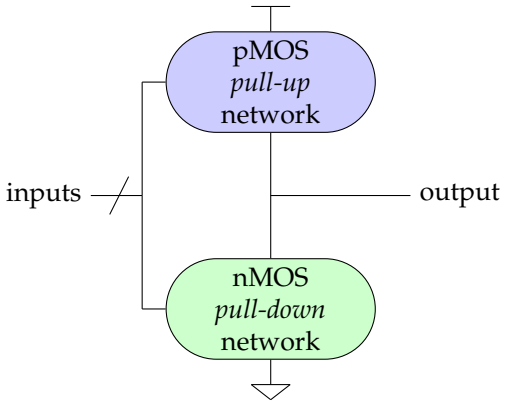


Figure 2.5: General logic gate using *pull-up* and *pull-down* networks.

In general, when we join a *pull-up* network to a *pull-down* network to form a logic gate as shown in Fig. 2.5, they both will attempt to exert a logic level at the output. From this table it can be seen that the output of a CMOS logic gate can be in four states. The 1 and 0 levels have been encountered with the inverter and NAND gates, where either the pull-up or pull-down is OFF and the other structure is ON. When both pull-up and pull-down are OFF, the high-impedance or floating Z output state results. This is of importance in multiplexers, memory elements, and tri-state bus drivers. The crowbarred (or contention) X level exists when both pull-up and pull-down are simultaneously turned ON. Contention between the two networks results in an indeterminate output level and dissipates static power. It is usually an unwanted condition.

2.1.4.1 The Inverter

The simplest form of a CMOS logic circuit is an inverter, that represents the logic formula out $= \overline{\text{in}}$ that consists of a pMOS *pull-up* transistor and an nMOS *pull-down* transistor, as depicted in Fig. 2.6.

When the input voltage $V_{in} = 0$, the gate of the p-channel transistor is at V_{DD} below the source potential $-V_{DD}$, i.e., the pMOS is ON. No current flows through the n-channel transistor, that is OFF. If we increase V_{in} to the threshold voltage and then to V_{DD}, the operation reverts: the n-channel transistor will conduct while the p-channel is now OFF. It is easy to note that the output of the CMOS inverter will always get the opposite voltage of V_{in}.

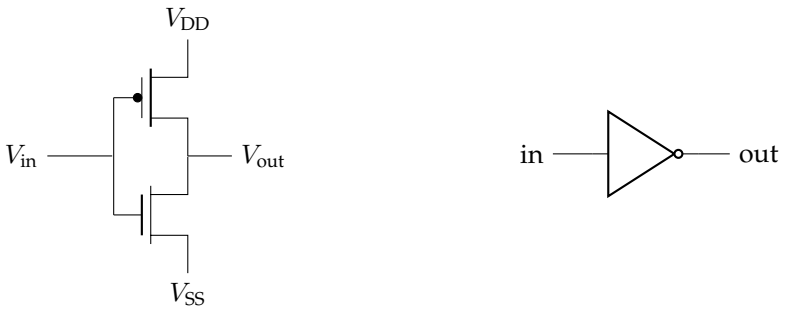


Figure 2.6: CMOS inverter gate schematic and symbol.

2.1.4.2 The NAND Gate

The CMOS NAND gate contains two series of nMOS transistors between the output and V_{SS} and two parallel pMOS transistors between the output and V_{DD}. A NAND gate consists of two inputs, A and B, following the formula out $= \overline{\text{in1.in2}}$. Fig. 2.7 shows the schematic and symbol of a CMOS NAND gate.

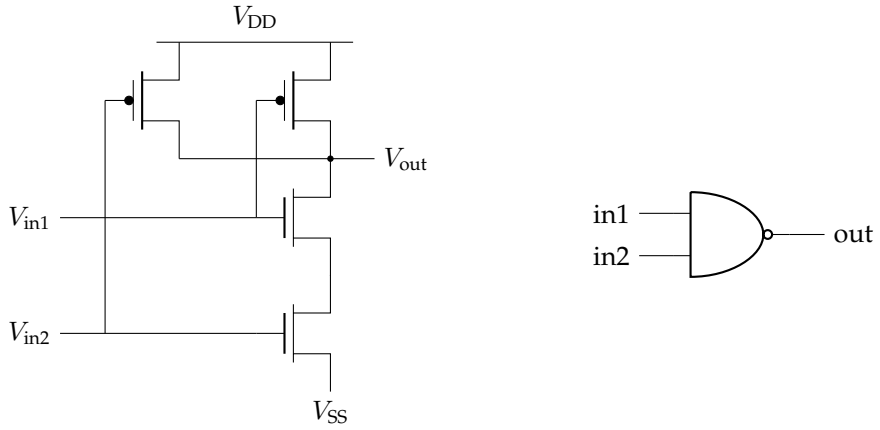


Figure 2.7: CMOS NAND gate schematic and symbol.

The NAND gate is an important metric when evaluating the actual complexity of a design. A common digital integrated circuit design complexity factor is measured by number of logic gates, that eventually reflect the size of the overall circuit. The problem arises from the fact that measuring an ASIC circuit area makes it difficult to compare to another ASIC design targeted to a different technology node. Even among FPGA implementations, the internal FPGA's look-up tables and surrounding logic are shrunk to a given process technology. The size of the NAND gate at a given technology gives a technology-independent way of measuring the design complexity, and also a fair comparison between two circuit designs. Instead of comparing logic instances, the total circuit area (technology-dependent) is divided by the smallest 2-input NAND gate of the digital IC library. The result can be fairly compared to any other circuit. Some engineers prefer to use the total circuit area (logic + routing), while some others do not use routing information as it can vary from one CAD tool to another. Besides that, a fair comparison is achieved by comparing ASIC designs and FPGA counterparts separately.

2.1.4.3 Compound Gates

The flexibility of CMOS logic is such that a compound gate can be created from different connections and arrangements between the *pull-up* and *pull-down* transistors. This allows more complex logic gates to be tailored to the best possible circuit layout. If a certain logic pattern is highly likely to be used for a digital design, the digital library may incorporate this logic into a basic digital cell. For example, the derivation of the circuit for the function $Y = \overline{(A.B) + (C.D)}$ is shown in Fig. 2.8. This function is called AND-OR-INVERT-22 (AOI22) and it is usually provided by commercial digital library foundries. It performs the NOR of a pair of 2-input NANDs.

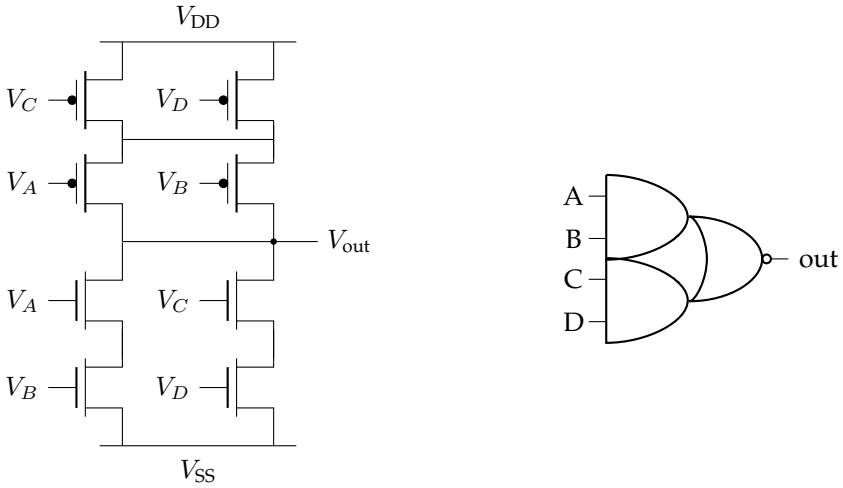


Figure 2.8: CMOS AND-OR-INVERTER-22 schematic and symbol.

2.1.4.4 Tri-state Buffers

In addition to the logic levels 0 and 1, it is possible to construct a CMOS logic gate that has a third logic value, denoted Z. The enable input EN controls whether the primary input is passed to the output or not. If EN $= 1$, the gate acts as a normal buffer. If EN $= 0$, the input is effectively disconnected from the circuit, leaving the output of the gate "floating". A tri-state buffer is usually connected to a bus that allows multiple signals to travel along the same connection, as long as exactly one buffer is enabled at a time. When disconnected, the tri-state buffer output holds neither logic 0 or 1, but instead gives a state of very high impedance. As a result, no current is drawn from the power supply.

A logical first approach of a tri-state buffer is represented in Fig. 2.9 as a transmission gate. It is as simple as two CMOS transistors, but it is imperfect: when EN $= 1$, the output receives the input, however the signal is not restored. If the input is noisy or degraded, the output will receive the same noise, which should be avoided. If several of these gates are along the same logic path, the overall path delay highly increases.

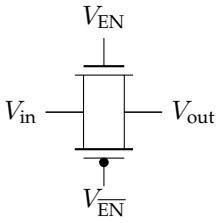


Figure 2.9: CMOS transmission gate.

Fig. 2.10 shows the tri-state buffer circuit schematic and logic symbol. This tri-state is actually implemented as a tri-state inverter that produces a restored output, although inverted. For a non-inverter

tri-state buffer, we have to add an inverter in front of the actual circuit. When outputs are tri-stated, the influence of that gate on the rest of the circuit is removed. This behavior could be exploited to create lightweight countermeasures against power attacks, as presented in detail in Section 4.4.

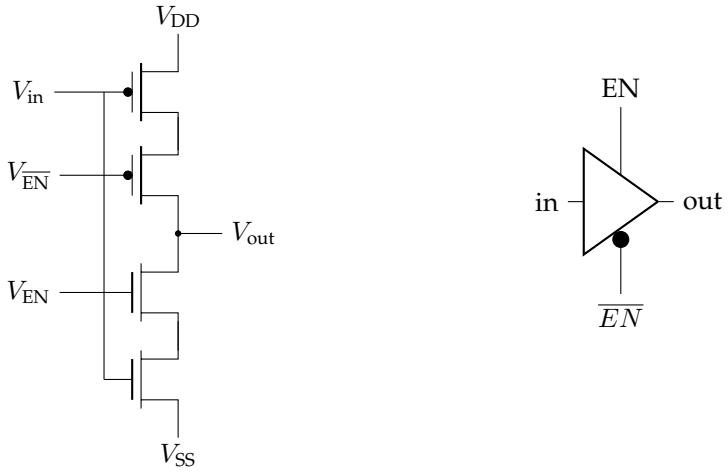


Figure 2.10: CMOS tri-state buffer schematic and symbol.

2.1.5 CMOS I-V Characteristics

A signal strength is defined as a measure of how closely the signal approximates an ideal voltage source, i.e., the stronger the signal, the more it is able to drive higher current. In the CMOS technology, the strongest signals are V_{DD} (logic 1) and V_{SS} (logic 0).

By analyzing the two basic CMOS components, the pMOS and the nMOS transistors, we can find the best transmitter. In fact, they are both good and bad voltage sources. This comes form the fact that the nMOS transistors are an almost perfect switch when transmitting a logic 0, but imperfect at passing a logic 1. Therefore we say that the nMOS transistor passes a *strong* 0, but a *weak* 1. The situation is the opposite for the pMOS transistor. The pMOS is capable to be an optimal source of a strong logic 1, but passes a weak logic 0.

In all our examples so far, the CMOS logic gates are composed of nMOS transistors in the pull-down network and pMOS transistors in the pull-up network. Therefore, nMOS components only have to pass logic 0 to the output, and pMOS components only pass the logic 1, so the output is strongly driven. This considerably simplifies the design layout compared to other forms of logic, where the pull-up and the pull-down switch networks have to be ratioed. Instead, CMOS gates operate in the same manner, no matter the technology node size. As a drawback, the design of a CMOS component must be inverting – the nMOS pull-down network turns ON when inputs are at logic 1, delivering logic 0 to the output.

Nevertheless, the characteristics that made the static CMOS technology extremely successful is undoubtedly electrical – because there is never a path through ON transistors from V_{DD} to V_{SS} (in contrast to technologies such as single-channel MOS, GaAs and bipolar), CMOS gates dissipate very low static power.

2.1.5.1 CMOS Electrical Properties

The CMOS transistor is a *majority-carrier* component in which the gate controls the current flowing through a conducting channel from the source to the drain. In an nMOS transistor, the majority carriers are electrons; in a pMOS transistor, the majority carriers are holes. Between the gate pin and the transistor's doped silicon body there is a thin layer of insulating film of SiO_2 called the *gate oxide*. The most important property of the *gate oxide* is that it is a very good insulator, so almost zero current flows from the gate to the body.

While the nMOS source and drain have free electrons, its body contains free holes but no electrons. Applying a gate-to-source voltage V_{gs} that is less than the threshold voltage V_t, the junctions between body and source or drain are zero-biased or reverse-biased, therefore little or no current flows. This transistor state is called *cutoff*. Analyzing only one single component, the small current flowing through a *cutoff* transistor is insignificant, but can become significant if we sum up to several million transistors on a chip. Moreover, the smaller the technology node is, the more predominant the leakage is.

When V_{gs} becomes bigger than V_t, an inversion region of electrons creates a conductive connection between source and drain. Now a small difference between the drain voltage V_{ds} and source voltage V_{gd} makes the current I_{ds} to flow through the channel. This state is called *linear*, *resistive*, *triode*, *nonsaturated* or *unsaturated*. The current increases with both the gate voltage and the drain voltage. In this mode, the transistor acts as a linear resistor in which the current flow is proportional to V_{ds}.

If V_{ds} becomes sufficiently large that $V_{gd} < V_t$, the nMOS conductive channel is no longer inverted and becomes *pinched-off*. Electrons are still being injected towards the drain, however the channel connection with the drain is cut, and the gate voltage is the only voltage that controls the current flowing from source to drain. This state is called *saturation*.

In summary, the nMOS transistor has three operational modes:

— *cutoff* – no channel is formed, $V_{gs} < V_t$ and $I_{ds} = 0$;
— *linear* – channel is formed between source and drain, $V_{gs} > V_t$ and I_{ds} increases with V_{ds};
— *saturation* – channel is pinched-off, $V_{gs} > V_t$ and I_{ds} is independent of V_{ds}.

The pMOS counterpart works analogously, but in an opposite fashion. In the pMOS, the n-type body is tied to a high potential, therefore the p-type source and drain are naturally reversed-biased. When the pMOS gate voltage is also at a high potential, no current flows between drain and source. When the gate voltage $V_{gs} < V_t$, holes are attracted to form a p-type channel beneath the gate that allows current flow. The threshold voltages of nMOS and pMOS are not necessarily equal.

The *Shockley* model. The first CMOS transistor model [Sho52] relating the current and voltage (I-V) is known as *Shockley* model. The model assumes that the channel length is long enough that the lateral electric field (the field between source and drain) is relatively low, which does not hold for nanometer technologies. Besides, the *Shockley* model does not describe leakage consumption and other nanometer anomalies.

To compute the current that flows through an ON nMOS transistor ($V_{gs} > V_t$), we have to first compute the amount of charge in the transistor channel and also the rate at which the current moves. Given that the capacitor charge is defined as $Q = CV$, the charge in the channel can be defined as

$$Q_{\text{channel}} = C_g(V_{gc} - V_t)$$

where C_g is the gate capacitance and $V_{gc} - V_t$ is the potential to attract the charge. The nMOS gate can be modeled as a parallel plate capacitor with capacitance proportional to area over thickness. Let the nMOS gate length be L, the gate width be W and the oxide thickness be t_{ox}. Therefore the capacitance C_g can be written as

$$C_g = \kappa_{ox}\varepsilon_0\frac{WL}{t_{ox}} = \varepsilon_{ox}\frac{WL}{t_{ox}} = C_{ox}WL$$

where ε_0 is the permittivity of free space, κ_{ox} is the permittivity of SiO_2 and C_{ox} is the capacitance per unit area of the gate oxide.

We define the electric field E as the voltage difference between drain and source (V_{ds}) divided by the channel length

$$E = \frac{V_{ds}}{L}$$

and the average velocity υ at which each channel carrier is accelerated as

$$\upsilon = \mu E$$

where μ is called the *mobility*. Therefore, the current between source and drain is the total amount of charge in the channel divided by the time required to cross

$$\begin{aligned} I_{\text{ds}} &= \frac{Q_{\text{channel}}}{L/\upsilon} \\ &= \mu C_{\text{ox}} \frac{W}{L} (V_{\text{gs}} - V_{\text{t}} - \frac{V_{\text{ds}}}{2}) V_{\text{ds}} \\ &= \beta (V_{\text{GT}} - \frac{V_{\text{ds}}}{2}) V_{\text{ds}} \end{aligned}$$

and therefore

$$\beta = \mu C_{\text{ox}} \frac{W}{L}$$

with

$$V_{\text{GT}} = V_{\text{gs}} - V_{\text{t}}$$

The factor β merges the geometry and technology-dependent parameters.

When the transistor is *pinched-off*, the channel is no longer inverted at the drain and therefore $V_{\text{ds}} > V_{\text{GT}}$. At this point, increasing the drain voltage has no effect on the current. Replacing V_{ds} by V_{GT}, we find the expression for the saturation current

$$I_{\text{ds}} = \frac{\beta}{2} {V_{\text{GT}}}^2$$

To summarize, the I-V characteristics of an nMOS transistor are defined in terms of its three states:

$$I_{\text{ds}} = \begin{cases} 0 & V_{\text{gs}} < V_{\text{t}} \quad \textit{cutoff} \\ \beta (V_{\text{GT}} - \frac{V_{\text{ds}}}{2}) V_{\text{ds}} & V_{\text{ds}} < V_{\text{GT}} \quad \textit{linear} \\ \frac{\beta}{2} {V_{\text{GT}}}^2 & V_{\text{ds}} > V_{\text{GT}} \quad \textit{saturation} \end{cases}$$

Fig. 2.11 shows the transistor's I-V characteristics. According to the first-order *Shockley* model, the current is zero when gate voltage is below V_{t}. By increasing the gate potential, current increases linearly with V_{ds} for small V_{ds}. As V_{ds} reaches the saturation point ($V_{\text{ds}} = V_{\text{GT}}$), current becomes independent of V_{ds}.

The pMOS transistor presents a similar behavior, although the voltages present negative values and the currents are reversed. The pMOS smaller current amplitudes compared to the nMOS are due to lower mobility of holes compared to the mobility of electrons. Hence, a pMOS transistor of the same size of an nMOS counterpart is inherently slower.

2.1.5.2 Non-Ideal I-V Effects

The *Shockley* model neglects several important effects that are important for devices with channel lengths below 1 micron. This section summarizes the most important sub-micron effects on CMOS transistors.

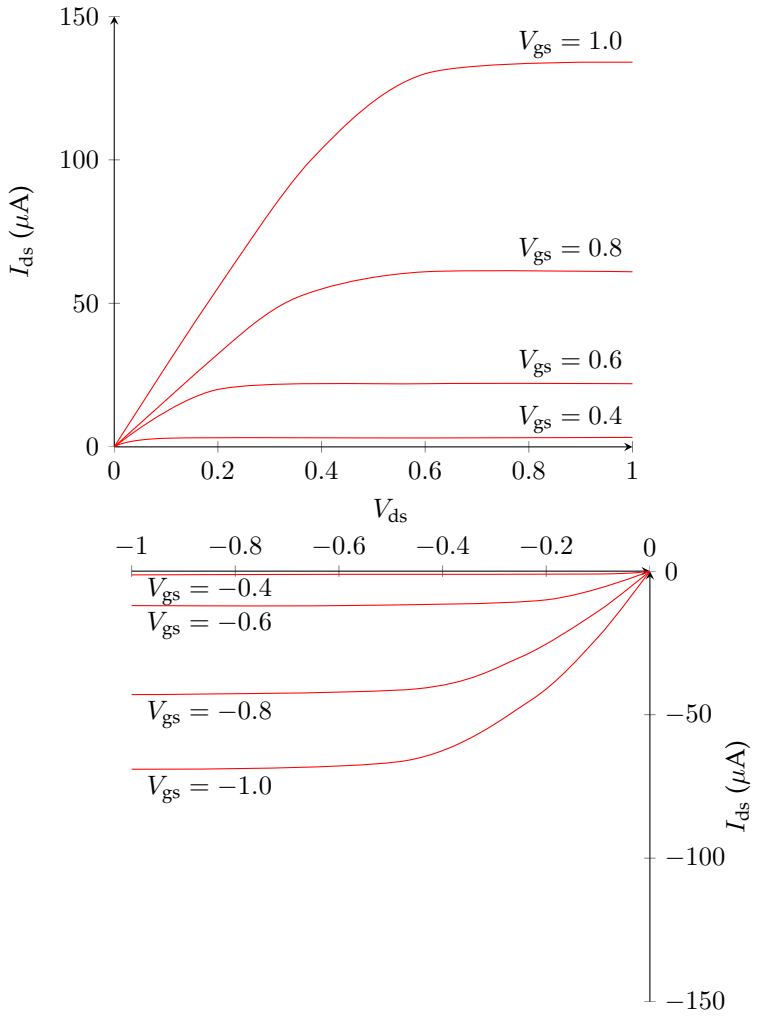


Figure 2.11: I-V characteristics of ideal nMOS (upper graph) and pMOS (lower graph) transistors [WH10].

Mobility Degradation and Velocity Saturation. It has been defined that current is proportional to the lateral electrical field $E_{lat} = V_{ds}/L$ between source and drain. The previous model assumed that the carrier mobility μ is independent of the applied fields. While this model holds for low fields, it breaks down when strong lateral or vertical fields are applied.

Mobility degradation is the effect that causes carriers to collide with the oxide interface when a high voltage at the gate of the transistor attracts the carriers to the edge of the channel. The faster carriers go, the faster they collide. Carriers approach a maximum velocity v_{sat} when high fields are applied. The limiting of carrier velocity at high field is defined as velocity saturation. According to [TKM88], the velocity can be approximated by:

$$v = \begin{cases} \dfrac{\mu_{eff} E}{1 + \frac{E}{E_c}} & E < E_c \\ v_{sat} & E \geq E_c \end{cases}$$

By continuity, the critical electric field is

$$E_c = \frac{2 v_{sat}}{\mu_{eff}}$$

And the critical voltage (the voltage in the drain-source at which the critical effective field is reached) is

$$V_c = E_c L$$

We can now derive the new I_{ds} formulae of linear and saturation transistor states from Section 2.1.5.1 taking into account the velocity saturation:

$$I_{ds} = \begin{cases} \frac{\mu_{eff}}{1+\frac{V_{ds}}{V_c}} C_{ox} \frac{W}{L} (V_{GT} - V_{ds}/2) V_{ds} & V_{ds} < V_{GT} \quad \textit{linear} \\ C_{ox} W (V_{GT} - V_{dsat}) \upsilon_{sat} & V_{ds} > V_{GT} \quad \textit{saturation} \end{cases} \tag{2.1}$$

As the lateral field grows, the current saturates, and $V_{ds} = V_{dsat}$. Equating the two parts of above equation 2.1 we can find the saturation voltage

$$V_{dsat} = \frac{V_{GT} V_c}{V_{GT} + V_c} \tag{2.2}$$

Equation 2.2 allows us to find the saturation current when $V_{ds} > V_{sat}$

$$I_{dsat} = W C_{ox} \upsilon_{sat} \frac{{V_{GT}}^2}{V_{GT} + V_c}$$

Channel Length Modulation. As previously discussed, a transistor in saturation mode presents a depletion region between the drain and the body, and the channel length is effectively shortened by this effect.

Assuming that the body voltage $V_{db} \approx V_{ds}$, increasing V_{ds} decreases the effective channel length. I_{ds} increases with V_{ds} in saturation, therefore shorter channel length results in higher current. As defined by [GM90], we can model the channel length factor by multiplying the *Shockley* model current formula by $(1 + V_{ds}/V_A)$, where V_A is called the *early voltage*. Therefore, in the saturation region, we can define

$$I_{ds} = \frac{\beta}{2} {V_{GT}}^2 (1 + \frac{V_{ds}}{V_A})$$

Threshold Voltage Effects. Until now, the threshold voltage V_t was treated as a constant. Nevertheless, V_t

- increases with the source voltage,
- decreases with the body voltage,
- decreases with the drain voltage, and
- increases with channel length [RMMM03].

To model the body effect on the threshold voltage, we have to consider that the body actually acts as a fourth transistor terminal. A voltage V_{sb} is applied between the source and body increases the amount of charge required to invert the channel, therefore it increases V_t. The threshold voltage can be modeled as

$$V_t = V_{t0} + \gamma(\sqrt{\phi_s + V_{sb}} - \sqrt{\phi_s})$$

where V_{t0} is the threshold voltage when the source is at the body potential, ϕ_s is the surface potential at threshold, γ is the body effect coefficient.

Leakage. The ideal CMOS model considers that OFF transistors are open circuits, therefore no current between source and drain. However, even when turned off, transistors leak small amounts of power, called *leakage power*. Three components are present in leakage consumption: *subthreshold conduction*, due to the fact that there is a thermal carrier emission over the potential barrier set by the threshold; *gate leakage*, a quantum-mechanical effect caused by tunneling through the thin insulator between source and body; and *junction leakage*, caused by small currents that flow from the source and drain wells to the body via the p-n junction.

Processes above 180nm present insignificant leakage when compared to the dynamic power consumption. Between 90 and 65nm processes, leakage can achieve 1 to 10nA per transistor, which becomes significant when multiplied by the number of transistors on a chip. The trend we observe is that, as the technology node gets smaller, the leakage component is reaching the same magnitude of the dynamic power dissipation.

Temperature Dependence. The main impact from the temperature dependence is that the transistor carrier mobility decreases as temperature raises. The threshold voltage decreases almost linearly with temperature and may be approximated by

$$V_{\mathrm{t}}(T) = V_{\mathrm{t}}(T_{\mathrm{r}}) - \kappa_{\mathrm{vt}}(T - T_{\mathrm{r}})$$

where T is the absolute temperature, T_{r} is the room temperature, and κ_{vt} is the fitting parameter typically about 1-2mV/K.

2.2 Hardware-Based Cryptosystems

2.2.1 Introduction

During and after World War II, the adoption of purely electronic cryptosystems was made possible by the introduction of logical circuits [Til99]. Traditionally, Digital Signal Processing (DSP) algorithms have been implemented in software, on specifically designed microprocessors, typically driven by their low cost [MM03]. Often the performance of software-based DSPs failed to meet the requirements. The market naturally adopted alternative solutions to cope with the higher standards: eventually, custom hardware solutions, such as ASICs and FPGAs, had to emerge to bust inherent software lower performance compared to hardware. Cryptosystems followed the same trend as DSPs. Hardware designs of encryption algorithms are much faster than equivalent software architectures (typically several orders of magnitude).

The majority of current cryptosystems runs on general-purpose microprocessors with a relatively low level of security, alongside with smart-card assisted implementations that store a private key in and perform basically private-key operations [Gut03]. Only a small number of cryptographic implementations run in dedicated hardware. The advantage of a software-only implementation is that they are very inexpensive and easy to deploy (compared to a dedicated hardware approach). The disadvantage is that they provide a very low-level protection for crypto-variables, allowing, for example, passive attacks such as memory scanning in search of crypto-keys, that are stored in the unprotected processor memory. If a symmetric key is compromised, the encrypted data is no longer secure and can be easily read. If an asymmetric private key is accessed, it can be used to intercept and modify messages and to forge digital signatures, which would be attributed to the key's rightful owner. A more sophisticated attack on software-base implementation consists in fooling the system into running malicious code in the most privileged security level (usually reserved for the kernel), or loading malicious selectors that provide access to all physical memory.

Although problems described above persist in software-based systems, these platforms are the majority of the market in which consumers care the most [Gut03]. Since program correctness is difficult to achieve, buggy and insecure systems are the usual state of affairs as we write these lines [CP98]. Another attribute that leads consumers is the software's ease of use feature; users prefer easier and intuitive systems rather than more secure ones. Real security is harder, slower and more expensive to design and to implement. Since consumers are not able to differentiate good security from bad security, the market is driven by better-looking features instead of relying in strong security [SM99, Ros96].

Consequently, instead of trying to unroot insecurity away from the cryptosystem, the correct approach for a more secure system is therefore to push the crypto away from the insecurity. In other words, despite the fact that the software layer of the system may be crowded of trojan horses, computer viruses or other software security threats, none of these can actually come near the crypto environment.

The FIPS 140 standard [Nat06] specifies a number of guidelines for the development of cryptographic secure modules. Originally, this standard only allowed implementations of cryptographic algorithms [DES77]; however, the specification lowered its requirements in mid-1990s, allowing software implementations as well [AES01]. FIPS 140 defines four security levels from 1 (that focus on validating the cryptographic implementation correctness) to 4 (the cryptosystem is validated as a tamper-resistance module). Although FIPS 140 allows software certification, this means that the operating system would have to be certified at the same level of the underlying hardware, which is very difficult (or even impossible) to achieve [Gut03]. As already stated, secure software-only crypto running on a general purpose PC has to protect data present in unprotected memory and can be read by virtually any process.

These reasons, among others, pushed security to hardware-based cryptosystems, where secrets such as the cipher *secret key* or the hash *authenticated tag* are under much higher control within the module. This section will explain in more details the challenges faced by hardware-based cryptosystems. Despite of the fact that they are inherently more secure, other attack types, such as side-channel attacks, take place in this context.

2.2.2 Definitions

A cryptosystem is defined as an encryption system together with a corresponding decryption system. Let M be the encryption system with a nonempty finite set $\{\chi_0, \chi_1, \chi_2, ..., \chi_{\theta-1}\}$ of injective relations $\chi_i : V^{(n_i)} \dashrightarrow W^{(m_i)}$. Each χ_i is called an encryption step.

Definition 2.1 *An encryption system $X = [\chi_{i1}, \chi_{i2}, \chi_{i3}, ...]$ is called finitely generated (by means of the encryption system M) if X is induced by a sequence $(\chi_{i1}, \chi_{i2}, \chi_{i3}, ...)$ of encryption steps $\chi_i \in M$ under the concatenation $\star$, i.e.,*

$x \overset{X}{\dashrightarrow} y$ holds for $x \in V^*, y \in W^*$ if and only if there exist decompositions $x = x_1 \star x_2 \star x_3 \star ... \star x_k$ and $y = y_1 \star y_2 \star y_3 \star ...y_k$ with $x_j \overset{\chi_{ij}}{\dashrightarrow} y_i$ for $j = 1, 2, ..., k$.

Definition 2.2 *The cardinal number of the encryption system M is denoted by $\theta = |M|$. n_i and m_i are defined to be the maximal plaintext and ciphertext widths (bit lengths), respectively. The encryption step is called endomorphic if $V = W$.*

Definition 2.3 *An encryption step $\chi_i : V^{(n_i)} \dashrightarrow W^{(m_i)}$ is finite (provided that V and W are finite) and can be specified in principle by enumeration (encryption table). An actual enumeration is often called, as already mentioned, a* cipher*; the encryption step is then named* encoding step *or* enciphering step.

Definition 2.4 *An encryption $X = [\chi_{i1}, \chi_{i2}, \chi_{i3}, ...]$, finitely generated by M, is* monoalphabetic *if it comprises or uses a single encryption step; otherwise it is called* polyalphabetic.

Definition 2.5 *An encryption $X = [\chi_{i1}, \chi_{i2}, \chi_{i3}, ...]$, finitely generated by M, is said to be* monographic *if all the $n_i = 1$; otherwise it is called* polygraphic. *In a special case of particular interest for encryption by machines, all encryption steps of M show equal maximal encryption width n and equal maximal decryption width m: then M is necessarily finite.*

2.2.3 Hardware Design Architecture

There are three primary definitions of speed depending on the context of the problem: throughput, latency, and timing. In the context of processing data in an FPGA, throughput refers to the amount of data processed per clock cycle. A common throughput metric is bits per second. Latency refers to the time between data input and processed data output. The typical metric for latency will be time or clock cycles. Timing refers to the logic delays between sequential elements. When we say a design does not meet timing, we mean that the delay of the critical path, that is, the largest delay between flip-flops (composed of combinatorial delay, clock-to-output delay, routing delay, setup timing, clock skew, and so on) is greater than the target clock period. The standard metrics for timing are clock period and frequency.

The following metrics are usually discussed when design hardware [Kil07]:

— High-throughput architectures for maximizing the number of bits per second that can be processed by the design.
— Low-latency architectures for minimizing the delay from the input of a module to the output.
— Timing optimizations to reduce the combinatorial delay of the critical path.
— Adding register layers to divide combinatorial logic structures.
— Parallel structures for separating sequentially executed operations into parallel operations.
— Flattening logic structures specific to priority encoded signals.
— Register balancing to redistribute combinatorial logic around pipelined registers.
— Reordering paths to divert operations in a critical path to a noncritical path.

Different hardware design architecture styles can be considered when designing hardware-based cryptosystems [MM03]. The main styles are described below:

Iterative Looping Only one round is designed, hence for an n-round algorithm, n iterations of that round are carried out to encrypt;

Loop Unrolling Involves the unrolling of multiple rounds;

Pipelining Achieved by replicating a round function and placing registers between each round to control the flow of data;

Sub-Pipelining The addition of further registers into a pipelined design when a round function of the pipelined architecture is complex. It decreases the pipeline's delay between stages but increases the number of clock cycles required to encrypt.

A pipelined architecture provides the highest overall throughput. Thus, if a high-speed design is required, a fully pipelined architecture should be chosen. Further speed improvements can be achieved by sub-pipelining the design. However, this incurs additional delays in the output data. If, on the other hand, area is crucial, an iterative architecture will produce the most compact design. For specific speed and area requirements, hybrid architectures can be employed.

2.2.3.1 Throughput and Latency

Perhaps the most important parameters in cryptographic hardware implementations, *throughput* and *latency* are important measurements that describe how fast hardware-based cryptosystems can operate.

Latency is defined as the total duration (based on the system's clock cycles and recorded in time units) required to compute the system's output from the time that the inputs were available. The faster the clock frequency can go, the faster the system operates, as the latency decreases. When it is impossible to increase the clock frequency, due to system constraints or because the design has reached the clock limit due to its combinatorial logic paths, some techniques like pipelining, as described in Section 2.2.3, can be applied to break long combinatorial delays into shorter ones. Although inserting extra registers in the middle of long combinatorial paths will surely increase clock frequency, the overall system latency might not be decreased. For example, consider a system S that can deliver its output one clock cycle after the inputs were provided, given that the maximum reachable clock frequency is 500MHz. Consequently, in this scenario the latency is 2ns. Consider now that the designer was able to insert a register barrier somewhere in the middle worst (longer) path, increasing the frequency to 800MHz. Now each clock takes 1.25ns, but the output is available after two clock cycles. Therefore, the latency is now 2.5ns.

Throughput is defined in terms of latency, and it represents the amount of data per time that a system can output. Consider a cryptosystem with output size of m bits that can process n blocks simultaneously with latency L. The throughput Tp is generally defined as

$$Tp = \frac{m \times n}{L}$$

Typical throughput units are Mbit/s (megabit per second) and Gbit/s (gigabit per second). In the first previously mentioned scenario where system S operates at 500Mhz, consider that S outputs 128 bits every clock cycle, with no parallelism. In this case, the latency is 2ns and the throughput $T = (128 \times 1)/(2 \times 10^{-9}) = 64 \times 10^9 = 64$Gbit/s. Now if the designer is able to insert a logic barrier between the input and the output, pushing the clock to operate at 800MHz, the latency is 2.5ns and the throughput $T = (128 \times 1)/(2.5 \times 10^{-9}) = 51.2 \times 10^9 = 51.2$Gbit/s. As pointed out, pipelining does not solve performance issues if it is not well balanced and designed.

2.2.3.2 Area

A common simple metric in industry is that circuit area is a primary factor that determines the final chip cost. This assumption does not take into account the cost of packaging, which also increases with the circuit area and the number of I/Os. Certain cryptosystem specifications restrict the area available for use. For example, in a smart card, besides the area constraint, the design is also constrained by the number of I/Os. Moreover, limits may be imposed on the power consumed by the chip, which is directly proportional to its area.

In FPGA implementations, area is limited by the available reconfigurable logic blocks (CLB). When synthesizing a design to a FPGA target, the CAD tool typically reports the number of basic configurable logic blocks and the number of equivalent logic gates. Measuring different designs mapped to different FPGA families is particularly challenging as the FPGA manufacturing technology may be different. As a result, the transistor and LUT sizes differ. Commonly in academia and industry, the metric of basic CLBs is used to express the total design size in FPGA, as it provides a more accurate figure of area estimation.

In ASIC implementations, CAD tools usually report number of logic gate instances utilized, as well as the area that these instances sum up to in μm^2. These numbers are function of the standard cell library used during logic synthesis. ASIC CAD tools can also estimate area of wiring the logic together, and the clock power network, which majorly affect the final chip's power consumption.

The Gate Equivalent Metric Comparing two ASIC designs that were mapped to different technology libraries also present similar problems as comparing FPGA implementations. As the transistor size changes, the raw area comparison is unfair. To solve this issue, academia and industry have adopted the *gate equivalent* GE metric, defined as

$$GE = \frac{A}{N}$$

where A is the total design area (technology-dependent), not considering the interconnection area estimation, and N is the area of the smallest NAND gate available in the target technology node (also technology-dependent). The result gives a technology-independent measure of how large the design is, and allows comparisons among chips synthesized in different technology nodes.

2.2.4 Cryptographic Hardware Design

Cryptographic primitives and algorithms are intended to be implemented in both software or hardware platforms. Software implementations are designed and coded in programming languages such as C++, Java and assembly language, targeting general-purpose microprocessors and smart cards. On the other hand, hardware implementations are designed and coded in Hardware Programming Languages (HDLs) such as VHDL and Verilog. These languages map the circuit logic description to logic blocks. Two main implementation approaches can be used: FPGAs or ASICs.

In 1997, the National Institute of Standards and Technology (NIST) initiated a development effort, together with academia, to put in place a new standard for private-key encryption [Koc08] called the *Advanced Encryption Standard* (AES). A new algorithm was selected based on three characteristics: security, efficiency in software and hardware and flexibility. The rapid increase of hardware-based cryptosystems led NIST to think of efficiency in hardware as a main concern.

The final five AES candidates [SKW+98, ABK98, RRSY98, DR99, BCD+99, BCD+99] did not present any cryptanalytic flaws, and their software performance evaluations led to inconclusive results [Koc08], thus hardware efficiency evaluations played a key role in the final AES choice. The candidate Rijndael presented outstanding hardware performance and flexibility, and became the AES winner.

Symmetric block ciphers are used in several operating modes. Their hardware implementation can be divided in two main categories depending on the round-loop-back characteristic:

— Non-feedback modes, such as the Electronic Codebook (ECB) and the Counter Mode (CTR);
— Feedback modes, such as Cipher Block Chaining (CBC), Cipher Feedback Mode (CFB), and Output Feedback Mode (OFB).

Let C_s be a cryptosystem. If the encryption and decryption mode of each subsequent data block can be performed independently of previously processed blocks, we define C_s as a non-feedback block (or mode). The main characteristic of a non-feedback block is that all blocks can be processed in parallel. In contrast, in feedback mode encryption data cannot be processed in parallel because a next message block uses the previous computed output as input. Before introducing the AES winner in more details

(Section 2.3), the next sections are intended to explain the most used techniques to implement private-key hardware cryptosystem for both feedback and non-feedback modes.

2.2.4.1 Iterative Looping

This is the traditional approach for the design of a round-based block cipher. A full round of the block cipher is implemented with combinatorial logic with a logic barrier. The logic barrier can be present at the beginning, to register the inputs, or at the end of the logic path, to register the round output. The output of this block is then fed back into itself, using a multiplexer to select either the initial inputs or the loop signal, as presented in Fig. 2.12.

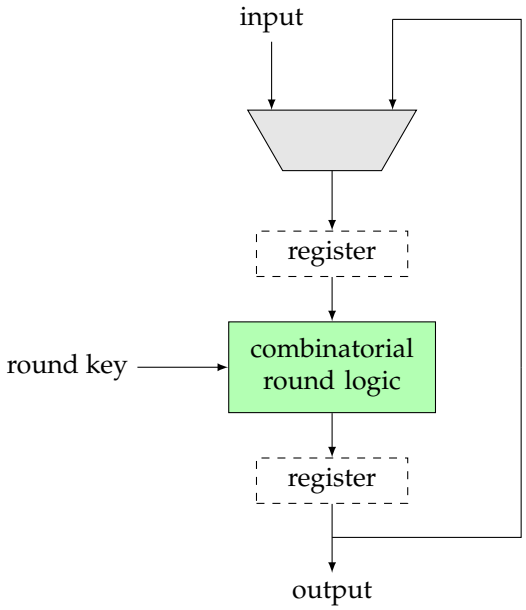


Figure 2.12: General Iterative Looping architecture.

Although it presents low-overhead in terms of area, this design style presents two downsides: it is only possible to encrypt/decrypt one block at a time, and the design gets busy during n clock cycles, where n is the number of rounds of the block cipher. Consequently, the latency L is defined as

$$L = n \times T_{\text{clk}}$$

where T_{clk} is the clock period. The throughput of the iterative looping therefore is given by

$$Tp = \frac{m}{n \times T_{\text{clk}}}$$

where m is the cipher block size in bits.

2.2.4.2 Loop Unrolling

This technique replicates the combinatorial round logic to process the output in less clock cycles. In the *partial loop unrolling*, the design implements K rounds in one combinatorial block, and this result is fed back into the block input, similarly to what happens in the iterative looping (K must be a divisor of n). In the *full loop unrolling*, all the n block cipher rounds are combined in one combinatorial block (thus $K = n$), and both the feedback connection and the multiplexer selecting the input are removed. As a result, the number of clock cycles necessary to compute the output is divided by a factor of K, although

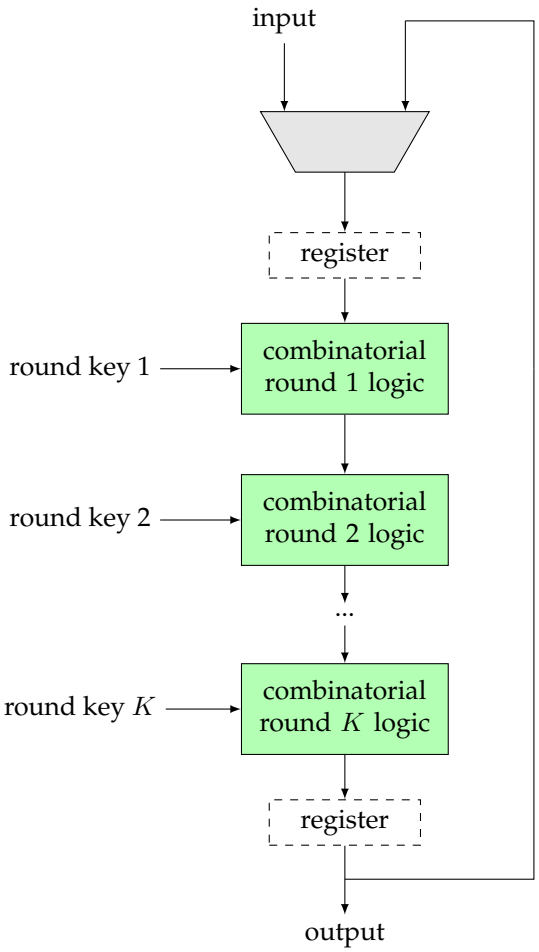


Figure 2.13: General architecture of Loop Unrolling.

the clock period is drastically increased. The general diagram of the loop unrolling scheme is depicted in Fig. 2.13.

The loop unrolling latency is given by

$$L = \frac{n}{K} \times T'_{\text{clk}}$$

and the throughput is

$$Tp = \frac{m \times K}{n \times T'_{\text{clk}}}$$

where $T'_{\text{clk}} \neq T_{\text{clk}}$.

2.2.4.3 Pipelining

One of the most used techniques to improve throughput is called *pipelining*. Pipelining consists in replicating the iterative looping as many times as the maximum available area constraint allows to achieve the highest throughput possible. This technique differs from the loop unrolling because in pipelining, every two consecutive combinatorial round logic blocks are divided by a register, forming a logical barrier between the blocks. By doing so, we achieve clock frequencies comparable to the ones achieved by the iterative looping technique, although substantially increasing the design throughput.

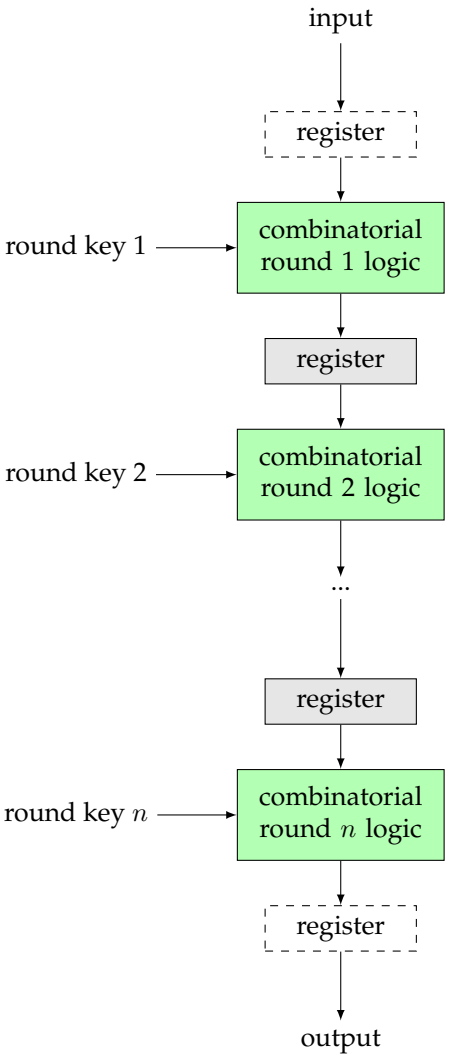


Figure 2.14: General architecture of Pipelining.

Fig. 2.14 shows the general scheme of the pipelining technique. Combinatorial logic blocks are enclosed by register barriers that break down the clock cycle period. The registers at the input and at the output (shown in dotted borders) are optional.

The overall latency of a pipeline scheme remains the same as the iterative looping latency, i.e.,

$$L = n \times T''_{\text{clk}}$$

although T_{clk} might be different than T''_{clk}. In order to calculate the throughput, we have to take into account that the circuit can be fed with new inputs at every new clock cycle. After L time units, the output will be updated at every new clock cycle as well. Therefore this scheme can handle n cipher operations at the same time. As a result, the final throughput is given by

$$Tp = \frac{n \times m}{n \times T''_{\text{clk}}} = \frac{m}{T''_{\text{clk}}}$$

2.2.4.4 Sub-Pipelining

This technique is mainly used when full pipelining costs too much area. In the *sub-pipelining* approach, registers are added between combinatorial round logic blocks, just like in the pipeline scheme, but not every round is duplicated. Instead, K rounds are replicated in series, and the output of the last round loops back to the design input. Again K must be a divisor of n.

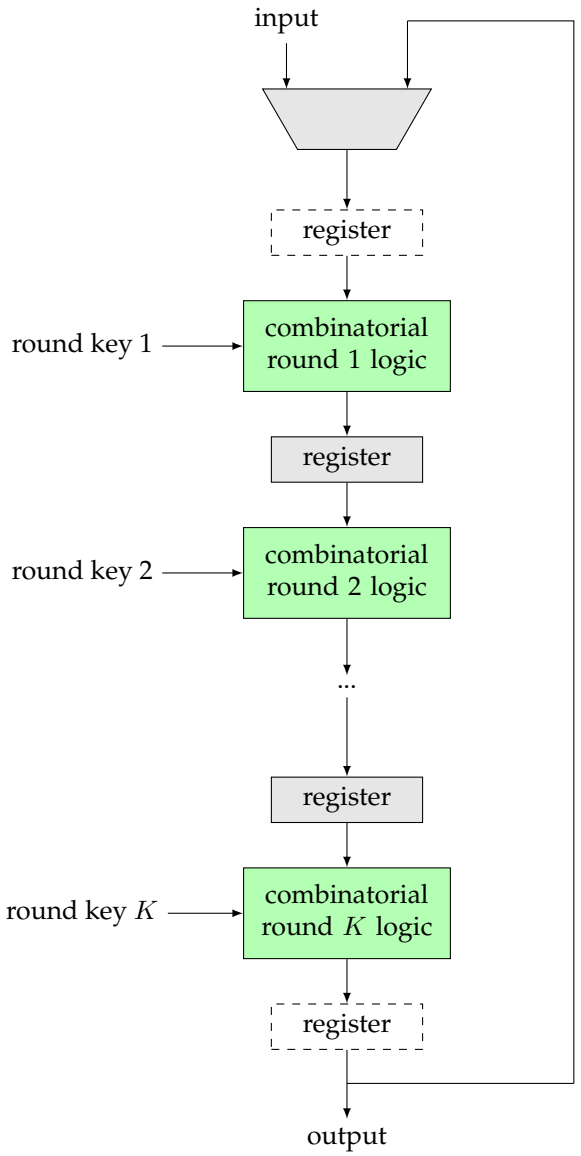


Figure 2.15: General architecture of Sub-Pipelining.

Fig. 2.15 represents the sub-pipelining architecture, containing K combinatorial logic blocks with a feedback path from the output to the input. A multiplexer selects whether the input or the feedback loop is sent to the first combinatorial block. The loop has to be repeated $n/K - 1$ times for the cipher to complete. The latency is defined as

$$L = n \times T'''_{\text{clk}}$$

as in the pipelining design style. To compute the sub-pipelining throughput, we have to consider that K inputs are fed into the core, followed by a delay of $L - nT_{\text{clk}}$ time units during which the design is busy computing the output. As a result, the throughput is

$$Tp = \frac{K \times m}{n \times T'''_{\text{clk}}}$$

2.2.4.5 Pseudo-Random Sequences in Hardware

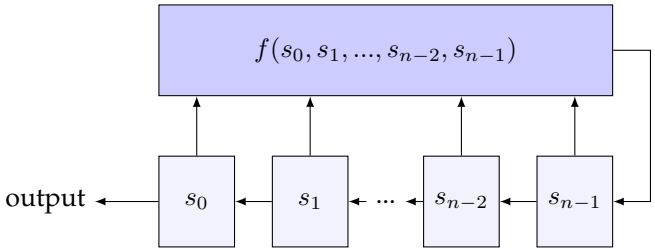


Figure 2.16: General architecture of a Feedback Shift Register (FSR).

A hardware cryptosystem, based on logical circuits, works with the alphabet $(0, 1)$ because the transistor can only drive V_{DD} or V_{SS} to its output. Based on that, a purely digital circuit cannot be truly random, as its underlying finite state machine and combinatorial logic are deterministic.

Cryptographic primitives often make use of pseudo-random binary sequences. A fast hardware implementation of such scheme is the FSR. A Feedback Shift Register of length n contains n memory cells that represent the state of the FSR. The function f, called the *feedback function*, is a Boolean function, therefore represented by logical functions, mapping $(0, 1)^n$ in $(0, 1)$. After the first time unit, the FSR will output s_0 and transit to state $(s_1, s_2, ..., s_n)$, where $s_n = f(s_0, s_1, ..., s_{n-1})$. At each execution, the FSR goes on and output the next state based on the previous state, generating an infinite sequence. The general scheme of a FSR is shown in Fig. 2.16.

In the case where f is a linear function, it can be written as

$$f(s_0, s_1, ..., s_{n-1}) = c_0 s_0 \oplus c_1 s_1 \oplus ... \oplus c_{n-1} s_{n-1}$$

where c_i are binary. Such scheme is named Linear Feedback Shift Register (LFSR), and the coefficients c_i are called the *feedback coefficients*. If $c_i = 0$, the corresponding switch in Fig. 2.17 is opened, while if $c_i = 1$, the corresponding switch is closed.

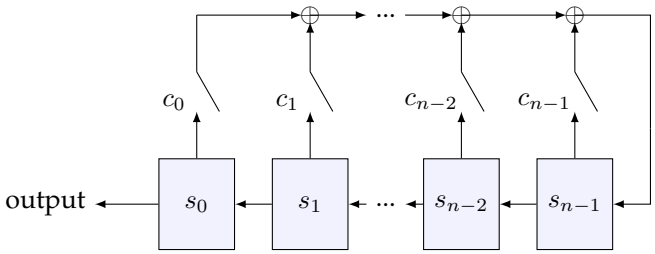


Figure 2.17: General architecture of a Linear Feedback Shift Register (LFSR).

There are at most $2^n - 1$ different states in a LFSR of length n. Therefore the period of $s_i \geq 0$ will never exceed $2^n - 1$. The length of the sequence before repetition occurs depends upon two factors, the feedback taps (XORs) and the initial state. An LFSR of any given size n (number of registers) is capable of producing every possible state during the period $N = 2^n - 1$ shifts, but will do so only if proper feedback taps have been chosen. A *pseudo-noise sequence* is defined as the output of a LFSR of length n that has period exactly equal to $2^n - 1$.

2.3 Private-Key Cryptosystems

Symmetric-key cryptography, also known as private-key or secret-key cryptography, involves using the same key for both encryption and decryption processes (hence the term *symmetric*). For this type of cipher to work, it is necessary to both parties to agree on the key prior to encryption or decryption. It is also extremely important that the key remains secret, as the algorithm is public and therefore known. In other words, the whole secrecy of such schemes relies on the key.

In order to work, private-key cryptography rely on three algorithms [Vau05]:

— a key generator that provides expanded keys in a pseudo-random fashion;
— an encryption algorithm that, given the plaintext input P, outputs the ciphertext C, using at each round the key provided by the key generator;
— a decryption algorithm that transforms the ciphertext C back to the original plaintext P, using the same secret key.

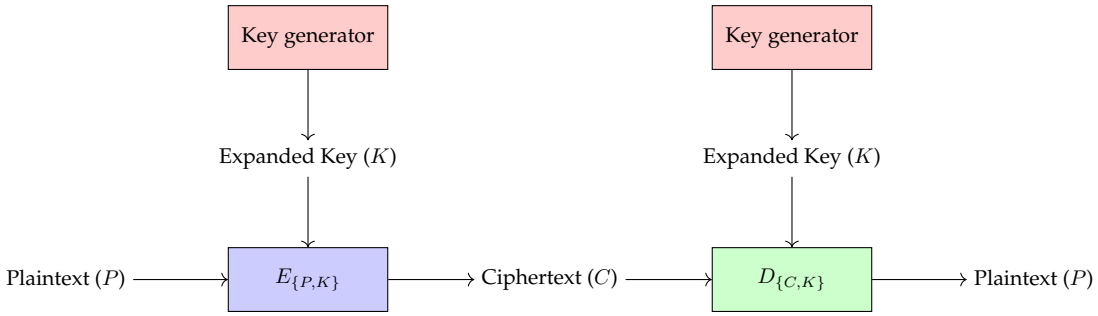


Figure 2.18: Secret-key cryptography (overview).

To encrypt, most secret-key schemes use two main techniques known as substitution and permutation. Substitution is simply a mapping of one value to another whereas permutation is a reordering of the bit positions for each of the inputs. These techniques are used a number of times in iterations called rounds. Generally, the more rounds there are, the more secure is the algorithm. Substitution provides non-linearity to the encryption scheme so that decryption will be computationally infeasible without the secret key. This is achieved, for instance, with the use of S-Boxes which are basically non-linear substitution tables where either the output is smaller than the input or vice versa.

The final goal of symmetric-key encryption is to enable confidential communication between two parties considering that the communication channel is insecure and therefore prone to eavesdropping. The next sections describe two of the most adopted and known private-key ciphers: the DES and the AES.

2.3.1 The Data Encryption Standard

The *Data Encryption Standard* (DES) was the predominant symmetric-key algorithm back in the day. The extinct National Bureau of Standards (NBS) felt the need to standardize a secure method of electronic data encryption and ended up approving an algorithm proposed by IBM's employee Horst Feistel in the early 1970s.

Although considered outdated by industry and academia, due to its small key space and successful proposed attacks [HC99, BS91], the reality is that the DES is still being used in many legacy applications. In fact, the standard is officially outdated as it was eventually not renewed by NIST in 2004.

2.3.2 The Advanced Encryption Standard

On January 2, 1997 the National Institute of Standards and Technology (NIST) held a contest for a new encryption standard. The previous standard, DES, was no longer adequate in terms of security,

although it had been the standard since November 23, 1976. Apart from the fact that the DES construction was fast in hardware implementations, computational power had increased a lot since then, and several successful attacks proved that the algorithm was no longer considered safe. In 1998 DES was cracked in less than three days by a specially made computer called the DES cracker [Fou98], created by the Electronic Frontier Foundation for less than $250,000 and the winner of the RSA DES Challenge II-2.

Current alternatives to a new encryption standard were Triple DES (3DES) and International Data Encryption Algorithm (IDEA). The problem was that IDEA and 3DES were too slow in hardware and IDEA was not free to implement due to patent restrictions. NIST opted for a free and easy-to-implement algorithm that could provide outstanding security. Additionally, among NIST requisites was that the algorithm had to be efficient and flexible in both hardware and software.

After holding the contest for three years, NIST chose an algorithm created by two Belgian computer scientists, Vincent Rijmen and Joan Daemen. They named their algorithm Rijndael after themselves. Supposedly Rijndael can only be pronounced correctly by people who can speak Dutch and the closest English approximation is "Rhine Dahl".

On November 26, 2001 the Federal Information Processing Standards (FIPS) Publication 197 announced a standardized form of Rijndael as the new encryption standard. This standard was called the *Advanced Encryption Standard* (AES) and it is currently the standard for private-key encryption.

2.3.2.1 AES Rounds

The AES is based on a substitution-permutation network and works the input block bytes as elements defined over the Galois Field (introduced in Section 1.4). The AES is a symmetric block-cipher that can process 128 bits of data, using keys of 128, 192 or 256 bits. The specification of Rijndael defines longer keys, although they were not standardized.

AES defines iterative operations on a 4×4 matrix of bytes called the *state*. Each element of the *state* array is denoted by $s_{r,c}$, where r and c are respectively the row and column positions, with $0 \leq r, c < 4$. At the start of the cipher, the input is copied to the state, which is then transformed by an XOR operation with the round key from the *Key Expansion* block. After that, the state undergoes the round function, that will be repeated N_r times, depending on the size of the key. N_r is defined to be 10, 12 or 14 for the key size of 128, 192 and 256 bits, respectively.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Figure 2.19: The AES *state*.

The AES *state*, pictured in Fig. 2.19, undergoes four different transformations during each round processing: *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*, for the encryption process, and *InvSubBytes*, *InvShiftRows*, *InvMixColumns* and *AddRoundKey*, for the decryption process. Note that, because the *AddRoundKey* step consists of a XOR between the *state* and the *RoundKey*, the inverse operation is idempotent. These AES transformations are detailed in the following sections.

***SubBytes* and *InvSubBytes*.** Transforms individual bytes of the *state* following two steps:

— Multiplicative inversion in $GF(2^8)$ with the reduction polynomial $m(x)$, where

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

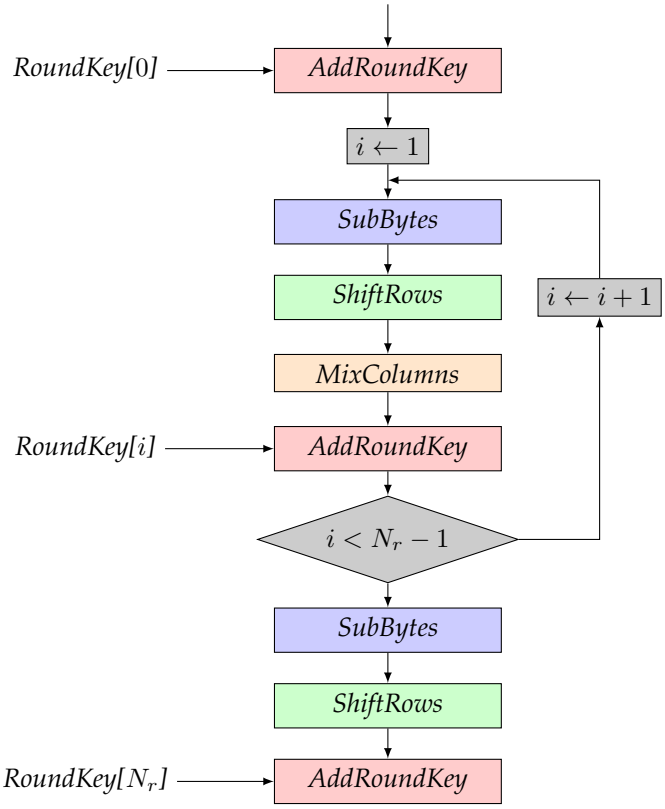


Figure 2.20: The AES encryption flowchart.

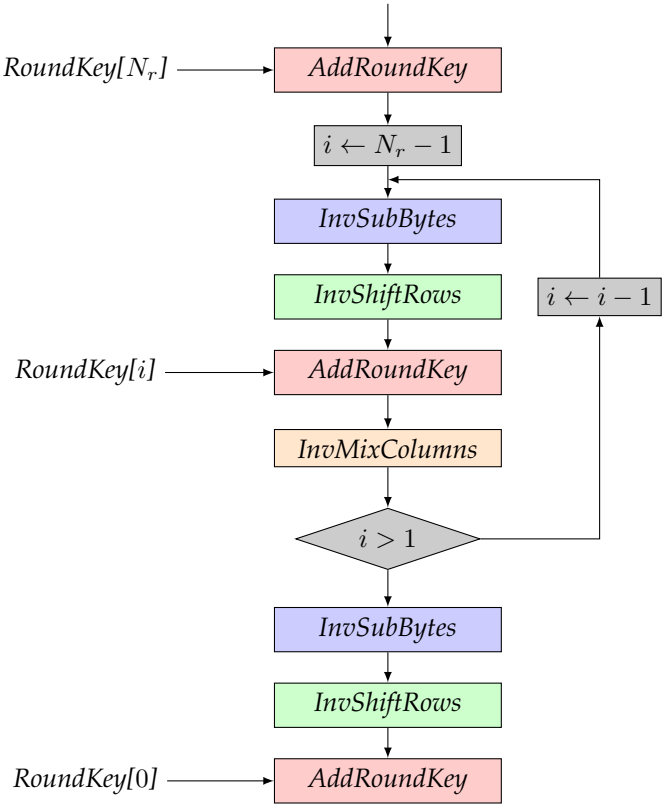


Figure 2.21: The AES decryption flowchart.

— Affine transformation over $GF(2)$, following

$$b'_i = b_i + b_{i+4} + b_{i+5} + b_{i+6} + b_{i+7} + c$$

where byte c is a constant defined by 01100011 (or `0x63`) and additions are considered modulo 8.

Typically in an AES hardware implementation, *SubBytes* is implemented as follows: each $s_{i,j}$ of the AES *state* is replaced with a *SubByte* $SB(s_{i,j})$ using an 8-bit substitution box, called the *S-Box*. This operation provides the non-linearity of the AES cipher, and it is based on the good non-linearity properties of the multiplicative inverse over $GF(2^8)$. The *S-Box* was carefully chosen to avoid any fixed points, i.e., $SB(s_{i,j}) \neq s_{i,j}$ and also $SB(s_{i,j}) \oplus s_{i,j} \neq$ `0xFF`.

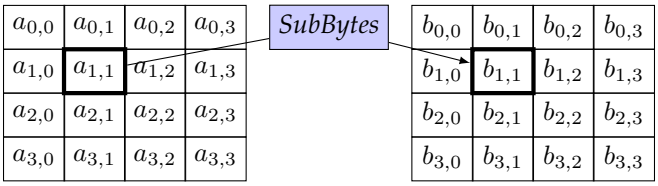


Figure 2.22: Application of *SubBytes* transformation to the *state*.

***ShiftRows* and *InvShiftRows*.** These transformations simply cyclically shift three bottom rows of the *state* by one, two and three byte positions, respectively, as shown in Fig. 2.23. The upper row is unchanged.

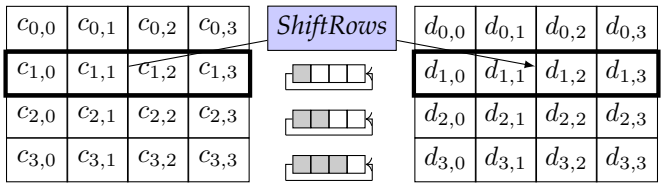


Figure 2.23: Application of *ShiftRows* transformation to the *state*.

***MixColumns* and *InvMixColumns*.** This transformation step is defined over 4-byte words that represent a column of the *state*. They can be seen as polynomials of degree ≤ 3 with coefficients in $GF(2^8)$, defined in the ring of polynomials $R = K[X]/(X^4 + 1)$ modulo $M(X) = X^4 + 1$.

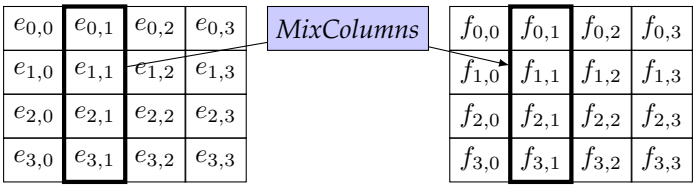


Figure 2.24: Application of *MixColumns* transformation to the *state*.

***AddRoundKey*.** The AES key scheduling is the process of generating $N_r + 1$ round keys based on the original secret key provided to the system.

2.3.2.2 AES in Hardware (FPGA and ASIC)

The Rijndael algorithm was selected the Advanced Encryption Standard, among other reasons, thanks to its performance. In [DPR00] it is shown that Rijndael achieves the lowest key-setup latency

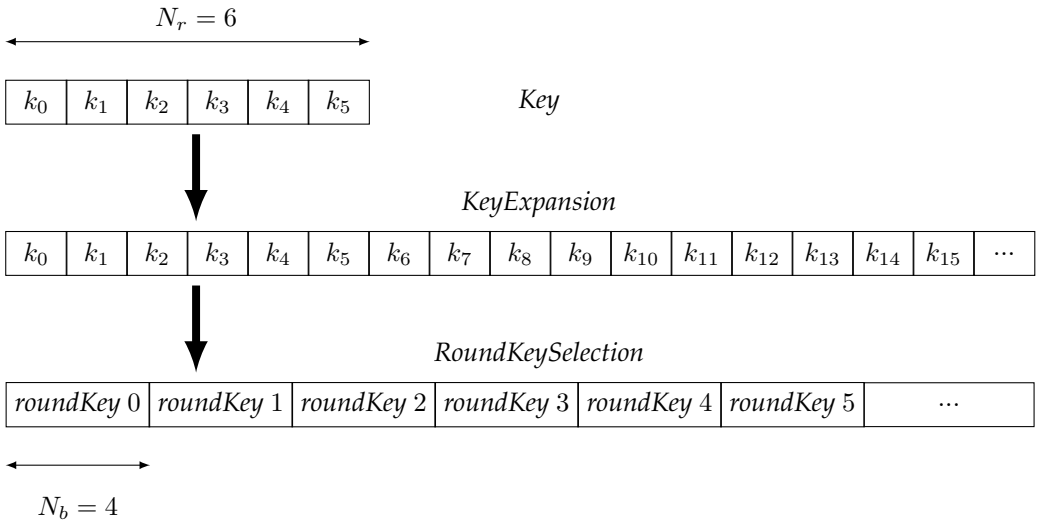


Figure 2.25: Decomposition of key scheduling into *KeyExpansion* and *RoundKeySelection* for $N_k = 6$ (192-bit key) and $N_b = 4$ (128-bit data block). In the figure, k_i is a word of 32 bits.

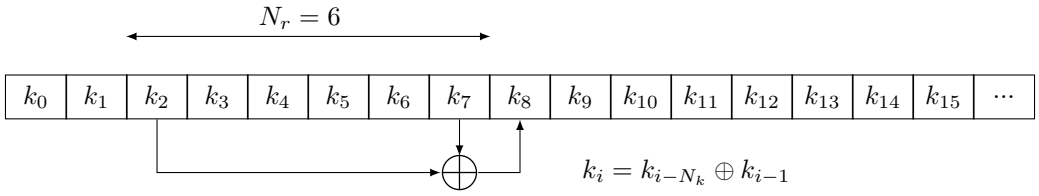


Figure 2.26: *KeyExpansion* formula for $i \bmod N_k \neq 0$.

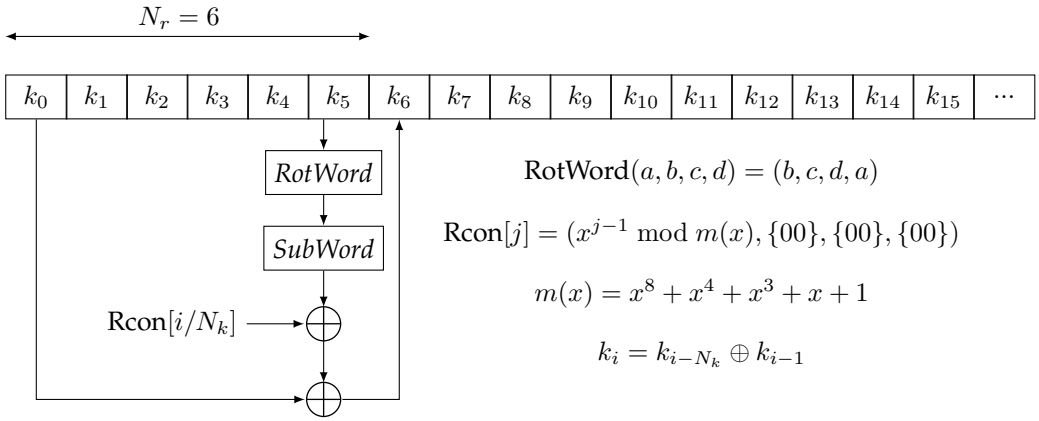


Figure 2.27: *KeyExpansion* formula for $i \bmod N_k = 0$.

time and also the highest encryption rate due to the ideal match of its algorithm characteristics with the hardware characteristics of FPGAs. Moreover, the winner candidate also achieves the best hardware utilization.

Although designs aiming at reducing the area usually focus on smaller datapaths, such as 32 or even 8 bits wide, the publications analyzed here always present a datapath of the size of AES internal state: 128 bits. Reported low area architectures [CG03] [RSQL04] have been based around a 32-bit datapath. As the AES operations *MixColumns* and *KeyExpansion* are fundamentally 32-bit, it was previously believed that this was optimal. An ASIC design by [FDW04] used an 8-bit datapath connected to a 32-bit *MixColumns* operator. However, even *MixColumns* may be rewritten in an 8-bit form accepting a higher control overhead and reduced throughput.

Algorithm 1 AES algorithm description.

Require: plaintext $P[4 \times 4]$
Require: key $K[4 \times 4]$
Ensure: ciphered $C[4 \times 4]$

1: state$[4 \times 4] \leftarrow P$

2: *AddRoundKey*(*state*, K[0,3])

3: **for** $round \leftarrow 1, N_r - 1$ **do**
4: *SubBytes*(*state*)
5: *ShiftRows*(*state*)
6: *MixColumns*(*state*)
7: *AddRoundKey*(*state*, $K[round \times 4, (round + 1) \times 3]$)
8: **end for**

9: *SubBytes*(*state*)
10: *ShiftRows*(*state*)
11: *MixColumns*(*state*)
12: *AddRoundKey*(*state*, $K[N_r \times 4, (N_r + 1) \times 3]$)

13: $C \leftarrow$ *state*
14: **return** C

[DPR00] compares its proposed AES design on FPGA with [EYCP00] and [GC01]. All of these articles achieve throughputs in the same order of magnitude for AES-128, despite the fact that [EYCP00] and [GC01] do not implement the key schedule, while [DPR00] does. These results are then compared to with the NSA ASIC-based implementations described in [WBRF00]. As we can see in the literature, the best FPGA AES designs achieve approximately half of the throughput that NSA describes as optimum for an ASIC implementation of AES.

[KV01] proposed an AES design where one full round is computed in one clock cycle. The design separates into two modules: the main datapath and the key schedule, that generate subkeys on-the-fly. Despite the high throughput achieved in this design, no pipeline or unrolling were implemented. In [EYCP00], an even higher throughput is achieved by a partially unrolled FPGA design. Similar throughput is achieved by an ASIC design presented in [IKM00]. A pipelined encryptor core is presented in [MM01] with a very high throughput for an FPGA implementation. The highest throughput is presented by a fully-pipelined ASIC design described by NSA in [WBRF00] (until 2002).

Table 2.3: Comparison of different AES implementations.

AESAlgorithm	[EYCP00] (FPGA)	[GC01] (FPGA)	[DPR00] (FPGA)	[WBRF00] (ASIC) ($0.5\mu m$)	[KV01] (ASIC) ($0.18\mu m$)	[EYCP00] (FPGA)	[IKM00] (ASIC) ($0.35\mu m$)	[MM01] (FPGA)	[WBRF00] (ASIC) (μm)
Throughput (Mbit/s)	300.10	331.50	353	605.77	1820	1937.9	1950	3239	5163

In CHES 2005, an interesting work [GB05] presented a super fast implementation of AES and compared with previously known best designs. The first design decision was to remove all the loops to form a loop-unrolled design where several register barriers create a "production line" in which a new data block can be input at each clock cycle. Each block progresses through the pipeline stages until it has been processed and output. This allows the design to increase its throughput, but increases latency (the time from the input of the first block of data until it is processed and output).

The second design approach was to express the *SubBytes* operation in computational form rather than look-up operations, due to the fact that look-up operations have an inherent delay to pass through the FPGA memory blocks. FIPS-197 [AES01] provides the specification of the mathematical derivation of SubBytes in terms of Galois Field (2^8) arithmetic.

The third and last approach to find the best throughput design was to carefully analyze the datapath of a whole round and perform the pipeline cuts with the best logic balancing. It was found that each round should be cut into 7 different places, resulting in a round with a 7 clocks latency, and the full latency of the AES design being 70 clock cycles. The optimized logic level (the number of combined LUTs between two adjacent pipeline stages) was found to be 3. Moreover, the pipeline stages of the final round were slightly modified for optimal logic balancing, resulting in an optimal throughput. Results are compared in the following table:

Table 2.4: Comparison of different pipelined AES implementations.

Design	FPGA	Freq. (MHz)	T'put (Mbit/s)	Latency (ns)	Area (slices)	Mbit/s / slice	Datapath
[JTS03]	Virtex-E XCV1000E-8	129.2	16,500	-	11,719	1.408	Enc
[SMMS03]	Virtex-E XCV2000E-8	158	20,300	-	5,810 + 100B RAM	1.091	Enc
[SRQL03]	Virtex-E XCV3200E-8	145	18,560	-	15,112	1.228	Enc
[HV04]	Virtex-II Pro XC2VP20	169.1	21,640	420	9,446 wo/ KE	2,290	Enc
[ZNC04]	Virtex-II XC2V4000	184.1	23,570	163	16,938	1.391	Enc
[ZP04]	Virtex-E XCV1000E-8	168.4	21,556	416	11,022 wo/ KE	1.956	Enc/Dec
[GB05]	Virtex-E XCV2000E-8	184.8	23,654	379	16,693	1.417	Enc/Dec

2.4 Cryptographic Hash Functions

2.4.1 Introduction

There are many applications for which one needs a function that is easy to compute, but hard to revert. Cryptographic hash functions are such constructions. They compute a fingerprint of this message, called the *message digest*. Hash functions are an essential part of digital signature schemes and message authentication codes. Hash functions are also widely used for other cryptographic applications, e.g., for the storing of password hashes or key derivation.

The signing of long messages is particularly challenging. The question arises when we want to hash strings longer than the hash block size. We could use a naïve approach by applying a scheme similar to the ECB mode of operation for block ciphers: divide the message x into blocks x_i of size shorter than the allowed input size of the signature algorithm, and sign each block separately. However, this approach yields three serious problems [PP09]:

High Computational Load The longer the message, the more time and energy it will take to compute its message digest, which may be unfeasible in certain environments with limited resources;

Message Overhead The transmission overhead is increased, as we have to send now the message and its signature;

Security Limitations New attacks can happen in this scenario. For example, an attacker can remove part of the message and its corresponding signatures, or she could reorder them, or even reassemble new messages and signatures out of fragments of previous messages and signatures.

What we want instead is a hash function capable of computing a fingerprint of the message x having always the same size, no matter the size of x. A basic protocol is depicted in Fig. 2.28, assuming that we have such a construction:

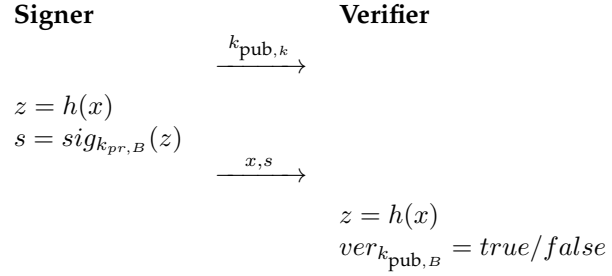


Figure 2.28: Basic protocol for digital signatures with a hash function.

The sender computes the hash of the message x with its private key $k_{k_{pr,B}}$. On the receiving side, the verifier computes the hash value z of the received message x and also checks that the correctness of signature s with the public key $k_{\text{pub},B}$.

Now that we have a better understanding of what a hash function should be and behave, let us define it more formally:

Definition 2.6 (Hash Function) *A hash function takes as input an arbitrary long message x and returns a short bit string z. The primary properties that a hash function must have are:*

- *Computation of hash h of x should be fast and easy, e.g., linear time;*
- *Inversion of h result should be difficult, e.g., exponential time. More precisely, given a hash value z, it should be difficult to find any message x such that $h(x) = z'$;*
- *The hash function should be collision resistant. This means that it should be hard to find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.*

Besides being able of computing a message digest from a message x of arbitrary length, it is also desirable that the function h be computationally efficient. Another desired property is that the digest is of fixed length independently of the input length. Practical hash functions have output lengths between 128 and 512 bits.

2.4.2 Security Requirements of Hash Functions

Although hash functions do not have keys, some properties must be respected for a hash to be considered secure. Notably, hash functions need to possess three properties to be secure:

— preimage resistance (or one-wayness);
— second preimage resistance (or weak collision resistance);
— collision resistance (or strong collision resistance).

These properties imply that a adversary cannot replace or modify the input data without changing its digest. Thus, if two strings have the same digest, one can be very confident that they are identical. The sections that follow detail these properties.

2.4.2.1 Preimage Resistance

Given a digest z it must be computationally infeasible to find an input message m such that $z = h(m)$. This property is also called *one-wayness*. To highlight why preimage resistance is crucial, consider that the sender encrypts the message but not the signature. In this case, the sender transmits the pair

$$(e_k(m), \text{sig}_{k_{\text{pr},B}}(z)).$$

Here consider that $e_k()$ is a symmetric cipher (AES, for example). Assume also that the verifier uses an RSA digital signature that is computed as (where $k_{\text{pr},B} = d$)

$$s = \text{sig}_{k_{\text{pr},B}}(z) = z^d \bmod x$$

An attacker can use the verifier's public key to compute

$$s^e \equiv z \bmod x$$

If the hash function in place does not replace the one-wayness property, the attacker can compute the message m from $h^{-1}(z) = m$. Thus, the symmetric encryption of m is circumvented by the signature, which leaks the plaintext.

2.4.2.2 Second Preimage Resistance

Using a hash function to digitally sign messages implies that two messages do not hash to the very same value. Therefore it should be computationally infeasible to create two different messages $x_1 \neq x_2$ having equal digests $z_1 = h(m_1) = h(m_2) = z_2$. This is called second preimage resistance or weak collision resistance.

Assume that the sender hashes and signs a message m_1. If an attacker is capable of finding a second message m_2 such that $h(m_1) = h(m_2)$, the substitution attack exemplified in Fig. 2.29 can take place.

Since the verification outputs $true$, the verifier is fooled into thinking that the message is authentic. This happens because the actual hashing does not process the actual message, but its hashed version instead. If the attacker is able to find a message m_2 with the same digest of m_1, signing and verifying give the same result.

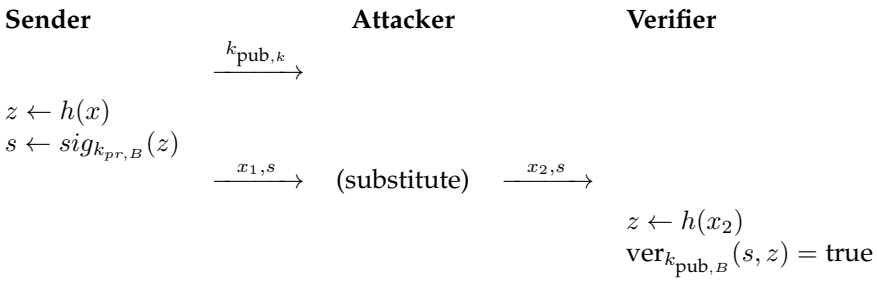


Figure 2.29: Substitution attack on a hash scheme without second preimage resistance.

In principle, it is impossible to avoid that an attacker finds m_2 such that $h(m_1) = h(m_2)$. Given that the number of possible hash input messages is infinite and the digest has a finite size n, the hash result spans from 0 to $2^n - 1$. Therefore, multiple input messages will hash to the same digest. This is known as the *pigeonhole principle* [Ajt88].

To avoid that, a hash function must be such that no analytical attack can happen. A strong hash construction must be designed such that, given x_1 and $h(x_1)$ it is impossible to construct x_2 such that $h(x_1) = h(x_2)$. Similarly to an exhaustive key search for symmetric ciphers, the attacker still can choose x_2 at random, compute $h(x_2)$ and check whether it matches with $h(x_1)$. With today's computational power, it is enough to have an output length of 80 bits to make this attack unpractical.

2.4.2.3 Collision Resistance

A hash function is *collision resistant* or *strong collision resistant* if it is computationally infeasible to find two different inputs $x_1 \neq x_2$ with $h(x_1) = h(x_2)$. This property differs from second preimage resistance in the sense that here the attacker has two degrees of freedom: he can modify both x_1 and x_2 at will to match $h(x_1)$ and $h(x_2)$. This attack is depicted in Fig. 2.30.

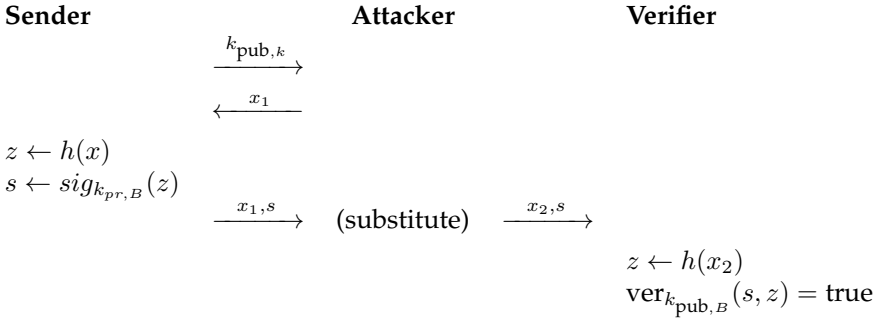


Figure 2.30: Attack on a hash scheme without second collision resistance.

This attack assumes that the adversary can fool the sender into hashing the message x_1, which is not always doable. As we know, collisions exist, therefore it is important to know how strong the hash is against that. If the hash output is 80 bits long, we have to check about 2^{40} messages to find second preimages, due to the *birthday attack* [GCC88]. This attack is based on the *birthday paradox*, which is a powerful tool often used by cryptanalysts.

The Birthday Paradox. How many persons are needed to have 50% chance of having at least two people born on the same day? To answer that question, we start by computing the probability of no collision. Let $P_{nc}(2)$ be the probability of having no collision between two persons

$$P_{nc}(2) = (1 - \frac{1}{365})$$

If we have now a third person, her birthday can collide with both persons, therefore

$$P_{nc}(3) = (1 - \frac{1}{365}) \cdot (1 - \frac{2}{365})$$

Following the progression, the probability of t people having no birthday collision is given by

$$P_{nc}(t) = (1 - \frac{1}{365}).(1 - \frac{2}{365}) \cdots (1 - \frac{t-1}{365}) = \prod_{i=1}^{t-1}(1 - \frac{i}{365})$$

Obviously, for $t = 366$ we have $P_{nc}(t) = 1$. We tend to believe that we need roughly half of that to achieve a 50% probability, but in reality we need only 23 persons! Let $P_c(t)$ be the probability of having at least one collision for t people, given by:

$$\begin{aligned} P_c(t) &= 1 - P_{nc}(t) \\ P_c(23) &= 1 - (1 - \frac{1}{365}) \cdots (1 - \frac{23-1}{365}) \\ &= 0.507 \approx 50\% \end{aligned}$$

2.4.2.4 Overview of Hash Algorithms

For practical applications, it is fundamentally important that the hash function be fast since the hash accepts an input of any size. Because of that, hash constructions usually apply *ad hoc* mixing operations rather than complicated mathematical constructions such as factoring or discrete logarithms. There are two general types of hash functions:

Dedicated hash functions These are algorithms dedicated to hash messages;

Block cipher-based hash functions Modified block cipher constructions to serve as hash function.

Hash constructions usually apply the *Merkle-Damgård construction*, where the input message is broken down into pieces of the same size and fed into the hash block, that uses a compression function as its main engine. Iteratively, the hash function gives the output based on all the given input messages. The final hash value is defined as the output of the last iteration of the compression function.

The most widespread hash function is the SHA (Secure Hash Algorithm) family, standardized by NIST [SHA93]. SHA versions differ on the level of security and the message digest size.

2.4.3 The Secure Hash Algorithm 1

The SHA-1 standard came to replace the no longer safe MD4 hash family. Although, it has been the most widely used message digest function since 1995, in 2005 a team of Chinese researchers were the first to propose a collision attack in the full SHA-1 in 2^{69} hash operations, much less than time-memory tradeoff attack (2^{80}) [WYY05].

Because of that, NIST adopted a new standard called the SHA-2, and since 2010 many organizations have recommended the SHA-1 replacement by SHA-2. Companies like Microsoft, Google, and Mozilla have all announced that their web browsers will stop accepting SHA-1 SSL certificates by 2017.

2.4.4 The Secure Hash Algorithm 2

In 2001, even before the publication of collision attack that made SHA-1 be considered unsafe, NIST published a new cryptographic hash function standard called SHA-2. Although also based on the Merkle-Damgård construction, SHA-2 includes significant changes from its predecessor, featuring larger digest sizes: 224, 256, 384 and 512 bits.

The performance of SHA-2 family depends on the length of the hashed message. The way the padding operation is implemented also impacts the overall hardware performance . The algorithm is basically divided into 3 steps: (i) message padding; (ii) expansion; and (iii) compression, according to Fig. 2.31.

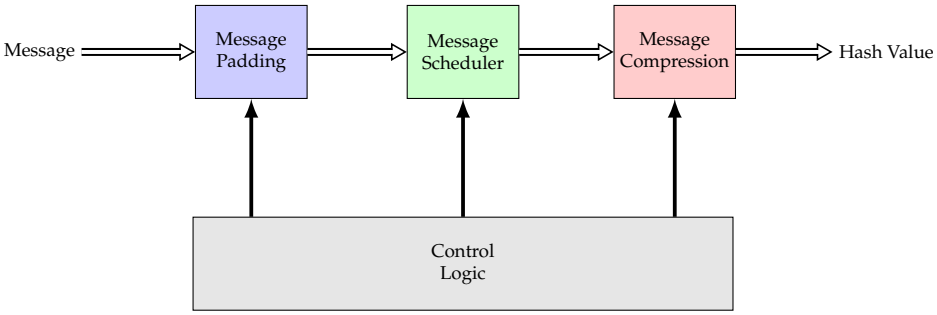


Figure 2.31: General block diagram of the hardware SHA-2 implementation.

i The binary message is appended to a single 1 and then padded with zeros until its length $\equiv 448 \bmod 512$. The resulting string is divided into M blocks and each of these are input to the algorithm. In hardware, this translates to processing each input message as 512-bit blocks until another input signal flags the end of the message. The last block M^i is then padded accordingly and then processed, which means that the last block usually takes more time to process.

ii The SHA-256 functions operate on 32-bit words, therefore each M^i block is viewed as 16 32-bit words denoted M_t^i, where $0 \leq t \leq 15$. The message expander, also called the message scheduler, takes each M^i and expands it into 64 32-bit blocks W_t according to the following equations:

$$\sigma_0(x) = \text{ROT}_7(x) \oplus \text{ROT}_{18}(x) \oplus \text{SHF}_3(x) \; \sigma_1(x) = \text{ROT}_{17}(x) \oplus \text{ROT}_{19}(x) \oplus \text{SHF}_{10}(x)$$

$$W_t = \begin{cases} M_t^i & 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & 16 \leq t \leq 63 \end{cases}$$

where the function $\text{ROT}_n(x)$ denotes a circular rotation of x by n positions to the right, while the function $\text{SHF}_n(x)$ denotes the right shifting of x by n positions. All additions in the SHA-256 algorithm are modulo 2^{32}.

iii The final step, considered as the SHA's core, utilizes 8 32-bit variables labeled from A to H and initialized to constant values. The compression function performs 64 iterations, given by:

$$T_1 = H + \sum_1(E) + \text{Ch}(E,F,G) + K_t + W_t$$
$$T_2 = \sum_0(A) + \text{Maj}(A,B,C)$$
$$H = G; G = F; F = E; E = D + T1; D = C; C = B; B = A; A = T_1 + T_2$$

where

$$\text{Ch}(x,y,z) = (x \wedge y) \oplus (\bar{x} \wedge z)$$
$$\text{Maj}(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$
$$\sum_0(x) = \text{ROT}_2(x) \oplus \text{ROT}_{13}(x) \oplus \text{ROT}_{22}(x)$$
$$\sum_1(x) = \text{ROT}_6(x) \oplus \text{ROT}_{11}(x) \oplus \text{ROT}_{25}(x)$$

and the inputs denoted K_t are 64 32-bit constants, defined in [SHA95]. After 64 iterations of the compression function, as intermediate hash value $H^{(i)}$ is calculated:

$$H_0^{(i)} = A + H_0^{(i-1)}$$
$$H_1^{(i)} = B + H_1^{(i-1)}$$
$$\ldots$$
$$H_7^{(i)} = H + H_7^{(i-1)}$$

The SHA-256 compression algorithm then repeats and begins processing another 512-bit block. After all N data blocks have been processed, the final 256-bit output, $H^{(N)}$, is formed by concatenating the final hash values:

$$H^{(N)} = H_0^{(N)} | H_1^{(N)} | H_2^{(N)} | \ldots | H_7^{(N)}$$

2.4.5 Implementation Tradeoffs and Design Methodologies

Basically, two classes of implementation tradeoffs can be applied: architectural and algorithmic optimizations. The first exploits algorithm-independent design techniques such as pipelining, sub-pipelining, pseudo-pipelining, loop unrolling, logic balancing and many others. The second exploits algorithm strengths translated into hardware improvements.

Implementation efficiency can be defined in two manners:

— in terms of performance, let the efficiency of a block-cipher to be the ratio Throughput (Mbit/s) / Area (slices);
— or in terms of resources, in which case efficiency is measured by computing Number of LUTs / Number of registers, which should be close to one.

2.4.6 Known SHA-2 Hardware Optimization Techniques

The critical path of the SHA core is the calculation of variable A, which involves addition (modulo 2^{32} for SHA-224/256, module 2^{64} for SHA-384/512) of 7 operands. The following techniques have been proposed to speed up the SHA function in hardware:

— Using Carry-Save Addition (CSA) [DMO04a] [DMO04b] [GLG+02] [LGG04] [MD05], that works separating the sum and carry paths. This shrinks the critical path of the carry propagation. The variable A can then be calculated with 3 CSAs since it accepts 3 input operands.
— Unrolling [CDKM04] [LGG04]. Multiple compression function rounds are implemented in combinational logic to reduce the number of clock cycles needed to perform this operation. This technique is more suitable when the core has a defined clock frequency and the compression function round operates at much faster frequencies.
— Quasi-pipelining [DMO04a] [DMO04b] [MD05]. This technique breaks the longer paths by adding registers and therefore pipelining the core. Implementing pipeline in the SHA core is not trivial due to inherent feedback loops. External control logic to enable/disable registers is required, thus increasing the core's area. This is the best technique to increase clock frequency and data throughput.
— Delay balancing [DMO04b]. To complete the addition operation with CSAs, Carry-Lookahead Addition (CLA) adders are used. By registering the sum and carry before using this cells, implementers actually manage to move the CLAs out of the critical path, thus increasing the operational frequency. Area overhead due to extra control circuitry is the drawback of this technique, which is quite effective.
— Logic balancing [TYLL02]. This technique consists in moving time-consuming operations, changing the order of operations without changing the actual result. In [TYLL02], the first addition in the critical path ($K_t + W_t$) is moved to the message expansion stage, since both operands are available before the others. This technique is rather similar to quasi-pipelining, while the later achieves even shorter paths.
— Using RAM to store constants [MM02]. When the design is targeted to FPGA, usually the usage of on-chip available memory is very suitable to reduce logic and speed up the design. By storing constant and intermediate values in memory, area is decreased and routing becomes simpler, increasing the operation frequency. Moreover, FPGA also allows the designer to code the logic in terms of look-up operations using Look-Up Tables (LUTs) which drastically reduces logic and therefore increases frequency.
— Using parallel counter [GLG+02]. 3-to-2 CSA adders can be replaced by 5-to-3 parallel counters (PCs) as they reduce the number of bits at each position in the sum from 5 to 3. 2 5-to-3 PCs, followed by a CSA and a carry-propagate adder (CPA) can reduce the calculation of variable A, the critical path.

2.4.7 FPGA-Based Cryptography

Private-key cryptographic algorithms are very suitable to be implemented in FPGA. Compared with software-based implementations, symmetric crypto implementations can achieve much higher data throughput. Besides, private-key operations such as bit-permutation, bit-substitution, substitution tables using look-up operations, etc. implemented on FPGA present much better performance than on a general-purpose computer.

Furthermore, crypto algorithms usually include parallel operations that can be more easily exploited at a cryptographic-round level than at block-cipher level [DPR00]. For example, in the ECB mode of operation, multiple blocks can be encrypted in parallel since they are independent from each other. Also, the *Key Expansion* module can work in parallel with the datapath, calculating the subkeys concurrently during encryption/decryption.

Yet another reason one would choose an FPGA as a target platform is security-related. A cryptographic operation running on a general-purpose processor has no countermeasures against side-channel attacks [Sch96]. Dedicated hardware module can contain several countermeasures to not reveal the secret key, the data being manipulated or even the intermediate values used during operation. Nevertheless, NSA authorizes encryption only in hardware [Sch96].

Finally, even if an ASIC equivalent is smaller, consumes less power and has better performance, it still lacks flexibility and it is also very expensive in low-to-medium volumes. FPGA-based solutions can be updated, reprogrammed and improved during the lifecycle of the target product. New standards can be more easily adapted to an FPGA, or a new countermeasure can be implemented and updated on the target system, with less impact and cost than an ASIC.

2.4.8 SHA-2 in Hardware (FPGA and ASIC)

In CHES 2006, a first SHA-2 hardware implementation was proposed [CKSV06]. It starts by briefly enumerating the usual techniques proposed to improve the implementation of SHA-2 algorithm, such as:

— the usage of parallel counters or well balanced Carry-Save Adders (CSA), in order to improve the partial additions, as already discussed in this thesis;
— unrolling techniques that optimize data dependency;
— delay balancing and the usage of improved addition units, since it is the critical operation;
— the usage of embedded memories to store the constant values (K_t);
— use of pipeline techniques to achieve higher working operation frequencies. Pipelining also increases the throughput, although latency is penalized.

In SHA-2, the operations are relatively simple, but the strong data dependency does not allow for much parallelization, since each round of the algorithm can only be computed after the values A to H of the previous round have been calculated. However, it should be noticed that, in each round, the computation is only required to calculate the values of A and E, since the remaining values are obtained directly from the values of the previous round. Based on that, [CKSV06] presents 3 design improvements:

Operation rescheduling Similar to logic balancing, this technique consists in identifying operations that can be computed beforehand, or afterward, to balance the critical path. The values that do not depend on the previous rounds calculations can be calculated in advance. In the case of SHA-2, variables A and E can be calculated earlier, according to the following formulae:

$$A_{t+1} = \sum_{0}(A_t) + \mathrm{Maj}(B_t, C_t, D_t) + \sum_{1}(E_t) + \mathrm{Ch}(E_t, F_t, G_t) + H_t + K_t + W_t$$

$$E_{t+1} = D_t + \sum(E_t) + \mathrm{Ch}(E_t, F_t, G_t) + H_t + K_t + W_t$$

Taking into account that the value H_{t+1} is given directly by G_t which in its turn is given by F_{t-1}, the pre-calculation of H can thus be given by $H_{t+1} = F_{t-1}$. Since the value of K_t and W_t can be pre-calculated and are simply used in each round, the formulae can be rewritten as:

$$\sigma_t = H_t + K_t + W_t = G_{t-1} + K_t + W_t$$

$$A_{t+1} = \sum_{0}(A_t) + \text{Maj}(B_t, C_t, D_t) + \sum_{1}(E_t) + \text{Ch}(E_t, F_t, G_t) + \sigma_t$$

where the value σ_t is calculated in the previous round. The value σ_{t+1} can be the result of a full addition or the Carry and the Save vectors from a Carry-Save Addition. With this computational separation the calculation of SHA-2 can be divided into two parts, allowing the calculation of σ to be rescheduled to the previous clock cycle. While the critical path is greatly reduced, this implementation turns into a pipeline and requires an extra clock cycle to compute, totaling 65 clock cycles (in the case of SHA-256).

Hash value addition and initialization As described in the SHA-2 specification, the variables A to H have to be added to the intermediate result to generate the final message digest. This means that 8 adders would be required. However, some hardware reuse can be engineered once most of the internal variables do not require any computation, since their values are assigned to previous values. Following that:

$H_t = G_{t-1} = F_{t-2} = E_{t-3}$

$D_t = C_{t-1} = B_{t-2} = A_{t-3}$

the computation of the Digest Message (DM) for the data block i can be calculated from the internal variables A and E, where:

$DM7_i = E_{t-3} + DM7_{i-1}$

$DM6_i = E_{t-2} + DM6_{i-1}$

$DM5_i = E_{t-1} + DM5_{i-1}$

$DM3_i = A_{t-3} + DM3_{i-1}$

$DM2_i = A_{t-2} + DM2_{i-1}$

$DM1_i = E_{t-1} + DM1_{i-1}$

Since the values A_t and E_t require the final value, which is only calculated during the last clock cycle, the calculation of $DM0_i$ and $DM4_i$ is performed separately. Instead of using a full adder, after the calculation of the final value of A and E, the Digest Message (DM) is added during the calculation of their final values by a Carry-Save Adder (CSA). Since the value of the previous Digest Message is known, it can be added during the first pipeline stage, not being on the critical path, located on the second stage of the pipeline, where the full adders are used. In the last round the values of A and E are not calculated.

Variable Initialization A multiplexer in front of each variable from A to H selects the newly calculated Digest Message. Again, variables A and E are the exception, because the final computed value of these variables is already the Digest Message.

In the first round, variables A to H have to be initialized. In the standard SHA-2, they receive constant values, but that's not what happens during computation of fragmented messages, where a different [v]*IV*Initialization Vector is used. Therefore, the initial values are not constant anymore. The same multiplexers mentioned above are used to control the loading of *IV* into the core at each new hash iteration. An optimization of this structure is that the calculation structure for the Digest Message can be used to load the *IV*, instead of being directly load to all the registers. The initial values of A and E are reset during the loading step, and the DM registers are connected as a circular buffer, where the value is only loaded into one of the registers, and shifted to the others.

The novel design presented in [CKSV06] is then compared with 3 other related articles [SK] [MCMM06] [HEL05]. It is shown that [CKSV06] achieves better performance when compared to each one of them, as shown in the table below:

Table 2.5: Comparison of different pipelined AES implementations.

Design	FPGA	Freq. (MHz)	T'put (Mbit/s)	Latency (ns)	Area (slices)	Mbit/s / slice	Datapath
[JTS03]	Virtex-E XCV1000E-8	129.2	16,500	-	11,719	1.408	Enc
[SMMS03]	Virtex-E XCV2000E-8	158	20,300	-	5,810 + 100B RAM	1.091	Enc
[SRQL03]	Virtex-E XCV3200E-8	145	18,560	-	15,112	1.228	Enc
[HV04]	Virtex-II Pro XC2VP20	169.1	21,640	420	9,446 wo/ KE	2,290	Enc
[ZNC04]	Virtex-II XC2V4000	184.1	23,570	163	16,938	1.391	Enc
[ZP04]	Virtex-E XCV1000E-8	168.4	21,556	416	11,022 wo/ KE	1.956	Enc/Dec
[GB05]	Virtex-E XCV2000E-8	184.8	23,654	379	16,693	1.417	Enc/Dec

Table 2.6: Comparison of different SHA-2 implementations.

Architecture	[SK]	[CKSV06]	[MCMM06]	[CKSV06]	[HEL05]	[CKSV06]
Device	XCV	XCV	XC2V	XC2V	XC2PV-7	XC2PV-7
Slices	1060	764	1373	797	815	755
Freq. (MHz)	83	82	133	150	126	174
Cycles	n.a.	65	68	65	n.a.	65
T'put (Mbit/s)	326	646	1009	1184	977	1370
T'put / Slice	0.31	0.84	0.74	1.49	1.2	1.83

Chapter 3

Cryptographic Hardware Acceleration and Power Minimization

Summary

The physical limits of CMOS technology down-scaling and the growth of on-chip features cause great complexity to the design of such chips. Therefore, techniques to improve circuit speed and to save power consumption are challengingly important to the success of ASIC product appliance, specially in cryptographically secure chips. For example, a lot has been discussed lately about Bitcoin ASIC power usage. Dedicated ASIC chips have been created to mine Bitcoins, but the tradeoff between how fast these chips can process and how much they actually consume in terms of power will define how efficient they actually are. This chapter presents two lightweight techniques to cope with this scenario applied to secure digital design: Section 3.1 presents a novel way of computing BCH error-correcting codes using the Barrett's modular division. Section 3.2 addresses the energy consumption mitigation problem on system-on-chip and embedded devices by proposing a smart energy management system able to save energy while reducing unresponsiveness penalties at the same time.

The organization of this chapter is as follows: Section 3.1.2 recalls Barrett's algorithm. Section 3.1.3 presents our main theoretical results, i.e., a polynomial variant of [Bar87]. Section 3.1.4 recalls the basics of BCH error-correcting codes (ECC). Section 3.1.5 describes the integration of the Barrett polynomial variant in a BCH circuit and provides benchmark results. Section 3.2.1 introduces the idea of managing energy on SoCs and embedded devices. Section 3.2.2 defines an SoC scenario considering a typical sequence of requests following a given distribution. In Section 3.2.3, we derive an optimal strategy to comply with our previously defined model. While Section 3.2.4 generalizes our model, Section 3.2.5 analyzes an alternative strategy for when we want to keep the unresponsiveness penalty as low as possible.

3.1 BCH with Barrett Polynomial Reduction

3.1.1 Introduction

BCH codes are cyclic codes that form a large class of multiple random error-correcting codes. The original BCH codes were binary codes of length $2^m - 1$. BCH codes were subsequently extended to non-binary settings. Binary BCH codes are a generalization of Hamming codes, discovered by Hocquenghem, Bose and Chaudhuri [BR60, Chi06] featuring a better error correction capability. BCH codes are widely used 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) [Ste94]. 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) [CEGK98].

Gorestein and Zierler [Zie60, GPZ60] generalized BCH codes to p^m symbols (where p is a prime). There are two important BCH code sub-classes. The best known of this sub-classes are Hamming codes for binary BCH codes and Reed Solomon for non-binary BCH codes. 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).

3.1.2 Barrett's Reduction Algorithm

Notations. $\|x\|$ will denote the bit-length of x throughout this paper.

$y \gg z$ will denote binary shift-to-the-right of y by z bits:

$$y \gg z = \left\lfloor \frac{y}{2^z} \right\rfloor .$$

Barrett's algorithm (Algorithm 2) approximates the result $c = d \bmod n$ by a quasi-reduced integer $c + \epsilon n$, where $0 \leq \epsilon \leq 2$. Let $N = \|n\|, D = \|d\|$ and fix a maximal bit-length reduction capacity L such that $N \leq D \leq L$. The algorithm will work if $D \leq L$. 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 an analysis of Algorithm 2.

Example 3.1 *Reduce* $8619 \bmod 93 = 63$.

$n = 93 \Rightarrow N = 7$

$\kappa = \left\lfloor \frac{2^{32}}{n} \right\rfloor$ =10110000001011000000101100
$d = 8619$ =10000110101011
c_1 =10000110~~101011~~ = 10000110
c_2 =10111000011011100001101110000100 0
c_3 =1011100 ~~00110111000011011100001000~~ =1011100
nc_3 =10000101101100
c_4 = 63

Algorithm 2 Barrett's algorithm.

Require: $n < 2^N, d < 2^D, \kappa = \left\lfloor \frac{2^L}{n} \right\rfloor$ where $N \leq D \leq L$
Ensure: $c = d \bmod n$

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** $c_4 \geq n$ **do**
6: $\quad c_4 \leftarrow c_4 - n$
7: **end while**
8: **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 $w + w' = (D-N)(L-N) + (D-N)N = (D-N)L$.

3.1.2.1 Dynamic Constant Scaling

The constant κ can be adjusted on the fly thanks to Lemma 3.1.

Lemma 3.1 *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:

$$\bar{\kappa} = \frac{\kappa}{2^U} - \frac{\alpha}{2^U} \text{ and } \kappa = \frac{2^L}{n} - \frac{\beta}{n}.$$

Therefore,

$$\min_{\alpha\,\beta}\left(\frac{2^{L-U}}{n} - \frac{\beta + \alpha n}{2^U n}\right) \leq \bar{\kappa} = \frac{2^{L-U}}{n} - \frac{\beta + \alpha n}{2^U n} \leq \max_{\alpha,\beta}\left(\frac{2^{L-U}}{n} - \frac{\beta + \alpha n}{2^U n}\right)$$

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 that $\bar{\kappa} = \kappa \gg L - D$. Let $c_5 = D - N + 1$. Replacing step 4 of Algorithm 2 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 $(D-N)^2 + N(D-N) = (D-N)D$.

Example 3.2 *Reconsidering example 3.1, i.e., computing* 8619 *mod* 93 *using the above technique, we obtain:*

$D = \lceil \log_2 8619 \rceil$	$= 14$	
$\bar{\kappa}$	$=10110000\ \sout{00101100000101100}$	
c_1	$=10000110\ \sout{101011}$	$=10000110$
$\bar{\kappa} c_1$	$=101110000100000$	
$\bar{\kappa} c_1 \gg c_5$	$=1011100\ \sout{00100000}$	
$n(\bar{\kappa} c_1 \gg c_5)$	$=10000101101100$	
c_6	$= 63$	

3.1.3 Barrett's Algorithm for Polynomials

3.1.3.1 Orders

Definition 3.1 (Monomial Order) *Let $P,\ Q$ and R be three monomials in ν variables. $\rhd$ is a monomial order if the following conditions are fulfilled:*

— $P \rhd 1$
— $P \rhd Q \Rightarrow \forall R,\ PR \rhd QR$

Example 3.3 *The lexicographic order on exponent vectors defined by*

$$\prod_{i=1}^{\nu} x^{a_i} \succ \prod_{i=1}^{\nu} x^{b_i} \Leftrightarrow \exists i, a_j = b_j \text{ for } i < j \text{ and } a_i > b_i$$

is a monomial order. We denote the lexicographic order by $\succ$.

3.1.3.2 Terminology

In the following, capital letters will next denote polynomials and $\nu \in \mathbb{N}$.

Let $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i}} \in \mathbb{Q}[\vec{x}] = \mathbb{Q}[x_1, x_2, ..., x_\nu]$.

The leading term of P according to $\rhd$, will be denoted by $\mathrm{lt}(P) = p_0 \prod_{j=1}^{\nu} x_j^{y_{j,0}}$.

The leading coefficient of P according to $\rhd$ will be denoted by $\mathrm{lc}(P) = p_0 \in \mathbb{Q}$.

The quotient $\mathrm{lm}(P) = \dfrac{\mathrm{lt}(P)}{\mathrm{lc}(P)} = \prod_{j=1}^{\nu} x_j^{y_{j,0}}$ is the leading monomial of P according to $\rhd$.

The above notations generalize the notion of degree to exponent vectors:

$$\deg(P) = \deg(\mathrm{lm}(P)) = \vec{y_0} = \langle y_{0,0}, \ldots, y_{\nu,0} \rangle.$$

Example 3.4 *For $\succ$ and $P(x,y) = 2x_1^2x_2^2 + 11x_1 + 15$, we have:*

$$lt(P) = 2x_1^2x_2^2,\ lm(P) = x_1^2x_2^2, \deg(P) = \langle 2,2 \rangle \text{ and } lc(P) = 2.$$

Definition 3.2 (Reduction Step) *Let $P,\ Q \in \mathbb{Q}[\vec{x}]$. We denote by $Q \underset{P}{\rightarrow} Q_1$ the* **reduction step** *of Q (with respect to P and according to $\rhd$) defined as the result given by the following operations:*

1. *Find a term t of Q such that monomial(t)=$\mathrm{lm}(P)m$*
2. *If such a t exists, return $Q_1 = Q - \dfrac{Pm}{\mathrm{lc}(P)}$. Else return $Q_1 = Q$.*

Example 3.5 *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 (with respect to P) is $Q \underset{P}{\rightarrow} Q_1 = \dfrac{3x_2}{2}$.

Lemma 3.2 *Let $P,\ Q \in \mathbb{Q}[\vec{x}]$ and $\{Q_i\}$ such that $Q \underset{P}{\rightarrow} Q_1 \underset{P}{\rightarrow} Q_2 \underset{P}{\rightarrow} \ldots$*

1. *$\exists i \in \mathbb{N}$ such that $j \geq i \Rightarrow Q_j = Q_i$*
2. *Q_i is unique*

We denote $Q \xrightarrow[P]{*} Q_i = Q \bmod P$ *and* $\left\lfloor \frac{Q}{P} \right\rfloor = \frac{Q - Q \bmod P}{P} \in \mathbb{Q}[\vec{x}]$ *and call* Q_i *the "residue of* Q *(with respect to* P *and according to* $\triangleright$*)".*

Example 3.6 *Euclidean division is a reduction in which* $i = 1$.

3.1.3.3 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 [TAL97]) polynomial multiplication strategies may grandly improve the following complexities for asymptotically increasing $\vec{L}$ 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 $\vec{\omega} \in \mathbb{Z}^\nu$, in this appendix we denote by $||\vec{\omega}||$ the quantity

$$||\vec{\omega}|| = \prod_{j=1}^{\nu} \omega_j \in \mathbb{Z}.$$

1. $Q \gg \vec{y_0}$.
   (a) **Cost**: $\mathrm{lm}(Q)$ is at most $\langle L, ..., L \rangle$ hence Q has at most L^ν monomials. Shifting discards all monomials having exponent vectors $\vec{\omega}$ for which $\exists j$ such that $\omega_j < y_{j,0}$. The number of such discarded monomials is $\mathrm{O}(||\vec{y_0}||)$, hence the overall complexity of this step is:
   $$\mathrm{cost}_1 = \mathrm{O}((L^\nu - ||\vec{y_0}||)\nu) = \mathrm{O}((L^\nu - \prod_{j=1}^{\nu} y_{j,0})\nu).$$
   (b) **Size**: The number of monomials remaining after the shift is
   $$\mathrm{size}_1 = \mathrm{O}(L^\nu - ||\vec{y_0}||) = \mathrm{O}(L^\nu - \prod_{j=1}^{\nu} y_{j,0}).$$
2. $K(Q \gg \vec{y_0})$.
   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 $\vec{L} - \vec{y_0}$. 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
   $$\mathrm{size}_K = \mathrm{O}((L - y_{1,0})L^{\nu-1}).$$
   (a) **Cost**: The cost of computing $K(Q \gg \vec{y_0})$ is
   $$\mathrm{cost}_2 = \mathrm{O}(\nu \times \mathrm{size}_1 \times \mathrm{size}_K).$$
   (b) **Size**: The size of $K(Q \gg \vec{y_0})$ is determined by $\mathrm{lm}(K(Q \gg \vec{y_0})) = \mathrm{lm}(K) \times \mathrm{lm}(Q \gg \vec{y_0})$ which has the exponent vector $\vec{u} = (\vec{L} - \vec{y_0}) + \langle L - y_{1,0}, L, ..., L \rangle$.
   $$\begin{aligned} \mathrm{size}_2 = \mathrm{O}(||\vec{u}||) &= \mathrm{O}(2(L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0})) \\ &= \mathrm{O}((L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0})). \end{aligned}$$

3. $B = (K(Q \gg \vec{y_0})) \gg (\vec{L} - \vec{y_0})$

   (a) **Cost**: The number of discarded monomials is $\mathrm{O}(||\vec{L} - \vec{y_0}||)$, hence the cost of this step is

   $$\mathrm{cost}_3 = \mathrm{O}((2(L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0}) - \prod_{j=1}^{\nu} (L - y_{j,0}))\nu).$$

   (b) **Size**: The leading monomial of B has the exponent vector $\vec{u} - \vec{L} - \vec{y_0}$ which is equal to $\langle L - y_{1,0}, L, ..., L\rangle$. We thus have $\mathrm{size}_B = \mathrm{size}_K$.

4. BP
   The cost of this step is

   $$\mathrm{cost}_4 = \mathrm{O}(\nu \times \mathrm{size}_B \times \mathrm{size}_P) = \mathrm{O}(\nu \times \mathrm{size}_B \times ||\vec{y_0}||).$$

5. Final subtraction $Q - BP$
   The cost of polynomial subtraction is negligible with respect to cost_4.
6. **Overall complexity**
   The algorithm's overall complexity is hence

   $$\max(\mathrm{cost}_1, \mathrm{cost}_2, \mathrm{cost}_3, \mathrm{cost}_4) = \mathrm{cost}_2.$$

3.1.3.4 Barrett's Algorithm for Multivariate Polynomials

We will now adapt Barrett's algorithm to $\mathbb{Q}[\vec{x}]$.

Barrett's algorithm and Lemma 3.1 can be generalized to $\mathbb{Q}[\vec{x}]$, by shifting polynomials instead of shifting integers.

Definition 3.3 (Polynomial Right Shift) *Let* $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i}} \in \mathbb{Q}[\vec{x}]$ *and* $\vec{a} = \langle a_1, a_2, ..., a_\nu\rangle \in \mathbb{N}^\nu$. *We denote*

$$P \gg \vec{a} = \sum_{\varphi(\vec{a})} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i} - a_i} \in \mathbb{Q}[\vec{x}], \text{ where } \varphi(\vec{a}) = \{i, \forall j, y_{i,j} \geq a_i\}.$$

Example 3.7

$$\text{If } P(x) = 17x^7 + 26x^6 + 37x^4 + 48x^3 + 11, \text{ then } P \gg \langle 5\rangle = 17x^2 + 26x.$$

Theorem 3.3 (Barrett's Algorithm for Polynomials) *Let:*

— $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,0}} \in \mathbb{Q}[\vec{x}]$ *and* $Q = \sum_{i=0}^{\beta} q_i \prod_{j=1}^{\nu} x_j^{w_{j,i}} \in \mathbb{Q}[\vec{x}]$ *s.t.* $\mathrm{lm}(Q) \rhd \mathrm{lm}(P)$

— $L \geq \max(w_{i,j}) \in \mathbb{N}, h(L) = \prod_{j=1}^{\nu} x_j^L$ *and* $K = \left\lfloor \frac{h(L)}{P} \right\rfloor$

— $\vec{y_0} = \langle y_{1,0}, y_{2,0}, ..., y_{\nu,0}\rangle \in \mathbb{N}^\nu$

Given the above notations, $(K(Q \gg \vec{y_0})) \gg (\langle L\rangle^\nu - \vec{y_0}) = \left\lfloor \frac{Q}{P} \right\rfloor$.

Proof: Let $G = h(L) \bmod P$ and $B = (\mathrm{K}(Q \gg \vec{y_0})) = \frac{h(L) - G}{P} \left\lfloor \frac{Q}{\mathrm{lm}(P)} \right\rfloor$.

$$\Downarrow$$

$$B = \frac{\sum_{\varphi(\vec{y_0})} q_i \prod_{j=1}^{\nu} x_j^{L + w_{j,i} - y_{j,0}} - G \sum_{\varphi(\vec{y_0})} q_i \prod_{j=1}^{\nu} x_j^{w_{j,i} - y_{j,0}}}{P}$$

Applying the definition of "$\gg$", we obtain

$$B \gg (\langle L\rangle^{\nu} - \vec{y_0}) = \deg_{\geq \vec{0}} \frac{Q_{\varphi(\vec{y_0})} - G \sum\limits_{\varphi(\vec{y_0})} q_i \prod\limits_{j=1}^{\nu} x^{w_{j,i}-L}}{P}, \text{ where } \vec{0} = \langle 0\rangle^{\nu}.$$

Thus,

$$B \gg (\langle L\rangle^{\nu} - \vec{y_0}) = \left\lfloor \frac{Q_{\varphi(\vec{y_0})}}{P} \right\rfloor - \deg_{\geq \vec{0}} \frac{G}{P} \sum_{\varphi(\vec{y_0})} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L} = \left\lfloor \frac{Q_{\varphi(\vec{y_0})}}{P} \right\rfloor.$$

We know that

$$P \rhd G \text{ and } L \geq \max(w_{i,j}), \text{ therefore } \deg_{\geq \vec{0}} \frac{G}{P} \sum_{\varphi(\vec{y_0})} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L} = 0.$$

Let $\bar{Q}$ be the irreducible polynomial with respect to P, obtained by removing from Q the terms that exceed $\mathrm{lm}(P)$.

$$\left\lfloor \frac{Q_{\varphi(\vec{y})}}{P} \right\rfloor = \frac{Q_{\varphi(\vec{y})} - (Q_{\varphi(\vec{y})} \bmod P)}{P} = \frac{(Q-\bar{Q})((Q-\bar{Q}) \bmod P)}{P}.$$

Hence,

$$B \gg (\langle L\rangle^{\nu} - \vec{y_0}) = \frac{(Q-\bar{Q})((Q-\bar{Q}) \bmod P)}{P}$$

$$\Downarrow$$

$$B \gg (\langle L\rangle^{\nu} - \vec{y_0}) = \left\lfloor \frac{Q}{P} \right\rfloor - \frac{\bar{Q} - \bar{Q} \bmod P}{P} = \left\lfloor \frac{Q}{P} \right\rfloor.$$

□ □

Algorithm 3 Polynomial Barrett Algorithm.

Require:

$$P, Q \in \mathbb{Q}[\vec{x}] \text{ such that } \prod_{j=1}^{\nu} x_j^L \rhd Q \text{ i.e. } \langle L\rangle^{\nu} \geq \deg Q$$

$$\text{We denote } \vec{y_0} = \deg P \text{ and } K = \left\lfloor \frac{\prod_{j=1}^{\nu} x_j^L}{P} \right\rfloor$$

Ensure: $R = Q \bmod P$

1: $B \leftarrow (K(Q \gg \vec{y_0})) \gg (\langle L\rangle^{\nu} - \vec{y_0})$
2: $R \leftarrow Q - BP$
3: **return** R

Remark. Let $Q = \sum\limits_{i=0}^{\alpha} q_{i,j} \prod\limits_{j=1}^{\nu} x_j^{w_{j,i}}$, $K = \sum\limits_{i=0}^{\beta} k_{i,j} \prod\limits_{j=1}^{\nu} x_j^{t_{j,i}}$, $\vec{y} = \langle y_1, ..., y_\nu\rangle$ and $\vec{z} = \langle z_1, ..., z_\nu\rangle$.

Let us have a closer look at the expression $B = (K(Q \gg \vec{y})) \gg \vec{z}$.

Given the final shifting by $\vec{z}$, the multiplication of K by $Q \gg \vec{y}$ can be optimized by being only partially accomplished. Indeed, during multiplication, we only have to form monomials whose exponent vectors $\vec{b} = \vec{w_i} + \vec{t_{i'}} - \vec{y} - \vec{z} = \langle b_1, ..., b_\nu \rangle$ are such that $b_j \geq 0$ for $1 \leq j \leq \nu$.

We implicitly apply the above in the following example.

Example 3.8 *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 $L = 6$ and we observe that $\nu = 2$. We pre-compute K:

$$\begin{aligned} K = {} & x_1^4x_2^4 - x_1^4x_2^2 + x_1^4 - 2x_1^3x_2^4 - 2x_1^3x_2^3 + 3x_1^3x_2^2 + 4x_1^3x_2 - 4x_1^3 + \\ & 4x_1^2x_2^4 + 8x_1^2x_2^3 - 5x_1^2x_2^2 - 20x_1^2x_2 + 3x_1^2 - 8x_1x_2^4 - 24x_1x_2^3 + \\ & 68x_1x_2 + 36x_1 + 16x_2^4 + 64x_2^3 + 36x_2^2 - 184x_2 - 239. \end{aligned}$$

We first shift Q by $\vec{y_0} = \langle 2, 2 \rangle$, which is the vector of exponents for $lm(P)$.

$$Q \gg \vec{y_0} = (x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3) \gg \langle 2, 2 \rangle = (x_1x_2 + 1)$$

Then, we compute $K(x_1x_2 + 1) = x_1^5x_2^5 - 2x_1^4x_2^5 - x^4y^4 + \{terms \prec x_1^4x_2^4\}$.

This result shifted by $\langle L \rangle^\nu - \vec{y_0} = \langle 6, 6 \rangle - \langle 2, 2 \rangle = \langle 4, 4 \rangle$ to the right gives:

$$A = x_1^5x_2^5 - 2x_1^4x_2^5 - x^4y^4 + \{terms \succ x_1^4x_2^4\} \gg \langle 4, 4 \rangle = x_1x_2 - 2x_2 - 1.$$

It is easy to verify that:

$$\begin{gathered} Q - PA = \\ = (x_1^3x_2^3 - 2x_1^3 + x_1^2x_2^2 + 3) - (x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1)(x_1x_2 - 2x_2 - 1) \\ \Downarrow \\ Q - PA = 4x_1x_2^3 + 6x_1x_2^2 - x_1^3x_2 + x_1^2x_2 + 3x_1x_2 + 2x_2 - 2x_1^3 + x_1^2 + x_1 + 4 \prec P. \end{gathered}$$

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 $w + w' = (D-N)(L-N) + (D-N)N = (D-N)L$.

Complexity: We refer the reader to Appendix A for a detailed computation of the complexity of Algorithm 3.

Note that the complexity of the partial multiplication considered instead of the standard multiplication is $O((\beta - \alpha)\beta)$.

3.1.3.5 Dynamic Constant Scaling in $\mathbb{Q}[\vec{x}]$

Lemma 3.4 *If $0 \leq u \leq L$, then $\bar{K} = K \gg \langle u \rangle^\nu = \left\lfloor \frac{h(L-u)}{P} \right\rfloor$.*

Proof: $K = \left\lfloor \frac{h(L)}{P} \right\rfloor \Rightarrow K = \frac{h(L) - h(L) \bmod P}{P}$.

Let $G = h(L) \bmod P \Rightarrow K = \dfrac{\prod_{j=1}^{\nu} {x_j}^L - G}{P}$.

Since

$$\langle u \rangle^\nu \in \mathbb{N}^\nu \Rightarrow K \gg \langle u \rangle^\nu = \deg_{\geq \vec{0}} \frac{\prod_{j=1}^{\nu} {x_j}^{L-u} - G_{\varphi(\langle u \rangle^\nu)}}{P}$$

$$\Downarrow$$

$$K \gg \langle u \rangle^\nu = \deg_{\geq \vec{0}} \frac{\prod_{j=1}^{\nu} {x_j}^{L-u}}{P} - \deg_{\geq \vec{0}} \frac{G_{\varphi(\langle u \rangle^\nu)}}{P}.$$

We know that $P \rhd G$, thus $P \rhd G_{\varphi(\langle u \rangle^\nu)}$, thus $\deg_{\geq \vec{0}} \dfrac{G_{\varphi(\langle u \rangle^\nu)}}{P} = 0$.

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 3.9 *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 $\bar{K}$:

$$\bar{K} = x_1^2x_2^2 - x_1^2 - 2x_1x_2^2 - 2x_1x_2 + 3x_1 + 4x_2^2 + 8x_2 - 5.$$

We first shift Q by $\vec{y_0} = \langle 2, 2 \rangle$, which is the vector of exponents for $lm(P)$.

$$Q \gg \vec{y_0} = (x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3) \gg \langle 2, 2 \rangle = (x_1x_2 + 1)$$

Then, we compute $\bar{K}(x_1x_2 + 1) = x_1^3x_2^3 - 2x_1^2x_2^3 - x_1^2x_2^2 + \{terms \prec x_1^2x_2^2\}$.

This result shifted by $\langle u \rangle^\nu - \vec{y_0} = \langle 4, 4 \rangle - \langle 2, 2 \rangle = \langle 2, 2 \rangle$ to the right gives:

$$A = x_1^3x_2^3 - 2x_1^2x_2^3 - x_1^2x_2^2 + \{terms \succ x_1^2x_2^2\} \gg \langle 2, 2 \rangle = x_1x_2 - 2x_2 - 1.$$

It is easy to verify that:

$$Q - PA =$$

$$= (x_1^3x_2^3 - 2x_1^3 + x_1^2x_2^2 + 3) - (x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1)(x_1x_2 - 2x_2 - 1)$$

$$\Downarrow$$

$$Q - PA = 4x_1x_2^3 + 6x_1x_2^2 - x_1^3x_2 + x_1^2x_2 + 3x_1x_2 + 2x_2 - 2x_1^3 + x_1^2 + x_1 + 4 \prec P.$$

3.1.4 Application to BCH Codes

3.1.4.1 General Remarks

BCH codes are cyclic codes that form a large class of multiple random error-correcting codes. Originally discovered as binary codes of length 2^m-1, BCH codes were subsequently extended to non-binary settings. Binary BCH codes are a generalization of Hamming codes, discovered by Hocquenghem, Bose and Chaudhuri [BR60], [Chi06] featuring a better error correction capability. Gorestein and Zierler [Zie60, GPZ60] generalized BCH codes to p^m symbols, for p prime. Two important BCH code subclasses exist. Typical representatives of these sub-classes are Hamming codes (binary BCH) and Reed Solomon codes (non-binary BCH).

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.

3.1.4.2 BCH Preliminaries

Definition 3.4 *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:*

$$\mathrm{BCH}(n,d) = \{(c_0, c_1, ..., c_{n-1}) \in \mathbb{F}_2^n \mid c(x) = \sum_{i=0}^{n-1} c_i x^i \text{ satisfies}$$

$$c(\alpha) = c(\alpha^2) = ... = c(\alpha^{d-1})\}$$

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 3.5 *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 $\alpha, \alpha^2, ..., \alpha^{2t}$.*

Definition 3.6 *Let $\phi_i(x)$ be the minimal polynomial of α^i. Then,*

$$g(x) = \mathrm{lcm}\{\phi_1(x), \phi_2(x), ..., \phi_{2t}(x)\}.$$

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:

$$g(x) = \mathrm{lcm}\{\phi_1(x), \phi_3(x), \phi_3(x), ..., \phi_{2t-1}(x)\}.$$

Definition 3.7 (Codeword) *An $n-$tuple $c = (c_0, c_1, ..., c_{n-1}) \in \mathbb{F}_{2^n}$ is a codeword if the polynomial $c(x) = \sum c_i x^i$ has $\alpha, \alpha^2, ..., \alpha^{2t}$ as its roots.*

Definition 3.8 (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 [1] to every codeword of C, i.e.:*

$$C^\perp = \{v \in \mathbb{F}_q^n | v \cdot c = 0, \forall c \in C\}.$$

1. The scalar product of the two vectors is equal to 0.

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:

Property: If $c = (c_0, c_1, ..., c_{n-1})$ is a codeword, then the parity-check matrix H of this code satisfies $c \cdot H^T = 0$, where:

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \dots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \dots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \dots & (\alpha^{2t})^{n-1} \end{pmatrix}.$$

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.

Remark The minimum distance of the previously defined $t-$error correcting BCH code is at least $2t+1$.

Definition 3.9 (Systematic Encoding) *In systematic encoding, information and check bits are concatenated to form the message transmitted over the noisy channel.*

The speed-up described in this paper 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 $c(x) = m'(x) - b(x)$ [2].

3.1.4.3 BCH Decoding

Syndrome decoding is a decoding process for linear codes using the parity-check matrix.

Definition 3.10 (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 $S = (S_1, S_2, \cdots, S_{2t})$.

Decoding consists of the following four steps:

1. Compute the syndrome S for the received codeword.
2. Determine the number of errors t and find the error-locator polynomial coefficients $\Lambda_i(x)$ from S.
3. Compute the roots of the error-locator polynomial to infer the error locations.
4. Compute the error values Y_i at the identified error locations and correct them.

2. where $b(x)$ is the remainder of the division of $c(x)$ by $g(x)$

3.1.4.4 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

$$S_i = r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j) = \sum_{k=0}^{\nu-1} e_k \alpha^{ik}, \text{ for } i \leq 1 \leq 2t.$$

Suppose that r has ν errors denoted e_{j_i}. Then

$$S_i = \sum_{j=1}^{\nu} e_{j_i} (\alpha^i)^{j_\ell} = \sum_{j=1}^{\nu} e_{j_i} (\alpha^{j_\ell})^i.$$

3.1.4.5 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_ℓs are called *error locators* and the *error-locator polynomial* is defined as:

$$\Lambda(x) = \prod_{\ell=1}^{\nu} (1 - X_\ell) = 1 + \Lambda_1 x + ... + \Lambda_\nu x^\nu.$$

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.* Peterson's algorithm [GP07] or Berlekamp-Massey algorithm [Hoc59]. Chien's search method [MS77] is applied to determine the roots of $\Lambda(x)$.

3.1.4.6 Peterson's Algorithm

Peterson's Algorithm 4 solves a set of linear equations to find the value of the coefficients $\sigma_1, \sigma_2, \ldots \sigma_t$.

$$\Lambda(x) = \prod_{\ell=1}^{\nu} (1 + \alpha^{jl}) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_t x^t$$

At the beginning of Algorithm 4, the number of errors is undefined. Hence the maximum number of errors to resolve the linear equations generated by the matrix S is assumed. Let this number be $i = \nu = t$.

3.1.4.7 Chien's Error Search

Chien's search finds the roots of $\Lambda(x)$ by brute force [Chi06], [MS77]. 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's search circuits stems from Equation 3.1 for $\Lambda(\alpha^{i+1})$.

$$\begin{aligned}
\Lambda(\alpha^i) &= 1 + \sigma_1\ \alpha^i + \sigma_2\ (\alpha^i)^2 + \cdots + \sigma_t\ (\alpha^i)^t \\
&= 1 + \sigma_1\ \alpha^i + \sigma_2\ \alpha^{2i} + \cdots + \sigma_t\ \alpha^{it} \\
\Lambda(\alpha^{i+1}) &= 1 + \sigma_1\ \alpha^{i+1} + \sigma_2\ (\alpha^{i+1})^2 + \cdots + \sigma_t\ (\alpha^{i+1})^t \\
&= 1 + \alpha\ (\sigma_1\ \alpha^i) + \alpha^2\ (\sigma_2\ \alpha^{2i}) + \cdots + \alpha^t\ (\sigma_t\ \alpha^{it})
\end{aligned} \tag{3.1}$$

Algorithm 4 Peterson's algorithm.

Initialization $\nu \leftarrow t$

Compute the determinant of S

$$\det(S) \leftarrow \det \begin{pmatrix} S_1 & S_2 & \cdots & S_t \\ S_2 & S_3 & \cdots & S_{t+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_t & S_{t+1} & \cdots & S_{2t-1} \end{pmatrix}$$

Find the correct value of ν

$$\begin{cases} \det(S) \neq 0 \longrightarrow \text{go to step 4} \\ \det(S) = 0 \longrightarrow \begin{cases} \textbf{if } \nu = 0 \textbf{ then} \\ \quad \text{The error-locator polynomial is empty} \\ \quad \text{stop} \\ \textbf{else} \\ \quad \nu \longleftarrow \nu - 1, \text{ and then repeat step 2} \\ \textbf{end if} \end{cases} \end{cases}$$

Invert S and compute $\Lambda(x)$

$$\begin{bmatrix} \sigma_\nu \\ \sigma_{\nu-1} \\ \vdots \\ \sigma_1 \end{bmatrix} = S^{-1} \times \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{2\nu} \end{bmatrix}$$

3.1.5 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. Five BCH encoder versions were designed and synthesized. Results are presented in detail in the forthcoming sections.

3.1.5.1 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[3] 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 5 provides the pseudocode for the `BCH-standard` architecture.

Algorithm 5 Standard modular division (`BCH-standard`).

Require: P, Q
Ensure: `remainder` $= Q \bmod P$

```
1:  diff_degree ← deg(Q) − deg(P)
2:  shift_counter ← diff_degree + 1
3:  shift_divisor ← P ≪ diff_degree
4:  remainder ← Q
5:  while shift_counter ≠ 0 do
6:      if remainder[p_degree + shift_counter − 1] = 1 then
7:          shift_counter ← shift_counter − 1
8:          shift_divisor ← shift_divisor ≫ 1
9:      end if
10: end while
11: return remainder
```

3. Considered in big endian order.

3.1.5.2 LFSR and Improved LFSR Architectures

The BCH-LFSR design is composed of a control unit and a Linear-Feedback Shift Register (LFSR) sub-module. The LFSR sub-module 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 Fig. 3.1, 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.

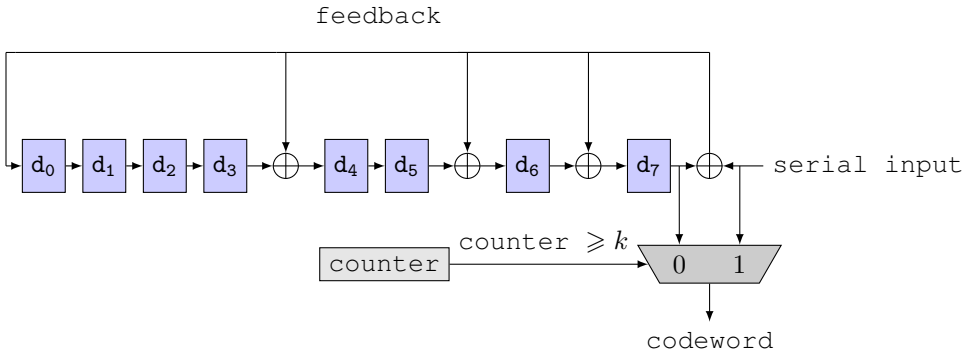


Figure 3.1: Standard LFSR architecture block diagram. (Design BCH-LFSR).

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 Fig. 3.2.

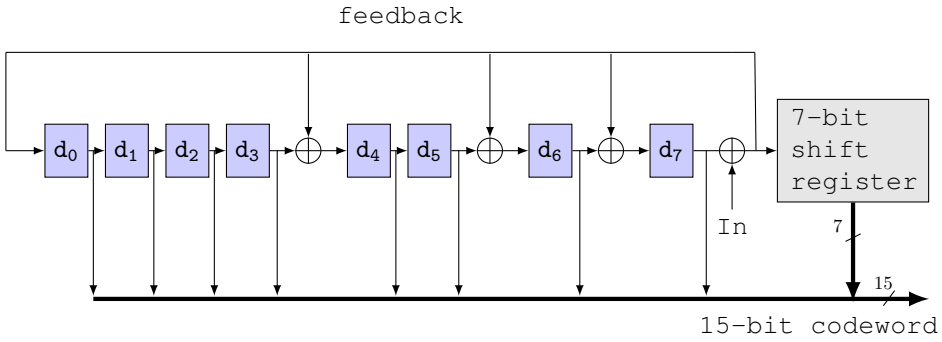


Figure 3.2: Improved LFSR architecture block diagram. In denotes the module's serial input. (Design BCH-LFSR-improved).

3.1.5.3 Barrett Architecture (regular and pipelined)

The LFSR sub-module can be replaced by the Barrett sub-module to evaluate its performance. Two Barrett implementations were designed: the first computes all the Barrett steps in one clock cycle, while the second approach, a pipelined block, works with the idea 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.

In the Barrett sub-module, 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 $L - y_0$.

3.1.5.4 Performance

The gate equivalent (GE) metric is the ratio between the total cell area of a design and the size of the smallest NAND-2 cell of the digital library. This metric allows comparing circuit areas while abstracting away technology node sizes. *FreePDK45* (an open source 45nm Process Design Kit [BD15]) was used as a digital library to map the design into logic cells. Synthesis results were generated by Cadence Encounter RTL Compiler RC13.12 (v13.10-s021_1). `BCH-Barrett` presented an area comparable to the smallest design (`BCH-LFSR`). Although `BCH-Barrett` does not reach the maximum clock frequency, Table 3.1 shows that it actually reaches the best throughput among the non-pipelined designs, around 2.08Gbps. The `BCH-Barrett-pipelined` achieves the best throughput, but it represents the biggest area and the more power consuming core. This is mainly due to the parallelizable nature of Barrett's operations, allowing the design to be easily pipelined and therefore further speed-up. The extra register barriers introduced in `BCH-Barrett-pipelined` forces the design to present bigger area and a higher switching activity, which increases power consumption.

Table 3.1: Synthesis results of the four BCH designs.

Design	Gate Instances	Gate Equivalent	Max Frequency (MHz)	Throughput (Mbps)	Power (μW)
`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

3.2 Managing Energy on SoCs and Embedded Systems

3.2.1 Introduction

Energy consumption is becoming a key concern in today's systems-on-chip (SoCs) as embedded devices [THM15] are required to process more and more data, at faster rates.

Consider a SoC $\mathcal{S}$ receiving cryptographic computation requests at unpredictable points in time. Each request must be responded immediately, otherwise unavailability causes a penalty. Alternatively, $\mathcal{S}$ may decide to go idle to save energy and extend its battery lifetime.

$\mathcal{S}$ may therefore be in one of two states: either manage requests ($\mathcal{A}$) or go idle ($\mathcal{B}$). While $\mathcal{S}$ is busy attending a request, it does not accept other requests until calculations are complete and the response is sent back to the requester:

The system status $\mathcal{A}$ means that $\mathcal{S}$ is available to answer requests.

The system status $\mathcal{B}$ means that $\mathcal{S}$ went idle to save energy.

Our goal is to endow $\mathcal{S}$ with a smart energy management system so $\mathcal{S}$ may go idle to save energy when incoming request probability is low enough to take such a risk. Status $\mathcal{A}$ dissipates power, therefore $\mathcal{S}$ would like to go idle as much as possible. Thus $\mathcal{S}$ should be in $\mathcal{B}$ mode as much as possible, to save energy, but should also be reasonably available to answer requests to reduce unresponsiveness penalties.

3.2.2 The Model

We use the following model:

— $x(t)$ is the number of incoming requests per time unit. The function $x(t)$ can be estimated by performing statistics on previously processed requests.

— γ is $\mathcal{S}$'s average power consumption per time unit. We assume that when $\mathcal{S}$ is in idle mode ($\mathcal{B}$), $\mathcal{S}$ does not consume energy.

— Answering an incoming request (state $\mathcal{A}$) consumes one (normalized) unit of energy.

— Let t denote the time spent by $\mathcal{S}$ to service a request. The total power consumed during t is γt.

— α is the penalty incurred by an incoming request while $\mathcal{S}$ is in state $\mathcal{B}$ (idle). If $\mathcal{S}$ is in $\mathcal{A}$-mode, no penalty is incurred as the incoming request can be immediately processed by $\mathcal{S}$. α is hence the (virtual) amount of energy consumed to "wake up". A large α means that inter-request idleness periods must be reduced to a minimum.

We consider a typical sequence of requests following a distribution given by $x(t)$. We denote by E the total power consumed by the system during its operation and by N the number of penalties incurred over this distribution. We define the total power consumption function:

$$P = E + N \cdot \alpha \tag{3.2}$$

The total power consumption P is thus the sum of the usefully consumed energy plus the total unresponsiveness penalties, over the distribution $x(t)$. Note that to properly add-up, α's unit must be an energy consumption value (same unit as E).

Note that if we take $\alpha = 0$ (no unresponsiveness penalty), then only the E power component must be minimized. In this case $\mathcal{S}$ immediately goes idle ($\mathcal{B}$-mode) as soon as $\mathcal{S}$ (which is in $\mathcal{A}$-mode) finishes processing a request.

On the other hand, for a large α the cost E in equation (3.2) becomes negligible compared to the penalty $N \cdot \alpha$. In that case only the number of penalties N may be minimized. Put differently, since $\mathcal{S}$ does not

tolerate unresponsiveness, the best strategy is simply to make $\mathcal{S}$ always available to service requests, which means that $\mathcal{S}$ never goes into a $\mathcal{B}$-mode.

Define $f(N)$ as the probability to cause a system malfunction before N penalties occur. f increases monotonously with N. It is hence desirable to reduce N as much as possible but... not at the cost of a too high power consumption.

For "medium" values of unresponsiveness sensitivity α, our goal is therefore to find a strategy that minimizes the total power consumption function P as determined by equation (3.2). More precisely, given as input $x(t), \gamma, \alpha$, our goal is to determine when $\mathcal{S}$ should go idle and when $\mathcal{S}$ should better switch back to $\mathcal{A}$-mode to await incoming requests.

Let $f(n-1) \leq \rho \leq f(n)$ we set $\alpha \triangleq \ell/n$. As a sanity-check we see that if $\mathcal{S}$ has an extremely strong aversion to system unresponsiveness, its n will be equal to 1 resulting in $\alpha = \ell$. At the other extreme, if $\mathcal{S}$'s policy does not care about unanswered requests then $n = \infty$ and $\alpha = 0$.

3.2.3 Optimizing Power Consumption While Avoiding System Malfunction

We now derive an optimal strategy to comply with equation (3.2).

We consider an observed increase ΔP in the total power consumption function during a short time period ΔT.

According to our model, ΔP comprises the cost of the usefully consumed power during ΔT, and possibly a penalty if an incoming request occurred when the system was in $\mathcal{B}$-mode. We distinguish four possible cases during the period ΔT:

1. $\mathcal{S}$ was in $\mathcal{B}$-mode and no request occurred.

   ▶ In this case no power is consumed and no penalty is incurred, so $\Delta P = 0$.

2. $\mathcal{S}$ was in $\mathcal{A}$-mode and no request occurred.

   ▶ By definition the power consumed by $\mathcal{S}$ during ΔT is $\gamma \cdot \Delta T$; therefore $\Delta P = \gamma \cdot \Delta T$.

3. $\mathcal{S}$ was in $\mathcal{B}$-mode and a request occurred.

   ▶ The power consumed by $\mathcal{S}$ for processing the request is 1 plus a penalty α because $\mathcal{S}$ was unable to service the request immediately. Therefore $\Delta P = 1 + \alpha$.

4. $\mathcal{S}$ was in $\mathcal{A}$-mode and a request occurred.

   ▶ No penalty is incurred and $\Delta P = 1$, which is the normalized power consumed for treating the incoming request.

Table 3.2: Increase ΔP of the power consumption function in a short time period ΔT.

	$\mathcal{S}$ was in $\mathcal{B}$-mode	$\mathcal{S}$ was in $\mathcal{A}$-mode
request	$1 + \alpha$	1
no request	0	$\gamma \cdot \Delta T$

We now determine the average increase of P during the time period ΔT. During ΔT the probability p to witness an incoming request is approximated [4] by:

$$p = x(t) \cdot \Delta T$$

4. Assume, for instance, that $x(t) = 10$ requests/hour and consider a one second time interval $\Delta T = 1/3600$ hours. The probability to witness a request during ΔT is indeed $10\Delta T = 1/360$. During each one second time interval the probability to witness a request is $1/360$ and over 3600 seconds we indeed get an average of 10 requests. Hence, p is indeed the probability to witness a request between time t and $t + \Delta T$ when ΔT is very small.

where $x(t)$ is the number of incoming requests per time unit.

Therefore, when the system is idle ($\mathcal{B}$-mode), with probability p there is an incoming request and the power consumption is increased by $1+\alpha$, whereas with probability $1-p$ the power consumption remains invariant. The variation $\Delta P_{\mathcal{B}}$ of the power consumption while being in $\mathcal{B}$-mode is therefore:

$$\Delta P_{\mathcal{B}} = p \cdot (1+\alpha) + (1-p) \cdot 0 = (1+\alpha) \cdot x(t) \cdot \Delta T \tag{3.3}$$

Similarly, when the system is in $\mathcal{A}$-mode we obtain the following average variation $\Delta P_{\mathcal{A}}$ of the power consumption:

$$\Delta P_{\mathcal{A}} = p \cdot 1 + (1-p) \cdot \gamma \cdot \Delta T = x(t) \cdot \Delta T + (1 - x(t) \cdot \Delta T) \cdot \gamma \cdot \Delta T$$

Neglecting the terms in ΔT^2, we get:

$$\Delta P_{\mathcal{A}} \simeq \big(x(t) + \gamma\big) \cdot \Delta T \tag{3.4}$$

From equations (3.3) and (3.4) we obtain:

$$\Delta P_{\mathcal{B}} \leq \Delta P_{\mathcal{A}} \quad \Leftrightarrow \quad (1+\alpha) \cdot x(t) \leq x(t) + \gamma$$

which gives our main result:

$$\boxed{\Delta P_{\mathcal{B}} \leq \Delta P_{\mathcal{A}} \Leftrightarrow \alpha \cdot x(t) \leq \gamma}$$

We can therefore distinguish two cases:

1. If $\alpha \cdot x(t) \leq \gamma$, then $\Delta P_{\mathcal{B}} \leq \Delta P_{\mathcal{A}}$. Power consumption is minimized by having $\mathcal{S}$ go idle;
2. If $\alpha \cdot x(t) > \gamma$, then $\Delta P_{\mathcal{B}} > \Delta P_{\mathcal{A}}$. Here it is more advantageous to switch $\mathcal{S}$ into $\mathcal{A}$-mode (even in the absence of an incoming request).

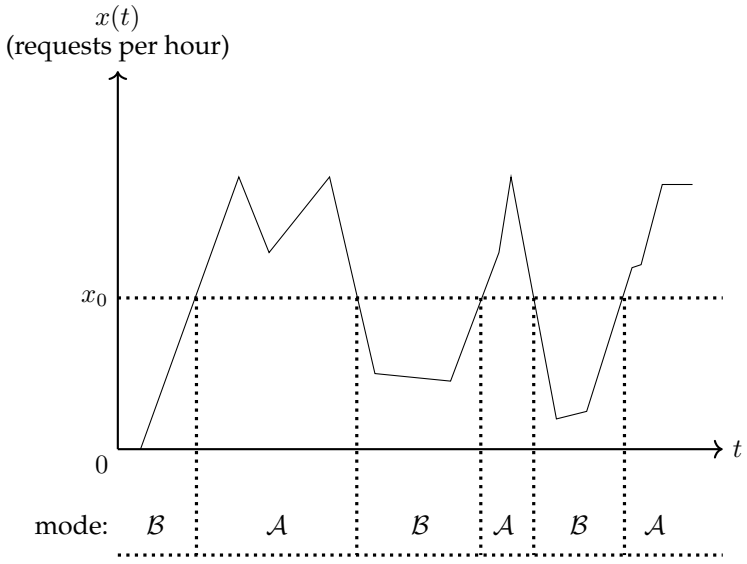


Figure 3.3: Example of a request function $x(t)$ with a threshold x_0. When $x(t) \geq x_0$, it is more advantageous to be in $\mathcal{A}$-mode. Otherwise $\mathcal{S}$ should better go into $\mathcal{B}$-mode.

Finally, we can define an incoming request frequency threshold $x_0 \triangleq \frac{\gamma}{\alpha}$.

As illustrated in Fig. 3.3, when $x(t) \leq x_0$, it is more advantageous to enter $\mathcal{B}$-mode, and when $x(t) > x_0$, it is more advantageous to stay in $\mathcal{A}$-mode. We note that this strategy does not depend on the energy

cost u necessary to process a request; here we assumed that $u = 1$, but the strategy would be the same for any value of u; this is because energy is spent whenever a request occurs, no matter if $\mathcal{S}$ was in $\mathcal{B}$ or in $\mathcal{A}$-mode.

Finally, we note that this strategy is clearly optimal since at any time t we are minimizing the increase in the power consumption function.

3.2.4 The General Case

Assume that the power consumed to process a request is u (instead of 1), and that the amounts of energy dissipated while in $\mathcal{A}$ and $\mathcal{B}$ modes (respectively $r_{\mathcal{A}}$ and $r_{\mathcal{B}}$) are potentially nonidentical and differ from γ (Table 2).

Table 3.3: Increase ΔP of the power consumption function in a short time period ΔT. $(r_{\mathcal{A}} + r_{\mathcal{B}})/2$ represents the average rate due to the alternation of modes $\mathcal{A}$ and $\mathcal{B}$ during the request.

	$\mathcal{S}$ was in $\mathcal{B}$-mode	$\mathcal{S}$ was in $\mathcal{A}$-mode
request	$u + \alpha + (\gamma - (r_{\mathcal{A}} + r_{\mathcal{B}})/2) \cdot \Delta T$	$u + (\gamma - r_{\mathcal{A}}) \cdot \Delta T$
no request	$(\gamma - r_{\mathcal{B}}) \cdot \Delta T$	$\gamma \cdot \Delta T$

An analysis, similar to that of the previous section yields:

$$\Delta P_{\mathcal{B}} \leq \Delta P_{\mathcal{A}} \quad \Leftrightarrow \quad -2r_{\mathcal{B}} + 2\alpha x(t) + r_{\mathcal{A}} x(t)\Delta T + r_{\mathcal{B}} x(t)\Delta T < 0$$

That is (neglecting the terms with ΔT):

$$\Delta P_{\mathcal{B}} \leq \Delta P_{\mathcal{A}} \quad \Leftrightarrow \quad \alpha \cdot x(t) < r_{\mathcal{B}}$$

As expected u vanished and when $r_{\mathcal{B}} = \gamma$ this yields x_0.

3.2.5 Probabilistic Strategies

In this section we analyze an alternative strategy for $\mathcal{S}$. Here, $\mathcal{S}$ generates a function $0 \leq v(t) \leq 1$ and tosses a biased coin (probability $v(t)$ for tail and $1 - v(t)$ for head) during each interval ΔT. If the coin falls on a head $\mathcal{S}$ will switch into $\mathcal{A}$-mode; otherwise, $\mathcal{S}$ will switch into $\mathcal{B}$-mode.

For such a strategy we need:

$$v(t) \cdot p \cdot (1 + \alpha) < (1 - v(t)) \cdot (p \cdot 1 + (1 - p) \cdot \gamma \cdot \Delta T)$$

Substituting p by $x(t)\Delta T$:

$$v(t) \cdot x(t)\Delta T \cdot (1 + \alpha) < (1 - v(t)) \cdot (x(t)\Delta T \cdot 1 + (1 - x(t)\Delta T) \cdot \gamma \cdot \Delta T)$$

Expanding, neglecting the terms in ΔT^2 and dividing by ΔT we get:

$$v(t) \cdot x(t) + \alpha \cdot v(t) \cdot x(t) < \gamma - \gamma \cdot v(t) + x(t) - v(t) \cdot x(t) \Rightarrow v(t) \cong \frac{\gamma + x(t)}{\gamma + (2 + \alpha)x(t)}$$

In other words, we get a “mirror" coin-toss function $v(t)$ that attempts to correct the variation of $x(t)$ by increasing or decreasing the coin-toss probability to keep the penalty as low as possible. This is well illustrated in the following graphics where we plotted two $x(t)$ functions (in blue) and their corresponding $v(t)$ (in purple). We see that a burst in $x(t)$ is immediately compensated by a descent of $v(t)$ and *vice versa*.

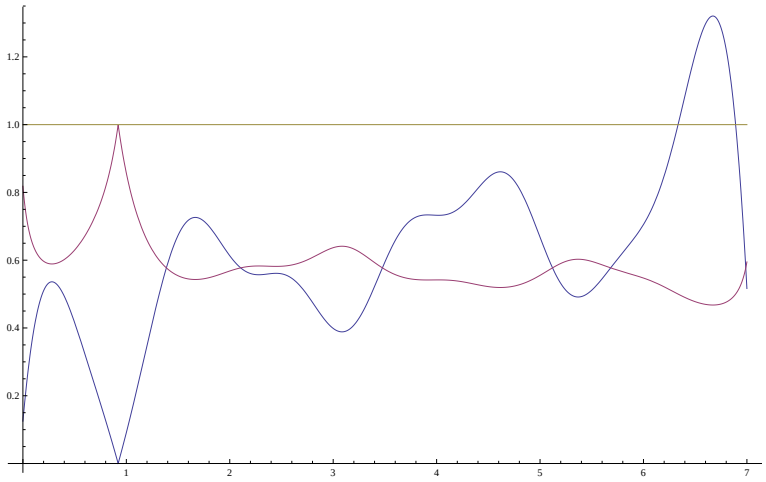


Figure 3.4: Example of $v(t)$ (purple) for an example function $x(t)$ (blue).

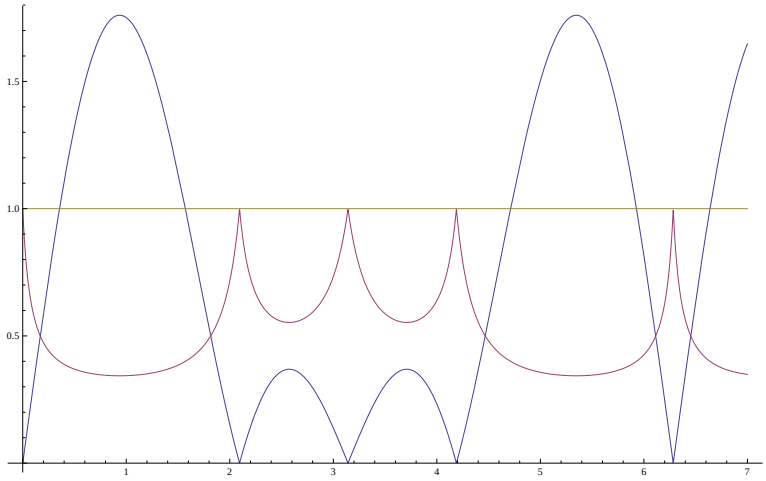


Figure 3.5: Example of $v(t)$ (purple) for $x(t) = |\sin(x) + \sin(2x)|$ (blue).

Chapter 4

Side-Channel Attacks and Hardware Countermeasures

Summary

This chapter presents in more detail the basis of side-channel attacks and proposes hardware countermeasures against power and fault attacks.

Section 4.1 introduces side-channel attacks from an economical perspective. Sections 4.2 and 4.3 define Differential Cryptanalysis and Differential Power Analysis, respectively. A lightweight countermeasure is presented in Section 4.4, in which a reconfigurable AES implementation makes use of the algorithm's features to create protection mechanisms against fault and power attacks. Section 4.5 presents CSAC, a cryptographically secure on-chip firewall. CSAC uses a standardized HMAC function to authenticate the reprogramming agent, binding the ability of modify system-on-chip read and write permissions to a trusted actor. Moreover, CSAC is able to detect fault attacks aiming at modifying the content of the permission rules already set on the firewall. ASIC and FPGA implementations show the tradeoffs of different protection levels. This chapter is closed by Section 4.6, in which a novel side-channel attack based on practical Instantaneous Frequency Analysis experiments is introduced. We show that, in addition to the signals amplitude and spectrum, traditionally used for side-channel analysis, instantaneous frequency variations may also leak secret data.

4.1 An Economical Introduction to Side-Channel Attacks

The existing information and communication systems require the implementation of components that can withstand high encryption rates while resisting a multitude of identified attacks. The need for integration in a very short cycle of new security platforms requires flexible components, capabilities and expandable rich computational possibilities. Such components must be usable for the development of products and also to be able to take advantage of a variety of applications and services.

Unfortunately, the opening of platforms in the "cloud" and the outsourcing of a lot of the data processing to third party networks, and primarily the Internet, has exposed computers to attacks on hardware, software and operating systems.

The increased level of trust in hardware developments such as the addition of firewalls or the introduction of tri-factor authentication methods, caused systems to slowdown, which in turn caused the user discomfort. It is clear that the use of security software solutions has reached a tipping point in negative investment.

These new problems create a strong need for trusted components intended to accelerate network communications. The problem is complex because beyond the technical problem of accelerating calculations, it is essential that protected platforms conform to the imperatives of national sovereignty – especially (but not limited to) when such platforms are intended for defense applications.

The typical user wants a system that is easy to operate and configure, protected from malware and resistant to physical attacks that could endanger equipment or sensitive data. Insofar as it is impossible to fully predict the exact use of the security device, it is imperative that all trusted components at least allow and ensure the safety of personal data, respecting the privacy and the integrity of the system in which the component is integrated.

Nowadays, there is an increasing need for new hardware-software solutions that meet the key challenges of creating sensitive applications. These devices have to be engineered to allow the host system to perform cryptographic operations at very high speed and benefit from a choice of resources, while providing the host defenses against malicious attacks or application changes.

Side-channel attacks (SCA), a type of cryptanalysis described as an implementation attack, extract secret information from physical signals emanating from operating circuits. Such signals may be the power consumption, electromagnetic radiation or the time required to accomplish a certain calculation [Koc96]. Side-channel attacks are also categorized as passive attacks. Passive attacks do not change the state of the target but allow to extract sensitive information by just being able to spy on it.

The side-channel attacks were introduced as a major threat by Kocher in 1996 [Koc96]. The principle is to observe the physical properties of the component and then retrieve information about the secret that is being used. An attack analyzing calculation time was the first proposed side-channel attack. Kocher [Koc96] used the differences in time between particular RSA executions to find bits of secret data. Sometimes variations may depend on the value of the processed data. Thus, in the case of the Original Montgomery modular multiplication [Mon85], a subtraction of the modulus may be required. The extra time can therefore reflect this subtraction, which can then be used by an attacker to retrieve bits of the secret exponent used in an RSA signature for example. Time measurements, however, require great precision to be exploitable.

The vast majority of secure components use the CMOS technology because it is cheap and effective. One of the most interesting properties of this technology is that the static (or leakage) power consumption of a CMOS circuit is low (presented in details in Section 2.1.5.2). This corresponds to the consumption of the circuit when it is in an equilibrium state. A noticeable and significant power consumption change is seen when CMOS transistors change state. This power component is called *dynamic power consumption*. More generally, a CMOS circuit consumes a significant amount of energy only when there is a switching activity in transistors and wires. There is a strong relationship between the component's consumption and the number of bits that change state at a given instant.

As explained in deeper details in Section 2.1.3, the inverter is a basic structure of a CMOS circuit. The inverter consists of p-channel transistor arrays of Metal Oxide Semiconductor (PMOS) and n-channel

Metal Oxide Semiconductor (NMOS). When no operation is performed, the voltage is either V_{dd} or V_{ss} at the input and output. However, during a transition from V_{dd} to V_{ss} (or vice versa) at the input, there is a short time during which a short circuit current flows through the transistor. Also at this time, the load capacitances, such as buses or logic gates, are charged or discharged.

The power consumption of a component can be observed by measuring the potential difference divided by the resistance using a resistor connected in series between the component's external power supply and V_{ss}. Then a digital oscilloscope is used to scan the stream and save it on a computer. The equipment required for the acquisition of consumption curves is fairly common. A computer is used first to send commands to the smart card to launch operations on it with given parameters. A first current sensor measures the consumption while a second one is used to trigger the acquisition of consumption curves by the oscilloscope. This second probe is thus linked to the input signal sent to the card. A stable power supply allows to fix with consistency and precision the current intensity flowing through the circuit. Attacks by current analysis, or power analysis, pose a very serious threat due to their effectiveness and ease of implementation.

Attacks using the electromagnetic emission of a component are similar to attacks done by measuring current, as described above. Electromagnetic measurement attacks are based on the fact that low power loads that are in motion produce a magnetic field which itself generates an electric field. Current microprocessors are made up of millions of transistors and interconnections that generate these electromagnetic emissions. This property was used to perform side channel attacks on cryptographic components.

[GMO01] were the first to provide experimental results of smart card on attacks using electromagnetic emissions. Agrawal et al. [AARR03] then proposed a more comprehensive study of this kind of side channel attack. They show that this type of covert channel can be used when an attacker cannot gain access to the target to measure of current consumption. In addition, they use electromagnetic radiation to break the countermeasures that resist attacks by power analysis. Electromagnetic emissions are often seen as giving very precise information on the data processed by the component. Nevertheless, it is very difficult, in practice, to obtain optimal information. Numerous criteria such as where to place the probe to record these emissions, or the size and type of probe to be used have yet to be characterized precisely.

4.2 Differential Cryptanalysis

Before introducing DPA in mathematical detail, let us first introduce what is defined as *Differential Cryptanalysis* [BS93]. The motivation for Differential Cryptanalysis comes from the fact that the private-key cipher operations are linear, except for the S-Boxes, therefore mixing the key in all the rounds prohibits the attacker from knowing which entries of the S-Boxes are actually used. Because of that, the attacker cannot know their output. Differential Cryptanalysis studies differences between two different plaintexts and the difference between the two corresponding ciphertexts to infer information about the key.

Consider a chosen plaintext attack model, where the adversary can play with the cryptosystem as a black-box (i.e., without any knowledge of internal operations or data) and submit plaintext messages at its input, receiving the corresponding ciphertext message as the output. The aim of this attack is to be able to recognize patterns and learn any information of the key, given the plaintext-ciphertext pair, therefore revealing the secret key.

The basic idea of differential cryptanalysis is to submit pairs of plaintext blocks P and P^* of which the difference is a fixed value P', i.e., $P' = P \oplus P^*$. We then analyze the ciphertext pair T and T^* difference until we find a fixed value T', with $T' = T \oplus T^*$. A first analysis phase consists of heuristically searching for *good* P' and T' values. Formally, the differential probability is the quantity defined as

$$\text{DP}^f(P', T') = \Pr[f(\delta + P') = f(\delta) + T']$$

where f is the encryption function and δ is a uniformly distributed random variable. The attack is considered to be efficient for high numbers of this probability.

4.3 Differential Power Analysis

DPA, first introduced by Kocher in [KJJ99], is a sophisticated technique that uses a set of power measurements to perform statistical analysis on small power differences, aiming at recovering the secret key. The remaining of this section details important nomenclature related to DPA.

Power Trace. The power consumed by a hardware cryptosystem depends on the data manipulated by its internal circuits and can be measured by an oscilloscope or a digital data acquisition board by inserting a resistor in series with the ground or power supply pin. The collected data is called a *power trace*. The power trace might contain a single encryption operation, or several operations in sequence.

Simple Power Analysis (SPA). This technique involves a straightforward interpretation of a power trace. Although the scope of this thesis does not cover the public-key cryptography in details, we define the algorithm below in order to explain how SPA works.

Algorithm 6 Computation of an RSA signature (modular exponentiation).

Require: $m, N, d = \{d_{k-1}, ..., d_0\}$ and $\mu : \{0,1\}^* \longrightarrow \mathbb{Z}/N\mathbb{Z}$
Ensure: $S = \mu(m)^d \bmod N$

1: $R_0 \leftarrow 1$
2: $R_1 \leftarrow \mu(m)$
3: **for** $j \leftarrow k-1, 0$ **do**
4: $\quad R_0 \leftarrow {R_0}^2 \bmod N$
5: $\quad$ **if** $d_j = 1$ **then**
6: $\quad\quad R_0 \leftarrow R_0 \times R_1 \bmod N$
7: $\quad$ **end if**
8: **end for**
9: **return** R_0

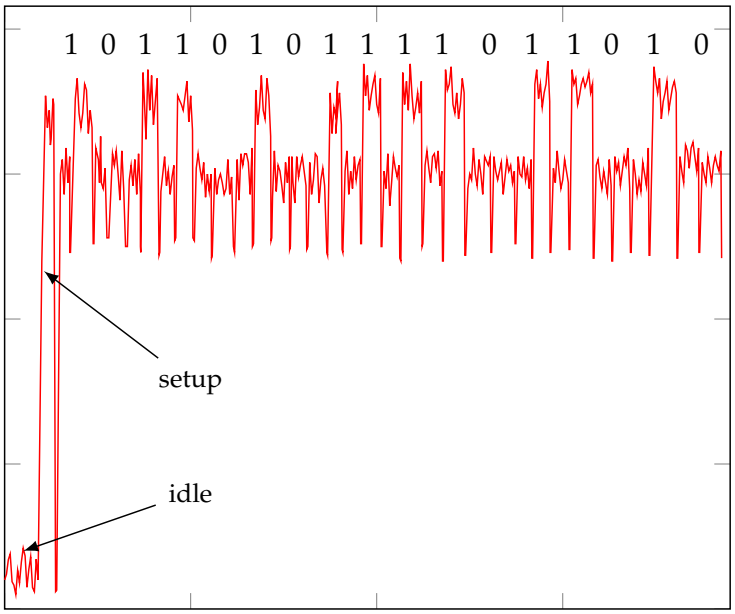


Figure 4.1: Power trace of an RSA exponentiation.

Algorithm 6 details the necessary steps to calculate an RSA digital signature, that has the purpose to guarantee the authenticity and the integrity of a given message. There are two keys involved in

the process: a private signing key and a public verification key. Typically, the production of a digital signature of a message m using RSA consists on the "hash-and-sign" paradigm [Koc08]: m is first hashed into $\mu(m)$ and the result is then raised to the d-th power modulo N, $S = \mu(m^d) \bmod N$, where d denotes the private RSA key. The public verification key is $\{e, N\}$ where $ed \equiv 1 \bmod \phi(N)$ and ϕ is Euler's totient function. The validity of the signature S of message m is verified by checking whether $S^e \equiv \mu(m) \bmod N$.

Algorithm 6 details a common way of implementing a modular exponentiation using the square-and-multiply algorithm. At iteration j of the algorithm's main loop, a modular squaring is performed. If bit d_j of d is equal to 1, a modular multiplication is also performed. As we can see from that, there are two distinct operations to be performed in RSA: modular squaring and modular multiplication. The power consumed to evaluate the RSA signature is correlated to the square and multiply operations, and therefore correlated to the bits of the secret key d.

Fig. 4.1 shows the RSA exponentiation of the first 16 bits. The trace starts by showing small activity due to idleness, followed by a big energy peak, due to the algorithm's setup. After that, we easily recognize two patterns: a lower power consumption trace, corresponding to the squaring $R_0 \leftarrow {R_0}^2 \bmod N$, and a higher power consumption trace, corresponding to multiplications defined as $R_0 \leftarrow R_0 \cdot R_1 \bmod N$. We can therefore infer that a lower pattern followed by another lower trace represents a bit $d_i = 0$, while a lower pattern followed by a higher trace represents the bit $d_i = 1$. In Fig. 4.1, 16 bits of the private RSA key are revealed by only analyzing the power consumed by the algorithm's execution.

***Hamming-weight* and *Hamming-distance* models.** The most common and simplest model to characterize the information leakage through the power consumption is called the Hamming weight, i.e., the number of non-zero bits in a given bit string of the manipulated data of the executed instruction.

Another commonly used model is the Hamming distance, that considers the number of bits flipped bits in the current state compared with the previous state. Let HW denotes the Hamming-weight function and st_t denotes the bit string state at clock cycle t. The Hamming distance is given by

$$\mathrm{HW}(\mathsf{st}_t \oplus \mathsf{st}_{t-1})$$

where $\oplus$ denotes the XOR (exclusive OR) operator that isolates in a string the number of flipping bits.

Bit Tracing. It is not always straightforward to locate a cipher operation in a power trace. Imagine that on a multi-purpose SoC the cipher operation occurs after a series of writes to the RAM, followed by a DRM computation. The power consumed by the cipher operation might not be visible just by looking at of the power trace shape.

Let σ denote a *Boolean selection function* which returns $\sigma(y) = 0$ or $\sigma(y) = 1$ depending on the value of y; let $\langle\cdot\rangle$ represent averaging and $\varphi_P(t)$ denote the power consumption of process P at time period t. After collecting several power traces of a same process P, a partition of two sets, ς_0 and ς_1, is created depending on a *known* intermediate value y:

$$\varsigma_0 = \{y \mid \sigma(y) = 0\}$$

and

$$\varsigma_1 = \{y \mid \sigma(y) = 1\}$$

We define the DPA trace as

$$\Delta_P(t) := \langle\varphi_P(t)\rangle_{\varsigma_1} - \langle\varphi_P(t)\rangle_{\varsigma_0}$$

namely, the difference of the average power consumption curve corresponding to sets ς_1 and ς_0 for each time period t.

The DPA trace magnifies the effect of the selection function σ. Suppose that a cryptographic process is performed on an 8-bit microcontroller that respects the Hamming-weight model. Suppose also that the 16-byte ciphertexts are known. Let us define σ as a selection function that returns the value of a given bit of the first ciphertext byte. Therefore sets ς_0 and ς_1 will contain ciphertexts for which a given bit is always 0 and always 1, respectively. As a result, the average Hamming-weight value for the first byte of ciphertexts in set ς_0 will be 3.5, and ς_1 will present an average value of 4.5. As the Hamming-weight model reflects the power consumption as a function of the Hamming weight of the manipulated data, this difference between the average Hamming weights will translate into a difference between the average power consumption for set ς_0 and set ς_1 when the first byte of the ciphertext is being manipulated, causing a peak in the DPA trace. By trying different ciphertext bits and comparing the peaks presented in the power trace, the attacker can correctly locate the point in time when the cipher operation occurs.

DPA Attack on AES. Besides locating the cipher operation in time, DPA can also be used to recover secret information. Although the attacker does not known the intermediate value of the key and therefore it is not possible to make a partition on intermediate values during the cipher operation, the following steps can be followed to discover the bits of the secret key [Koc08]:

— The attacker guesses a key value;
— The attacker applies the DPA methodology to some intermediate value depending on the key (and depending also on the cryptographic operation in question);
— If the DPA trace does not present peaks then the guessed key value was wrong and the attacker goes back to the first step. Otherwise, the key value is correct.

To illustrate this, let us consider a 128-bit AES. The plaintext m is first XORed with the key and then gradually updated by applying round functions *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey* in a series of 10 rounds. The plaintext m is defined as a 4×4 matrix of bytes $m_i = ({s_{u,v}}^{(i)})$, where $0 \leq u, v \leq 3$. The *SubBytes* function substitutes each byte $s^{(i)}_{u,v}$ by another byte through a non-linear permutation $SBox$. Since the *SubBytes* transformation has an effect on the whole byte value, the selection function σ can be defined as the value of a given output bit of $SBox(s^{(i)}_{u,v})$.

The following steps are followed by an attacker willing to discover the secret key:

— The attacker makes a guess on the key byte $k_{u,v}$, choosing one value among 256 possible combinations;
— The attacker partitions the t random 128-bit messages

$$m_1 = (s^{(1)}_{u,v}, ...,\ m_t = ({s_{u,v}}^{(t)}))$$

in two sets ς_0 and ς_1 given by

$$\varsigma_0 = \{m_i \mid \text{bit}_1(SBox(s^{(i)}_{u,v})) = 0\}$$
$$\varsigma_1 = \{m_i \mid \text{bit}_1(Sbox(s^{(i)}_{u,v})) = 1\}$$

where $\text{bit}_1(SBox(s^{(i)}_{u,v})$ denotes the first bit of byte $SBox(s^{(i)}_{u,v})$.
— Next, the DPA trace is constructed by

$$\Delta_P(t) := \langle\varphi_P(t)\rangle_{\varsigma_1} - \langle\varphi_P(t)\rangle_{\varsigma_0}$$

and the attacker observes whether there are significant DPA peaks. If not, the attacker goes back to the initial step with another guess for $k_{u,v}$. Otherwise, the attacker iterates the same attack to find the remaining key bytes.

This attack requires to use at most 256 DPA traces to recover one key byte and at most $256 \times 16 = 4096$ DPA traces to recover the cipher key.

Countermeasures. Since the first publication of side-channel attacks against cryptosystems, various countermeasures have been proposed to defend hardware and software against the correlation between the power consumed by the device and the data being manipulated by it.

A first class of countermeasures consists in reducing the available side-channel signal, which implies avoiding conditional branches where the secret information leads the algorithm to perform different operations [Tyg96]. In other words, the algorithm is coded in a way that the sequence of operations are regular and always performed in a given order.

Another class of countermeasures involves introducing noise to the power channel, making the power trace less or impossible to read. This includes power smoothing, which purpose is to render power traces smoother. The insertion of *wait states* in hardware or *dummy cycles* in software is another example, where the traces become misaligned and therefore harder to analyze. Another example at the hardware level is to make use of an external unstable source of clock, with unstable period and jitter. At software level, but also commonly used in hardware, *data masking* is yet another example of a popular class of countermeasures, usually giving excellent resilience against power attacks since the data manipulated is *masked* by a random and unrelated mask value, making the secret data uncorrelated to the actual unmasked value.

Efficiency is key when implementing a cryptosystem protected against side-channel attacks. Countermeasures must be proved resilient against power correlation while maintaining an acceptable performance level. It is common to apply several different classes of countermeasures to a single cryptosystem to increase side-channel resistance. It is therefore important to find lightweight alternatives to implement cryptographic primitives that use the best of hardware resources without compromising the security level.

4.4 Power Scrambling and the Reconfigurable AES

4.4.1 Introduction

The Advanced Encryption Standard (AES) algorithm, also known as Rijndael, is a widely used block-cipher standardized by NIST in 2001 [AES01]. Compared with its predecessor DES [AG01], the AES features longer keys, larger plaintexts and more involved basic binary transformations [BBK+03].

Despite the fact that AES is mathematically safer than the DES, straightforward AES implementations are not necessarily secure and several authors [Koc96, KJJ99, MOP07] have exhibited ways of exploring information that leaks from AES implementations. Such leakage is typically power consumption, electromagnetic emanations or the time required to process data. Additional constraints such as fault resistance, chip technology, performance, area, power consumption, and even patent compliance further complicate the design of real-life AES coprocessors.

This chapter addresses resistance against two physical threats on AES: power and fault attacks. The proposed AES architecture leverages the algorithm's structure to create low-cost protections against these attacks. This allows very flexible runtime configurability without significantly affecting performance.

The remaining of this chapter is organized as follows: Section 4.4.2 recalls the AES' main features and proposes an architecture for implementing it. Section 4.4.3 explains how to add power scrambling and fault detection to the proposed implementation. The result is a chip design allowing 29 different software-controlled runtime configurations. Section 4.4.4 introduces an idea of reducing the memory required to store *state keys* in the decryption mode. Section 4.4.5 compares simulation and synthesis results between an unprotected AES and our protected implementations.

4.4.2 The Proposed AES Design

The AES is a symmetric iterative block-cipher that processes 128-bit blocks and supports keys of 128, 192 or 256 bits [AES01]. Key length is denoted by $N_k = 4, 6,$ or 8, and reflects the number of 32-bit words in the key. At start, the 128-bit plaintext P is split into a 4×4 matrix S of 16 bytes called *state*. The *state* goes through a number of rounds to become the ciphertext C.

The number of rounds N_r is a function of N_k. Possible $\{N_r, N_k\}$ combinations are $\{10, 4\}$, $\{12, 6\}$ and $\{14, 8\}$. A particular round $1 \leq r \leq N_r$ takes as input a 128-bit *state* $S^{[r]}$ and a 128-bit *round key* $K^{[r]}$ and outputs a 128-bit *state* $S^{[r+1]}$. This is done by successively applying four transformations called *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*.

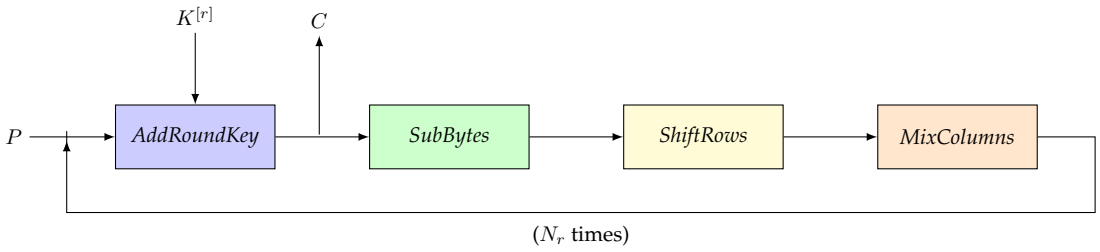


Figure 4.2: AES encryption flowchart.

AES encryption starts with an initial *AddRoundKey* transformation followed by N_r rounds consisting of four transformations, in the following order: *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*. *MixColumns* is skipped in the final round ($r = N_r$). If during the last round *MixColumns* is bypassed, we can look upon the AES as the 4-block iterative structure shown in Fig. 4.2.

Decryption has a similar structure in which the order of transforms is reversed (Fig. 4.3) and where inverse transformations are used (Note that *AddRoundKey* is idempotent). In both designs, a register barrier at the end of each transformation block is used to save intermediate results. Therefore the

intermediate information that eventually yields $S^{[r]}$ is saved four times during each AES round. It takes $4N_r + 1$ clock cycles to encrypt (or decrypt) a data block using this design.

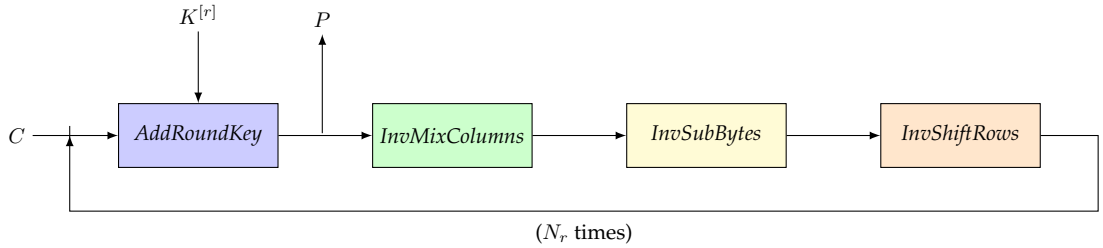


Figure 4.3: AES decryption flowchart.

Fig. 4.2 and Fig. 4.3 show that, during each clock cycle, only one block of the chain actually computes the *state*, while the other three blocks are processing useless data. This is potentially risky, as the three concerned blocks "chew" computationally useless data related to P (or C) and $K^{[r]}$ and thereby expose the design to unnecessary side-channel attacks. This computation is shown in Fig. 4.4 where red arrows represent the path of usefully active combinatorial logic.

4.4.3 Energy and Security

4.4.3.1 Power Analysis

To benchmark our design the AES was implemented on FPGA. Power was measured at 1GS/s sampling rate with 250 MHz bandwidth using PicoScope 3407A oscilloscope. To guarantee the identical conditions every new plaintext was given to the FPGA at the same clock after the reset.

We performed a Correlation Power Attack (CPA) on the first AES S-Box output since S-Box operation is generally considered as the most power gluttonous. Our power model was based on the number of flipped register's bits in the S-Box module when the initial register's barrier R_0 is rewritten with the S-Box output as follows:

$$\text{HD}(S - Box[P \oplus K_0], R_0) = \text{HW}(S - Box[P \oplus K_0] \oplus R_0) \tag{4.1}$$

where R_0 is the previous register's state; P is a given plaintext; K_0 is the AES master key.

The value R_0 was assumed to be constant since all the encryptions were performed at the same clock after the reset. When R_0 could not be computed then all possible 256 values were tried. Pearson correlation coefficient was used to link the model and the genuine consumed power.

The following section presents a reference evaluation of the unprotected AES implementation showing its vulnerability compared to two (LFSR and tri-state buffers) side-channel countermeasures introduced later.

4.4.3.2 Power Scrambling

It is a natural idea to shut down unnecessarily active blocks. To do so, each block receives a new 1-bit input named *ready* activating the block when *ready* $= 1$. If *ready* $= 0$, the block's pull-up resistors are disconnected using a tri-state buffer connected to the power source. This saves power and also prevents the circuit from leaking "unnecessary" side-channel information.

Logically the pipeline architecture that we have just described has to be less vulnerable against First Order DPA attacks. Its four register barriers introduce additional noise, so we expect that the correlation shall be at least smaller that for the AES design with one round per clock computation.

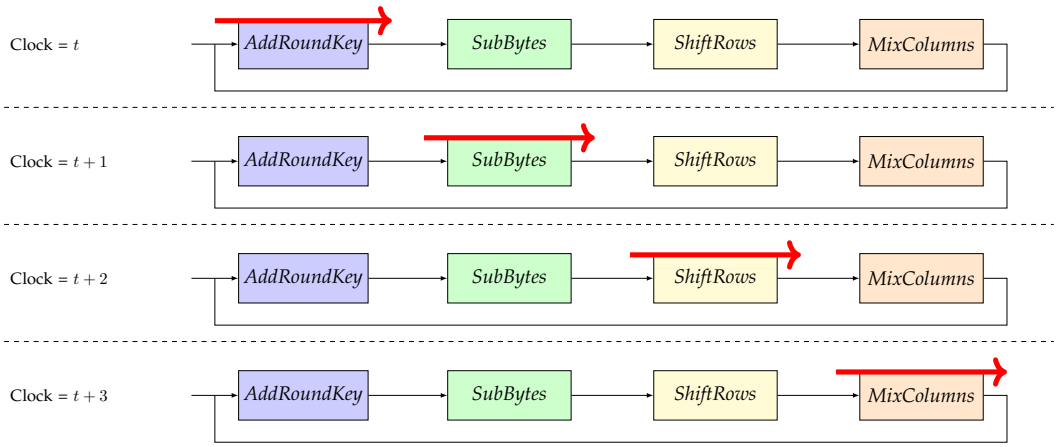


Figure 4.4: Flow of computation in time.

To asses the security of each proposed design, we will compare an incorrect key byte correlation to a correct key byte correlation. Fig. 4.5 shows these two coefficients. As expected, the correct key is correlated to the power traces, however even for 500,000 traces Pearson correlation coefficient is smaller than 0.015. Anyway, this implementation is vulnerable.

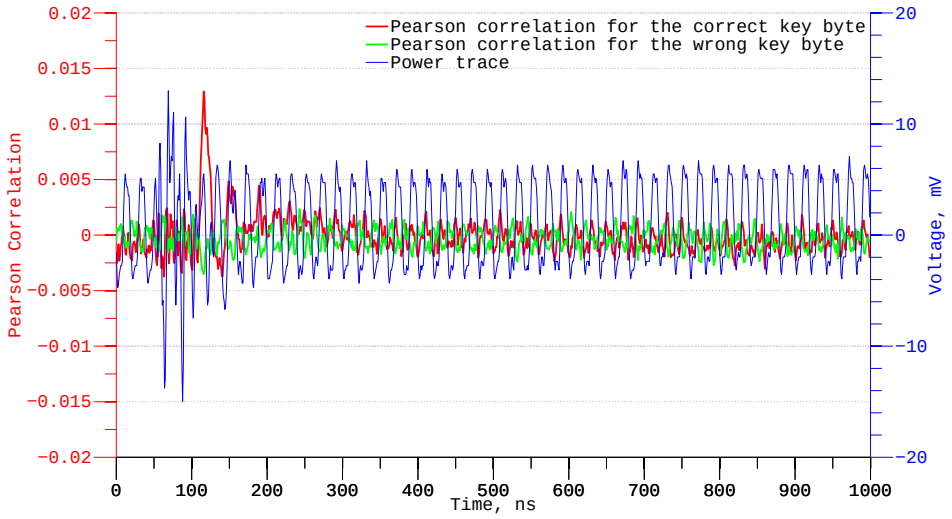


Figure 4.5: Unprotected implementation: Pearson correlation value of a correct (red) and an incorrect (green) key byte guess. 500,000 power traces.

To exploit the unused blocks to hide the device's power signature even better we propose two modifications. The first consists in injecting (pseudo) random data into the unused blocks, making them process that random data. Subsequently, three of the four blocks will consume power in an unpredictable manner. Note that because we use the exact same gates to compute and to generate noise, the expected spectral and amplitude characteristics of the generated noise should mask leakage quite well. Although any random generator may be used as a noise source, we performed our experiments using a 128-bit LFSR. An LFSR is purely coded in digital HDL, making tests easier to implement.

Fig. 4.6 shows that a multiplexer controlled by the *ready* signal selects either the useful intermediate *state* information or the pseudo-random LFSR output. For the *AddRoundKey* block, LFSR data replaces the key. Therefore when *AddRoundKey*'s *ready* $= 0$, pseudo-random data (unrelated to the key) are XORed with the *state* coming from the previous block (*MixColumns* if encrypting, *InvShiftRows* if decrypting). For the other blocks, the pseudo-random data replaces the *state* when *ready* $= 0$.

Attacks performed on this implementation revealed that this countermeasure increases key lifetime.

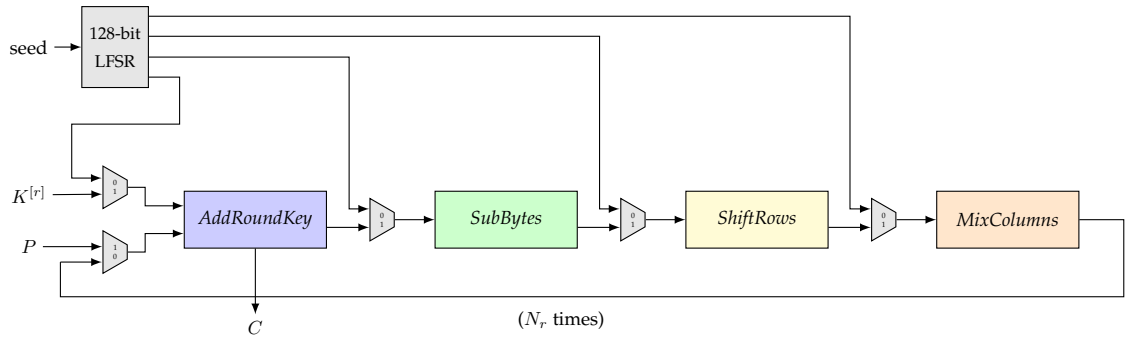


Figure 4.6: Power scrambling with a PRNG.

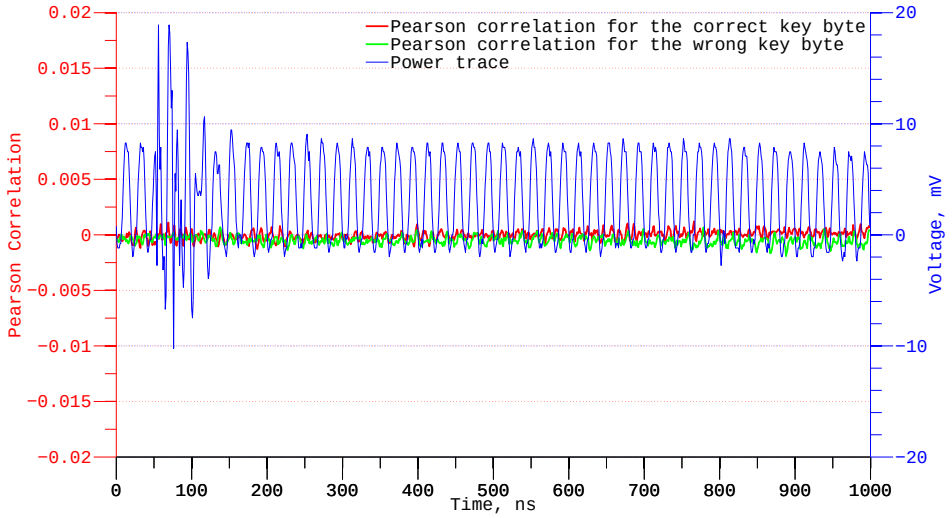


Figure 4.7: LFSR implementation: Pearson correlation value of a correct (red) and an incorrect (green) key byte guess. 1,200,000 power traces.

Fig. 4.7 is the equivalent of Fig. 4.5 for the protected implementation using an LFSR. The correct key correlation can not be distinguished from the incorrect key correlation even with 1,200,000 traces. However, we assume that this implementation still might be vulnerable if more traces are acquired or if Second Order DPA is applied.

Real-life implementations must use true random generators. Indeed, if a deterministic PRNG seed is used the noise component in all encryptions becomes constant and cancels-out when computing differential power curves.

A second design option interleaves tri-state buffers between blocks to hide power consumption. By shutting down the three useless blocks, we create a scrambled power trace where one block computes meaningful data while the other three "process" high impedance inputs, which means that these blocks "compute" leakage current coming from their inputs.

As illustrated in Fig. 4.8, the input signal $ready_i$ determines which blocks are tri-stated and which block is computing the AES *state*. In other words, the $ready_i$ signal "jumps" from one block to the next, so that only one block is computing while the other three are scrambling the power consumption. Although this solution has a smaller overhead in terms of area (as it does not require random number generation) tri-state buffers tend to be slow. Furthermore, the target environment (FPGA or IC digital library) must offer tri-state cells.

The experimental results we obtained on FPGA were surprising, we couldn't attack the design with 800,000 power traces. The correlations shown in Fig. 4.9 do not allow to visually distinguish the correct key from a wrong guess. As before we assume that this implementation can be still attackable if more

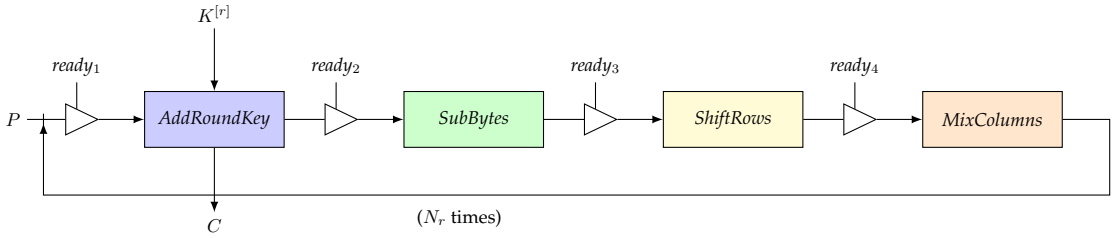


Figure 4.8: Power scrambling with tri-state buffers.

power traces are acquired or if Second Order DPA is applied.

A full study of this solution would require an ASIC implementation with real tri-state buffers, as an FPGA emulates these buffers and may turn out to be resistant because of an undesired CLB mapping side effects.

Figure 4.9: Tri-state buffers implementation: Pearson correlation value of the correct key byte (green) and a wrong key byte guess (red). 800,000 power traces.

4.4.3.3 Transient Fault Detection

We will now use idle blocks to check for transient faults. Each block in the chain can "stutter" during two consecutive clock cycles to recompute and check its own calculation. For instance, as shown in Fig. 4.10, at clock t, a given block B_i receives a $ready_i$ signal, computes the *state* and saves it in the register barrier R_i. At clock $t+1$, the result enters the next block $B_{i+1 \bmod 4}$ which is now working, while B_i reverts to checking, i.e., B_i recomputes the same output as at clock t and compares it to the saved B_i value. This process is repeated for the other blocks in the chain. If any transient fault happens to cause a wrong result at the output of any block, the error will be detected within one clock cycle.

4.4.3.4 Permanent Fault Detection

The AES structure of Section 4.4.2 also allows us to use one block of the chain to compute a pre-determined plaintext or ciphertext. The encryption (or decryption) of a chosen input (*e.g.* the all-zero input Z) is pre-computed once for all and hardwired (let $W = \text{AES}(Z)$ denote this value). While the system processes the actual input through one block (out of four) during any given clock cycle, another block is dedicated to recompute W. One clock after the actual C emerges, AES(Z) can be compared to the hardwired reference value W. If $W \neq \text{AES}(Z)$, a transient or a permanent fault occurred.

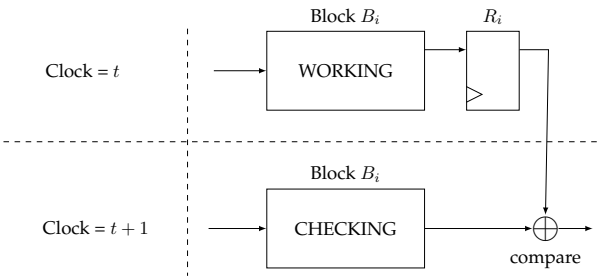


Figure 4.10: Transient fault detection scheme for AES.

In this scenario, the system starts by computing AES(Z) in the first clock cycle, followed by the actual computation of C. This allows the implementation to check up all the blocks during the execution and make sure that no permanent fault occurred. In the last clock cycle, while C is being processed in the last block, the correctness of AES(Z) is compared with the hardwired value before outputting C.

In Fig. 4.11, the red arrows represent data flow through the transformation blocks. After the initial clock cycle, the first block starts computing C. The WORKING blocks represent the calculation of C. The CHECKING blocks represent the calculation of AES(Z).

While AES(Z) will be calculated in $4N_r + 1$ clock cycles, C will be calculated in $4N_r + 2$ cycles. If the fault needs to be caught earlier, the solution described in [BBK+03] can be adapted. Yet another option consists in comparing intermediate Z encryption results (i.e., intermediate *state* values) to hardwired ones. Note that our design differs from [BBK+03] where a the decryption block is used for checking the encryption's correctness [BBK+03].

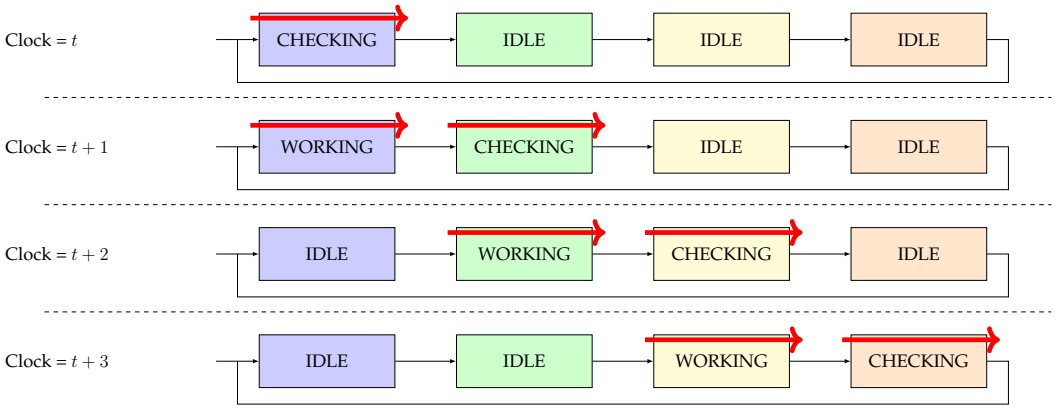


Figure 4.11: Permanent fault detection scheme for AES.

4.4.3.5 Runtime Configurability

The proposed AES architecture is a 4-stage pipeline where each stage can be used independently of the others. As already noted, blocks can perform five different tasks:

— Compute a meaningful state;
— Be in idle state to save energy;
— Scramble power consumption;
— Check for transient faults by recomputing previous calculation;
— Check for permanent faults by computing a known input.

To explore all possible combinations, we proceed as follows: first, we generate all $5^4 = 625$ combinations (5 operations for 4 transformation blocks). We can consider a subset of these combinations if we

work with 4 operations only, and remember that each E entry represents two actual options (tri-state or idle). This reduces the number of combinations to $4^4 = 256$. We eliminate all configurations that are circular permutations of others, i.e., already counted configurations shifted in time. We also eliminate the meaningless configurations in which there isn't at least one block computing. All configurations having more than one permanent fault protection block at a time are removed as they don't add any extra protection. Finally, we eliminate the cases where a transient fault checking is not preceded by a computing block or by a permanent fault verification.

Table 4.1: 29 possible configurations.

	Block 1	Block 2	Block 3	Block 4
	C	C	C	C
	C	C	C	E
	C	C	C	T
	C	C	C	P
	C	C	E	E
	C	C	E	T
	C	C	E	P
	C	C	T	T
	C	C	T	P
	C	C	P	E
	C	C	P	T
	C	E	C	E
	C	E	C	T
	C	E	C	P
	C	E	E	E
	C	E	E	T
	C	E	E	P
	C	E	T	T
⋆	C	E	T	P
	C	E	P	E
⋆	C	E	P	T
	C	T	C	P
	C	T	T	T
	C	T	T	P
⋆	C	T	P	E
	C	T	P	T
	C	P	E	E
⋆	C	P	E	T
	C	P	T	T

Table 4.2: Number of configurations.

C	E	P	T	Configurations
4				1
3	1			1
1	3			1
3		1		1
3			1	1
1			3	1
2			2	1
1	1		2	1
1		2	1	1
2	2			2
1		1	2	2
2	1	1		3
1	2	1		3
1		1	2	3
2		1	1	3
1	1	1	1	4

Table 4.1 shows that the design can perform 29 different task combinations, where *C* stands for computing, *E* stands for energy (power scrambling, idleness or any combination of these two if there are more than two *Es* in the considered configuration), *T* stands for transient fault checking and *P* stands for permanent fault checking. These options can be activated *during runtime* according to the system's constraints such as power consumption or speed. If there are no specific requirements, we recommend any of the four best configurations protecting against all attacks at once. These are singled-out in Table 4.1 by a ⋆.

Table 4.2 shows the number of configurations per protection goal. Note that for a given protection goal, different configurations can be alternated between executions without any performance loss.

4.4.4 Halving the Memory Required for AES Decryption

As we have seen, it takes $4N_r + 1$ clock cycles to encrypt or decrypt an input. The first block of the chain, *AddRoundKey* XORs the *state* with the *subkey*. Therefore, the *Key Expansion* block is designed to deliver a new 32-bit *subkey* chunk at each clock cycle.

When decrypting, the AES uses *subkeys* in the reverse order, so all *subkeys* need to be expanded and stored in memory before decryption starts. For that, decryption requires a $128N_r$-bit buffer. These $128N_r$ bits are stored in a register having N_r records of 128 bit each. Nevertheless, it is possible to halve the number of records by using the following idea: let sk_{N_r} be the *subkey* required at round N_r. All *subkeys* are computed but only the last $N_r/2$ *subkeys* are stored in memory. After the first 4 clock cycles, *AddRoundKey* block uses sk_{N_r} (the first *AddRoundKey* uses the initial *key* sk_0 which we assume to be already recorded). After 4 more cycles, sk_1 is saved in the record previously occupied by sk_{N_r}. The buffer continues to be used in such a way that each previously used (i.e., read) *subkey* is replaced by a new *subkey* of rank smaller than $N_r/2$. By the time that AES decryption requires $sk_{N_r/2}$, the *subkeys* sk_1 to $sk_{N_r/2-1}$ would have already been replaced *subkeys* sk_{N_r} to $sk_{N_r/2}$.

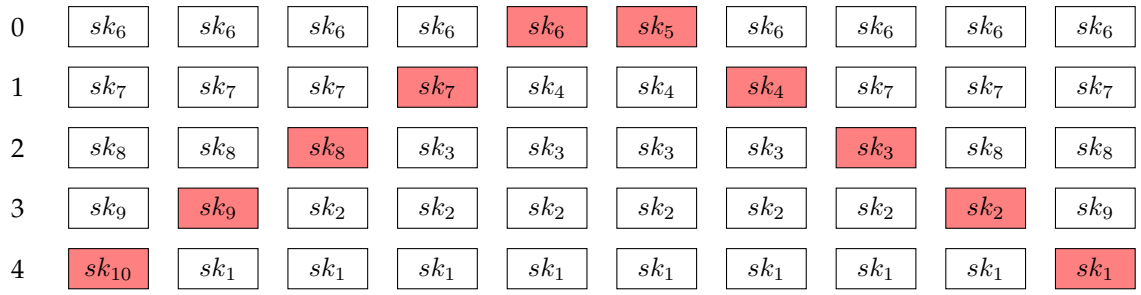


Figure 4.12: Memory halving for AES decryption when $N_r = 10$.

As shown in Fig. 4.12, only 5 records are required when $N_r = 10$. Analogously, $\{6, 7\}$ records are required for $N_r = \{12, 14\}$. The red positions are *subkeys* being used at each *AddRoundKey* operation, from left to right. Note that we assume that the initial *key* sk_0 is known and does not need to be stored.

The algorithm is formally defined as follows: Create a buffer of $N_r/2$ records denoted $r[0], \ldots, r[N_r/2 - 1]$. Place in each $r[i]$ the *subkey* $sk_{i+1+N_r/2}$.

Define the function:

$$f(i) = \frac{|2i - N_r - 1| - 1}{2}$$

When sk_i is needed, fetch it from $r[f(i)]$. After this fetch operation update the record $r[f(i)]$ by writing into it $sk_{N_r - i + 1}$.

4.4.5 Implementation Results

A 128-bit datapath AES encryption core was coded and tested in Verilog and compiled using Cadence *irun* tool. Cadence *RTL Compiler* was used to map the design into a 45nm *FreePDK* open cell digital library. Fig. 4.13 represents the inputs and outputs of the AES core. The module contains a general clock signal called *CLOCK_IN*, an asynchronous low-edge reset called *RESET_IN* and a *READY_IN* signal that flags the beginning of a new encryption. Plaintext is fed into the device *via* the 128-bit bus *TEXT_IN*, while the 128-bit key is fed to the system through the input called *KEY_IN*. The module outputs two signals: *TEXT_OUT*, which contains the resulting plaintext and *READY_OUT*, that represents a valid output.

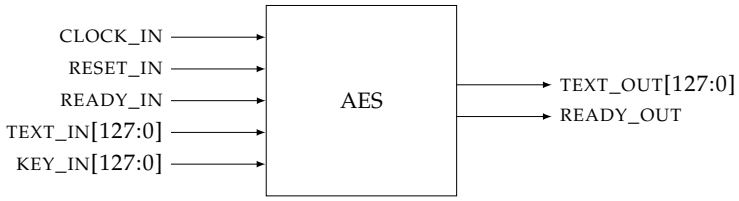


Figure 4.13: AES design's inputs and outputs.

Table 4.3 compares an unprotected AES core to the countermeasures described in this paper. The increase in terms of area is $\sim 6\%$ for the LFSR implementation and $\sim 4\%$ for the tri-state design. The LFSR implementation showed almost no increase in terms of power consumption. Since tri-state buffers shut down three out of four blocks per clock, we expect a reduction in the power consumption. The tri-state design saves roughly 20% of power compared to the unprotected AES. As tri-state buffers tend to be slower, this design lost 20% in terms of clock frequency and throughput, while the LFSR version showed no speed loss, as expected.

Table 4.4 shows the three designs benchmarks in FPGA. They were coded in Verilog and synthesized to the Spartan3E-500 board using the Xilinx ISE 14.7 tool. LFSR and tri-state designs showed an area

Table 4.3: Unprotected AES, LFSR and tri-state buffer designs synthesized to the 45nm *FreePDK* open-cell library.

	Unprotected	**LFSR**	**Tri-state**
Area ($\mu m2$)	61,581	65,194	64,243
Number of cells	10,643	11,035	11,162
sequential	783	911	787
inverters	1,483	1,614	1,493
logic	8,375	8,506	8,368
buffers	2	4	2
tri-state buffers	0	0	512
Total power (mW)	2.10	2.16	1.68
leakage power	1.20	1.28	1.26
dynamic power	0.89	0.87	0.41
Timing (ps)	645	645	806
Frequency (GHz)	1.55	1.55	1.24
Throughput (Gbit/s)	4.84	4.84	3.87

overhead of $\sim 15\%$ compared to the unprotected AES implementation. In terms of performance, LFSR design showed no loss, while the tri-state core lost $\sim 7\%$.

Table 4.4: Spartan3E-500 utilization summary report.

	Unprotected	**LFSR**	**Tri-state**
Number of Occupied Slices	1,994	2,290	2,296
Number of Flip Flops	1,142	1,270	1,146
Number of LUTs	3,521	4,106	4,031
Timing (ns)	10.789	10.714	11.580
Frequency (MHz)	92.68	93.33	86.35
Throughput (Mbit/s)	289.3	291.3	269.6

4.5 Cryptographically Secure On-Chip Firewalling

4.5.1 Introduction

The emergence of on-chip-fabric solutions is a natural consequence of Moore's Law. Increasing integration naturally produces greater complexity and hierarchy, and SoCs have been at the center of this transformation. The evolution of a set of tools that manage the complexity at the point of integration for these complex IP blocks has been centered around the on-chip fabric and are now generally referred to as a NoC or Network-on-Chip. An example of a NoC and its application in a modern SoC is illustrated in Fig. 4.14.

Multi-interface integration with different bus sizes and protocols, in addition to the challenge of floor planning on a complex SoC, have favored the incoming data packet over an on-chip network. On the other hand, bus-based interconnects are very attractive when the system has difficult timing constraints. This type of SoC architecture is less flexible but doesn't carry any extra latency due to packet conversion. More recently, the services being provided by NoCs have begun to include functions only loosely associated with the connectivity that was their genesis. For example, power management options for IP cores are beginning to be expressed by the IP core to the NoC so that the NoC hardware can provide system-level power management options to upper-level software and manage the enabling and disabling of cores.

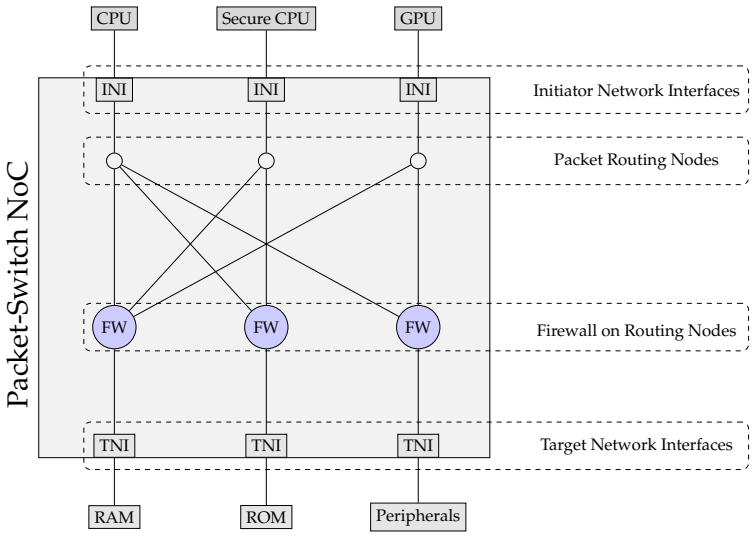


Figure 4.14: Firewalling in an SoC based on NoC interconnect.

While this function does not directly relate to connectivity, it has been inherited by the NoC since it has effectively become the high-level compiler or constraints manager for the SoC integrator. Fig. 4.15 illustrates the evolution of NoCs from simple on-chip buses to a suite of integration services for complex IP cores on an SoC.

Today, the IP cores express their connectivity requirements in the form of flows which represent the access requirements of a core or a thread. These flows are used to size and synthesize the network, communicate quality-of-service (QoS) requirements and, in a limited way, express the data sharing or non-sharing requirements between a core (or thread) and a resource. These very basic sharing rules are used to configure firewalls that limit an initiator access to a network, a resource, or an address range.

With the introduction of flows that carry special "trusted" status, these firewalls are also being used to create a very rudimentary security barrier between trusted and untrusted flows. This is the beginning of significant services that could be offered — a heterogeneous multi-core data security integration service.

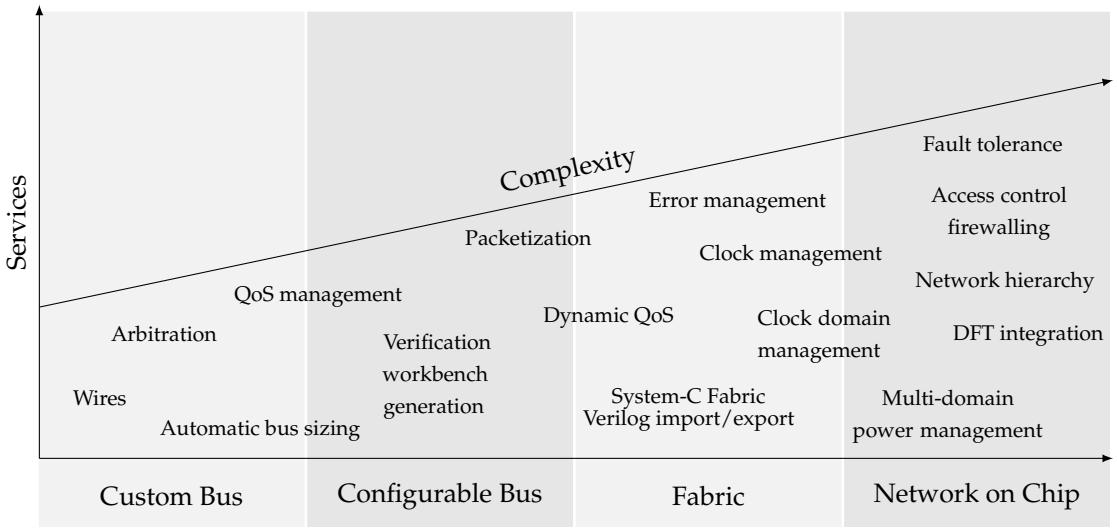


Figure 4.15: Evolution of NoC integration services.

4.5.2 Identifying Attack Surfaces on NoCs

Most commons attacks on NoCs are related to QoS considerations and have been well described in previous works [DEV+07,SPG+12], but some are targeted at data theft. NoC attacks can be classified in three main classes:

— Hijacking: writing to restricted addresses in order to change the system's configuration;
— Extraction of secret information: reading from secure addresses in order to retrieve sensitive data; and
— Denial of Service: reducing the system's throughput by replaying or forging request over the NoC.

These attacks are well understood and can be addressed with QoS and firewalling solutions [SPG+12]. However, the firewall is only as secure as the hardware and software that implement, enforce and maintain the region control. With SoC firewalling, we can identify three domains where an attack can take place, as illustrated on Fig. 4.16.

4.5.2.1 Request Path

If an attacker can successfully glitch a packet header or impersonate an initiator (by changing the initiator ID of a packet request to the firewall, for example), the attacker will be able to read from or write to a protected memory area. Although potentially dangerous, this type of attack is a single-event exploit and is rather limited. Typically, the request path and the response path are two physically different buses in the interconnect, so the packet response will be routed to the initiator that was impersonated, instead of being routed to the malicious initiator. Emerging solutions to maintain the reliability and integrity of the request path will also make successful modification of an in-flight request harder.

4.5.2.2 Firewall Reprogramming Path

However, escalating privileges in the access control block or firewalls presents a much higher threat. For instance, this access can be achieved by impersonating the reprogramming agent or glitching the bus during reprogramming to load unintended rules. If an attacker can modify access policies for a given resource, they can gain permanent access and therefore read or modify content at will, such as digital rights managements (DRM) content, banking or personal information. Recent work demonstrated how a malicious SoC hardware IP could compromise critical data [LG14], and how Internet of Things (IoT) devices could not be trusted with secure applications if firewalling and partitioning is not maintained properly.

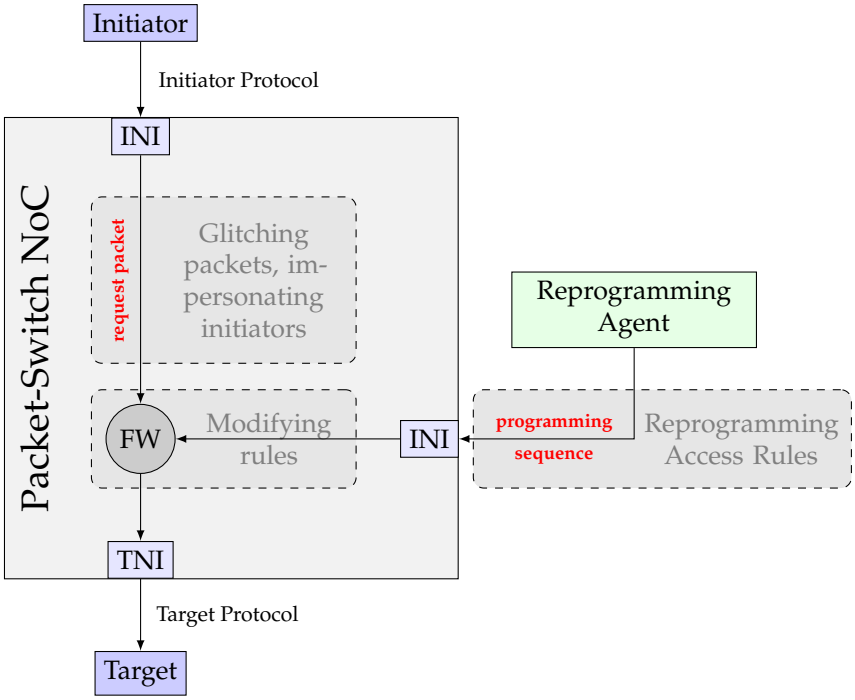


Figure 4.16: Different attack surfaces on a NoC firewall.

4.5.2.3 Firewall State at Rest

The firewall stores information in local registers that represent the access control rules for the current applications and the security policy of the SoC. This data is typically stored in a set of registers forming a look-up table that checks every access. Techniques to modify previously stored data at rest are well known [BECN+04,SCG+03] and if undetected can easily invalidate a security rule. Once a rule is invalid or mapped to another region through fault injection, the data that it is intended to protect can be modified or read by any software in the system.

4.5.3 Integration of Security Resources into an SoC

Before integrating security resources, the hardware architecture and the purpose for securing it must be analyzed to define which approach is appropriate to take. On [SCG+03], a secure processor connected to an untrusted off-chip memory requires that memory content data has not been tampered with or accessed by an unauthorized entity. This approach relies on data integrity verification and encryption and focuses on interconnect communication where multiple IP cores, secure and insecure ones, share the same resources.

Typically, information security has been studied in the context of communication systems [KLMR04]. In such a scenario, there are two entities willing to communicate over a public communication channel that is prone to attacks. Ideally, this channel should have three characteristics: *data confidentiality* (protecting sensitive information from eavesdropping), *data integrity* (ensuring that information has not been modified) and *peer authentication* (verifying that both parties are legitimate). By applying these concepts to a NoC, data confidentiality can be seen as protecting memory areas that contain private information with the use of a NoC firewall. In this domain, data integrity ensures the access rules are not modified during programming or at rest, while peer authentication allows the firewall to receive programming sequences from an authentic source.

4.5.3.1 Securing the Request Path

In an ideal world, a NoC should support a full point-to-point authenticated encryption between an initiator and a target. In addition to bringing security through a ciphered communication, this solution also offers data integrity and authenticity with the addition of an authentication tag. Unfortunately, this is problematic to implement efficiently within a NoC or a crossbar network. In fact, IP integrators usually cannot afford the impact on latency and overhead that this solution imposes. Lightweight hardware point-to-point encryption and address scrambling solutions exist [EKY11], but add too much latency to be generalized to any application on an SoC. Regarding authenticated encryption, the standard defined by NIST, the AES-CCB [Nat04], also brings too much latency overhead for NoC integration.

4.5.3.2 Securing the Firewall

Firewalling is the easiest solution for implementing on-chip access control. The simplest version of a firewall is a hardcoded look-up table that matches initiator IDs and target addresses with access rights. However, access rights require both security and flexibility, which can only be achieved with a large degree of programmability of the access control rules. Commercial access control IPs exist on the market [ARM13, Art14] and complex SoCs use firewalling to create on-chip access control [OMA14]. This offers access control over a given address space with reprogrammability. Unfortunately, firewall-based SoCs do not secure the programming sequence nor authenticate the entity responsible for reprogramming the firewall rules. Moreover, firewall-based SoCs do not maintain the integrity of the access rules during operation.

Digital signatures can secure the firewall programming by ensuring the authenticity of the programming entity, the region and its integrity. This approach protects a reprogramming agent from hijacking attack. By seeding the signature with a cryptographic nonce, replay attacks can also be prevented.

Integrity checking regions per access can also ensure the regions have not been tampered with, and can detect a modification fast enough to block the access on time.

A previous work presented in [SPG+12] describes an architecture using multiple security levels on a NoC, assuming that secure blocks are capable of defining dynamically a new set of rules to different scenarios. Unfortunately, this approach relies on hardware features customized in the NoC, making this solution less flexible.

4.5.4 Access Control Firewalling to On-Chip Resources

Access policies are usually implemented over an address space, so multiple targets can be covered using contiguous global addresses. Fig. 4.17 represents an initiator view of a partitioned address space of two targets: ROM and RAM. Each initiator has a different set of permissions for the address mapping according to its privileges. Once loaded in registers, an access rule is enforced using combinatorial logic checking of each transaction. The firewall's primary goal is to only allow transactions from initiators that have correct access rights to a given target.

2MB ROM		2GB RAM		
Region 1 SECURE	Region 2 NON-SECURE	Region 3 SECURE	Region 4 NON-SECURE	Default Region NO POLICY

Figure 4.17: Simple firewall partitioning of an address space covering two targets.

4.5.4.1 Endpoint versus NoC Firewalling

Firewalling at the NoC level presents one major advantage by having the lookup latency hidden by the packet conversion timing. In fact, the validity of the access can be checked while the packet is being converted to the target's protocol, thus access can be granted where the request exits the NoC.

An endpoint firewall is a timing-critical block that lies at the end of the request datapath (see Fig. 4.18), however, the techniques for securing the access rules are equally relevant and can be applied at the endpoint as much as at the ingress of the network.

Moreover, placing the firewall at the ingress or egress of the NoC requires synchronization between the master protocol and the slave protocol. This is usually achieved by registering inputs and outputs [CCGD12]. Since mobile applications and IoT devices use NoC for integration, one can conclude that embedding firewalls at the NoC node is the most efficient solution.

4.5.4.2 Cryptographically Secure Access Control

Cryptographically Secure Access Control (CSAC) is a security layer over the existing hardware management of virtualization of secure/non-secure environments.

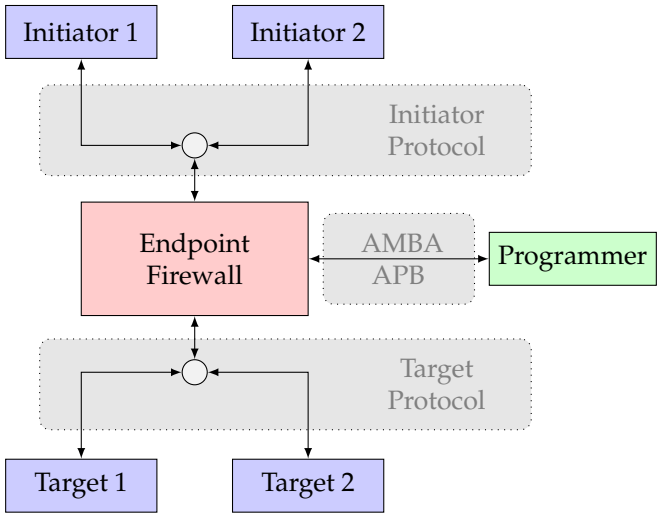


Figure 4.18: Endpoint firewall controlling access from initiators to a targets.

CSAC implements two security features. First, it cryptographically authenticates the reprogramming agent and ensures the integrity of its reprogramming sequences. Second, CSAC ensures the integrity of the access policies over the SoC by checking and hashing rules per access. The CSAC engine is based on a Keyed-Hash Message Authentication Code (HMAC) [Nat08], where the key is shared between the reprogramming agent and CSAC core. This key is programmed for each session and the programming is part of the hardware root-of-trust of the SoC. CSAC supports both hardware and software key delivery, which are performed at secure boot of the SoC.

CSAC Firewall Custom Regions. A region can be as complex as a system needs it to be. It contains multiple fields, from encoded or decoded initiator access rights, the address space covered by the region from base address to top address range or decoded sub-regions, and specific user bits that can carry data as integrity checks value and other tags. With this flexibility and the scalability of each field width, the end-user can tailor a CSAC firewall to a specific system use.

CSAC regions can scale up to 128 bits. Fig. 4.19 illustrates an example implementation of a region register with the following fields:

— Disable: disable or enable flag. Encoded on two bits.

— NSR, NSW: decoded access rights per initiator for non-secure read and non-secure write permissions. This field can scale up to 16 bits depending on the number of initiators connected to the CSAC.

— SR, SW: grouped access right for all initiators with secure read and secure write for that region. Encoded on two bits.

— BA, TA: base address and top address of the given region. These addresses are expressed in 4kB pages. This field can scale up to 36 bits according to the address space covered by the CSAC.
— Parity: parity bits for the different fields. These bits are continuously checked.
— CRC: CRC12 digest of all regions. This value is checked periodically for directed fault detection.

Each field is scalable as RTL parameters up to the value given in Fig. 4.19. This enables the user to change or scale region complexity according to the SoC needs.

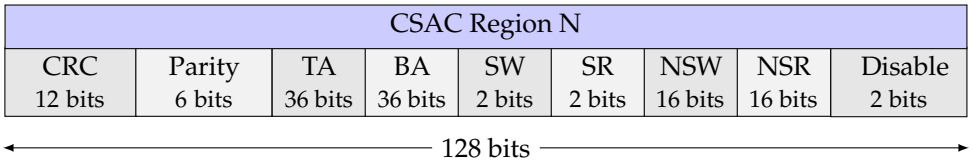


Figure 4.19: Content of a complex CSAC region.

When a request arrives to the CSAC, a wrapper decodes the incoming protocol to present to the CSAC all the fields required for a region lookup. Depending on the interconnect technology, if an illegal access is detected, the transaction can be blocked at the synchronization node for bus-based interconnect, or at the target socket in the case of a packet switch network. Independently of the illegal and blocked access, an IRQ can be raised to the processor. Some level of reconfigurability regarding IRQ policy is given to the user by programming dedicated registers.

CSAC Signing Engine. The CSAC signing engine is based on a Keyed-Hash Message Authentication Code (HMAC) using a cryptographic hash function h. Let K be the session key and PR_i the i^{th} incoming programming sequence. The HMAC tag of PR_i is given by

$$\text{HMAC}(\text{PR}_i) = h((K \oplus \text{opad}) \,||\, h((K \oplus \text{ipad}) \,||\, \text{PR}_i))$$

A firewall region is transmitted together with its signature. Both the region and the signature are 128 bits in size, corresponding to 256 bits of total data that correspond to a reprogramming sequence, which is composed of eight consecutive APB transactions written to the address of a region.

The reprogramming sequence is broken into 32-bit chunks each, to respect the size of the APB data bus. First, the most significant bits of the region are transmitted, in order, until the last chunk contains the least significant bits. After, the region signature follows the same order, from MSB to LSB. The APB address provides the region number to the CSAC, which is also used in the signature computation.

Protection Against Replay Attacks. To ensure that the CSAC is resistant against replay attacks, the region (access rule) signature is seeded with a cryptographic nonce. We call this nonce the *state variable* (SV) as it is a tracker of the session's history. At each new valid programming sequence PR_i, the CSAC updates the state variable SV_i with part of the discarded HMAC output. We denote by f the function taking part of the discarded bits of the truncated HMAC output, as we have $\text{SV}_i = f(\text{PR}_i, \text{SV}_{i-1})$. This tag becomes a function of all the preceding operations that the CSAC has performed and cannot be computed without this knowledge.

$$\text{SV}_i = f(\text{PR}_i, f(\text{PR}_{i-1}, \text{SV}_{i-2})) = f(\text{PR}_i, f(...f(\text{PR}_1, f(\text{PR}_0, 0))...))$$

Since the IP does not contain non-volatile memory, the state variable is reset at boot and initialized with the session key programming. It is important to note that the register destination address is included in the hash computation to avoid any unintended modification on the address bus.

Key Management Policy. CSAC security relies on the use of a key for authentication. This key is a secret shared with the authenticated master that reprograms the CSAC during operation. As this IP is intended for integration in large SoCs, the use of non-volatile memory for storing a personalized key should be avoided due to the added cost to the chip. Two options are possible for delivering the session key:

— The session key can be loaded through software. This solution uses a master key hardcoded in the design, the netlist key, and a session key programmed at secure boot within the root of trust with signed code before non-secure OS and applications are loaded. In this case, there is no personalization of the netlist key among devices as the security relies on a secure boot scheme that will ensure only the trusted code load the session key. The key integrity value is computed first for future key integrity checks, then the netlist key is checked for integrity, and then the session key is authenticated and programmed to the key registers.

— The session key can be loaded through a hardware Key Management System (KMS) with the security relying on this secure block. The hardware KMS will deliver the session key to the programmer to be functional.

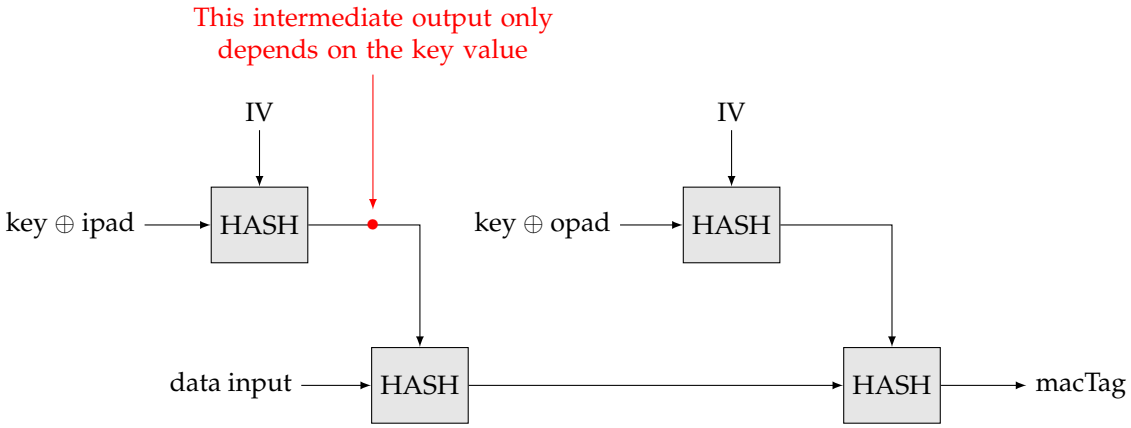


Figure 4.20: HMAC intermediate used for key integrity checks.

Before each signing operation, key integrity is checked using the embedded hash function with the first intermediate hash computation, as represented in Fig. 4.20. This operation allows key integrity checks to be performed continuously with very little overhead. By truncating the intermediate value (shown in red in Fig. 4.20) to 16 bits, the dedicated register is stored during session key programming.

Security Considerations Regarding SoC Integration. With design for test (DFT), SoC integrators and test engineers want a design that can be fully scanned and every exception heavily justified. On the other hand, security designers are concerned with back-end integration of scan flops on their designs, especially where secure registers containing a secret key are concerned. CSAC does not use one-time programmable (OTP) memory for programming keys but rather registers, so they can be easily integrated in any CMOS semiconductor process. Although CSAC allows full scanning ability over the key registers, it enforces a reset-on-scan methodology.

The purpose of this block is to clear the key register when entering test mode, and restore the netlist key into the key register when exiting test mode. It also ensures that the internal reset is not asserted during test mode, allowing for the control of the content of sensitive registers when entering and exiting a scan. Depending on the chosen design, the reset-on-scan block might contain few intentional latches.

CSAC Integrity and Overflow Check. The reference CSAC protects regions on a 4kB page basis, but also ensures that burst accesses do not cross the 4kB page boundary to avoid unintended or malicious access to regions with different access rights. The CSAC internal registers are protected against faults. In addition to parity check every access, CSAC offers integrity checking of regions using a 12-bit CRC function. The integrity check of a region is performed in four clock cycles and the design can be pipelined.

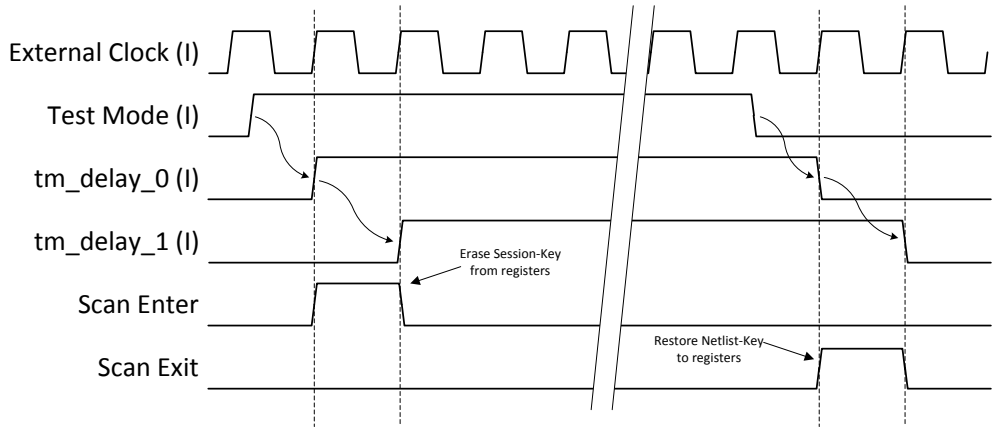


Figure 4.21: Timing diagram of reset-on-scan block.

4.5.4.3 CSAC Synthesis Results

In this scenario, CSAC was synthesized in five different versions using a digital library with technology node of $45nm$. The synthesis results were obtained with Cadence Encounter RTL Compiler v13.10. The results show the impact of including security features to the CSAC core. Each version was split into two firewall settings: one composed of four regions that provides access rights to six initiators and address a space of 4GB (Table 4.5); the other enabling the use of 14 initiators and covering a target address space of 64GB, with a total of 8 protection regions (Table 4.6). These results focus on comparing the cost of the enforcement logic and authentication engine.

The base design (a) represents a complete implementation of a firewall with no authentication engine or integrity logic (i.e., parity bits or CRC integrity checking). Design (b) enforces the programming correctness by sending parity bits embedded in the sequence, that will be constantly checked whenever a firewall access request is processed. Aside from parity bits, design (c) uses a CRC engine that periodically checks the integrity of the protection regions. Design (d) does not implement any enforcement logic in the region protection bits but authenticates each programming sequence using the HMAC mode in conjunction with the SHA-256 cryptographic hash. The last design, (e), implements all features in the same core.

Table 4.5: Synthesis results of five CSAC designs (4 regions, 6 initiators, 4GB of address space) on a $45nm$ technology node.

Design	Cells Instances	Gate Equivalent (GE)	Worst timing path (ps)	Average Power (mW)
Basic firewall (a)	746	4461	5020	0.32
Firewall + parity (b)	827	4760	5443	0.33
Firewall + parity and CRC (c)	1729	5967	5467	0.46
Firewall + SHA-256 HMAC (d)	5933	23156	10000	2.7
Firewall + all features (e)	6452	24677	10000	2.82

Tables 4.5 and 4.6 show physical synthesis results at a frequency of 100MHz. The gate equivalent (GE) metric was calculated by dividing the total cell area of each design by the area of the smallest 2-input NAND gate of the target digital library. In addition to the inherent performance loss, the security features impact area occupation and power consumption. Power consumption was estimated by the synthesis tool and represents the total power dissipation (dynamic and leakage).

The CSAC core was designed for only one clock domain. This means that the programming interface needs to work at the same speed as the firewall logic, which can be restrictive in some cases.

By using two clock domains — one for configuration, the other for firewalling — CSAC can speed up the firewall logic and avoid NoC clock frequency loss due to security. For example, it can be seen from Tables 4.5 and 4.6 that the HMAC SHA-256 engine decreases performance in designs (d) and (e) by approximately 27%. Since the authentication engine is only used when a new protection region is (re)programmed in CSAC, it does not impact the firewall enforcement logic in a two-clock domain scheme. This technique showcases that the authentication engine has a significant cost to performance, area and power consumption. In contrast, parity bits and CRC integrity checking relate to both domains as they are enforced when a programming sequence is sent to the core as well as when a firewall access request arrives to CSAC. As a result, these two countermeasures impact the overall performance.

Table 4.6: Synthesis results of five CSAC designs (8 regions, 14 initiators, 64GB of address space) on a $45nm$ technology node.

Design	Cell Instances	Gate Equivalent (GE)	Worst timing path (ps)	Average Power (mW)
Basic firewall (a)	2203	9031	7249	0.62
Firewall + parity (b)	2471	10082	8035	0.63
Firewall + parity and CRC (c)	3026	10836	7734	0.73
Firewall + SHA-256 HMAC (d)	6900	27555	10000	2.79
Firewall + all features (e)	7929	30343	10000	3.17

4.5.4.4 FPGA Implementation

This solution was implemented on a Xilinx ZINQ-700 development board embedding an ARM Cortex A9 MCU core and an FPGA. In this example, traffic scenarios were created on the programmable logic using a simple packet-based NoC connecting two initiators, the ARM core, and a DMA to a DDR memory. On the NoC node, the CSAC IP core was instantiated to securely partition the 1GB memory. To split the accesses on secure and non-secure initiators, the PROT[1] signal present on AMBA AXI protocol was used. As well, CSAC region registers were programmed in a C++ program that we compiled on an ARM instruction set.

Table 4.7: CSAC (4 regions, 6 initiators, 4GB of address space) synthesis on Zynq-7000 board.

Design	Slice LUTs (logic)	Slice Registers (flip flop)	Muxes
Basic firewall (a)	417	353	38
Firewall + parity (b)	441	437	47
Firewall + parity and CRC (c)	1103	772	47
Firewall + SHA-256 HMAC (d)	2429	1973	61
Firewall + all features (e)	3095	2350	77

A simple interrupt handler was created to prevent the ARM core from hanging and to alert CSAC software in case of any CSAC hardware status modification. This approach prevents the use of software polling. With this configuration, the illegal requests were blocked per access and the correctness of signature checks was verified. Fault tolerance is achieved by design and verified with assertion during verification of the IP and can also be evaluated in emulation with a laser fault injection bench.

Table 4.8: CSAC (8 regions, 14 initiators, 64GB of address space) synthesis on Zynq-7000 board.

Design	Slice LUTs (logic)	Slice Registers (flip flop)	Muxes
Basic firewall (a)	751	823	147
Firewall + parity (b)	810	851	86
Firewall + parity and CRC (c)	1689	1244	181
Firewall + SHA-256 HMAC (d)	2735	2435	201
Firewall + all features (e)	3688	2825	169

4.6 Practical Instantaneous Frequency Analysis Experiments

4.6.1 Introduction

The physical interpretation of data processing (a discipline named the *physics of computational systems* [MC80]) draws fundamental comparisons between computing technologies and provides physical lower bounds on the area, time and energy required for computation [Ben73,Key75]. In this framework, a corollary of the second law of thermodynamics states that in order to perform a transition between states, energy must be lost irreversibly. A system that conserves energy cannot make a transition to a definite state and thus cannot make a decision (compute) ([MC80],9.5).

At any given point in the evolution of a technology, the smallest logic devices must have a definite physical extent, require a certain minimum time to perform their function and dissipate a minimal switching energy when transiting from one state to another.

Because CMOS state transition energy is essentially proportional to the number of switched bits, transition energy leakage is the most popular side-channel attack vector. Because commuting also requires time, transition time and processed data might be also related.

Historically, timing attacks were developed to extract secrets from software algorithms [Koc96] while hardware algorithms were usually assumed to run in constant time and hence be immune to timing attacks. The constant hardware execution time assumption is supported by the fact that usual block-cipher hardware implementations require an identical number of clock cycles to process any data. This article shows that this intuition is not always true, i.e., two different inputs may require distinct processing time and can hence be distinguishable.

Energy consumed during each clock cycle creates a waveform in the power domain. A duty cycle, i.e., the time during which the power wave is not equal to its nominal value, can be considered as the execution time of a hardware implemented algorithm. As shown later the duty cycle may depend on the processed data. Fourier transform can not determine local duty cycles since frequency is defined for the sine or cosine function spanning the whole data length with constant period and amplitude. However, recent techniques described in this paper that can detect local frequencies and hence determine wave duty cycle.

In 2005 it was observed that not only signal amplitude, but also power spectrum, can leak secret information [GHT05]. Following the introduction of Differential Frequency Analysis (DFA) [GTC05], power analysis on frequency domain was investigated on a series of papers [Luo10,MG11,PGQK09,OSBHR10]. DFA applies Fourier transform to map a time-series into the frequency domain. Since each Fourier point is a linear combination of all other sample points, a spectrum is a direct function of the initial signal amplitude and hence, power spectra can also be used in side-channel attacks.

[Luo10] rightly noted that the term Differential Spectral Based Analysis (DSBA) is semantically preferable because DFA does not exploit variations in frequencies, but *differences in spectra*. As the matter of fact all time-domain power models and distinguishers remain in principle fully applicable in the frequency domain.

Dynamic Voltage Scrambling (DVS) is a particular side-channel countermeasure that triggers random power supply changes aiming to decorrelate the signal's amplitude from the processed data [BZ07, KGS+11]. While DVS degrades DPA's and DSBA's performances, nothing prevents the existence of more subtle side-channel attacks exploiting DVS-resistant die-hard information present in the signal. This paper successfully exhibits and exploits such DVS-resistant information.

Organization. We show that, in addition to the signal's amplitude and spectrum, traditionally used for side-channel analysis, instantaneous frequency variations may also leak secret data. To the authors' best knowledge, "pure" frequency leakage has not been considered as a side-channel vector so far. Hence a re-assessment of several countermeasures, especially, these based on amplitude alterations, seems in order. As an example this paper examines DVS, which makes AES implementation impervious to power and spectrum attacks while leaving it vulnerable to Correlation Instantaneous Frequency Analysis (CIFA), a new attack described in the following sections. Section 4.6.2 turns a signal processing algorithm called *Hilbert Huang Transform* (HHT) into an attack process. Section 4.6.3 illustrates an

HHT performed on a real power signal and motivates the exploration of instantaneous frequency as a side-channel carrier. Section 4.6.4 compares the cryptanalytic effectiveness of Correlation Instantaneous Frequency Analysis, Correlation Power Analysis and Correlation Spectrum Based Analysis on an unprotected AES FPGA implementation and on AES FPGA power traces with a simulated DVS.

4.6.2 Preliminaries

The notion of *instantaneous frequency*, computable by the HHT, was introduced in [HSL+98]. During the last decade, HHT has found many practical applications including oceanographic exploration and medical research [HS05]. This section recalls HHT's main mathematical features and describes the hardware setup used for evaluating the attacks introduced in this paper.

4.6.2.1 The Hilbert Huang Transform

The HHT represents the analyzed signal in the time-frequency domain by combining the *Empirical Mode Decomposition* (EMD) with the *Discrete Hilbert Transform* (DHT).

DHT is a classical linear operator that transforms a signal $u(1), \ldots, u(N)$ into a time series $H_u(1), \ldots, H_u(N)$ as follows:

$$H_u(t) = \frac{2}{\pi} \sum_{k \neq t \bmod 2} \frac{u(k)}{t-k} \tag{4.2}$$

DHT can be used to derive an *analytical representation* $u_a(1), \ldots, u_a(N)$ of the real-valued signal $u(t)$:

$$u_a(t) = u(t) + iH_u(t) \text{ for } 1 \leq t \leq N \tag{4.3}$$

Equation (4.3) can be rewritten in polar coordinates as

$$u_a(t) = a(t)e^{i\phi(t)} \tag{4.4}$$

where

$$a(t) = \sqrt{(u^2(t) + H_u^2(t))} \text{ and } \phi(t) = \arctan\left(\frac{H_u(t)}{u(t)}\right) \tag{4.5}$$

represent the *instantaneous amplitude* and the *instantaneous phase* of the analytical signal, respectively.

The *phase change rate* $w(t)$ defined in equation (4.6) can be interpreted as an *instantaneous frequency* (IF):

$$w(t) = \phi'(t) = \frac{d}{dt}\phi(t) \tag{4.6}$$

For a real-valued time-series the definition of $w(t)$ becomes:

$$w(t) = \phi(t) - \phi(t-1) \tag{4.7}$$

The derivative must be well defined since physically there can be only one instantaneous frequency value $w(t)$ at any given time t. This is insured by the *narrow band condition*: the signal's frequency must be uniform [KM10]. Further, the physical meaningfulness of DHT's output is closely related to the input's fitness into a narrow frequency band [Boa92]. However, we wish to work with non-stationary signals having more than one frequency. This is achieved by de-composing these signals into several components, called *Intrinsic Mode Functions*, such that each component has nearly the same frequency.

Definition 4.1 (Intrinsic Mode Function) *An Intrinsic Mode Function (IMF) is a function satisfying the following conditions:*

1. *the number of extrema and the number of zero crossings in the considered data set must be either equal or differ by at most one;*
2. *the mean value of the curve specified as a sum of the envelope defined by the local maxima and the envelope defined by the local minima is zero.*

First step: Empirical Mode Decomposition (EMD). EMD, the HHT's first step, is a systematic way of extracting IMFs from a signal.

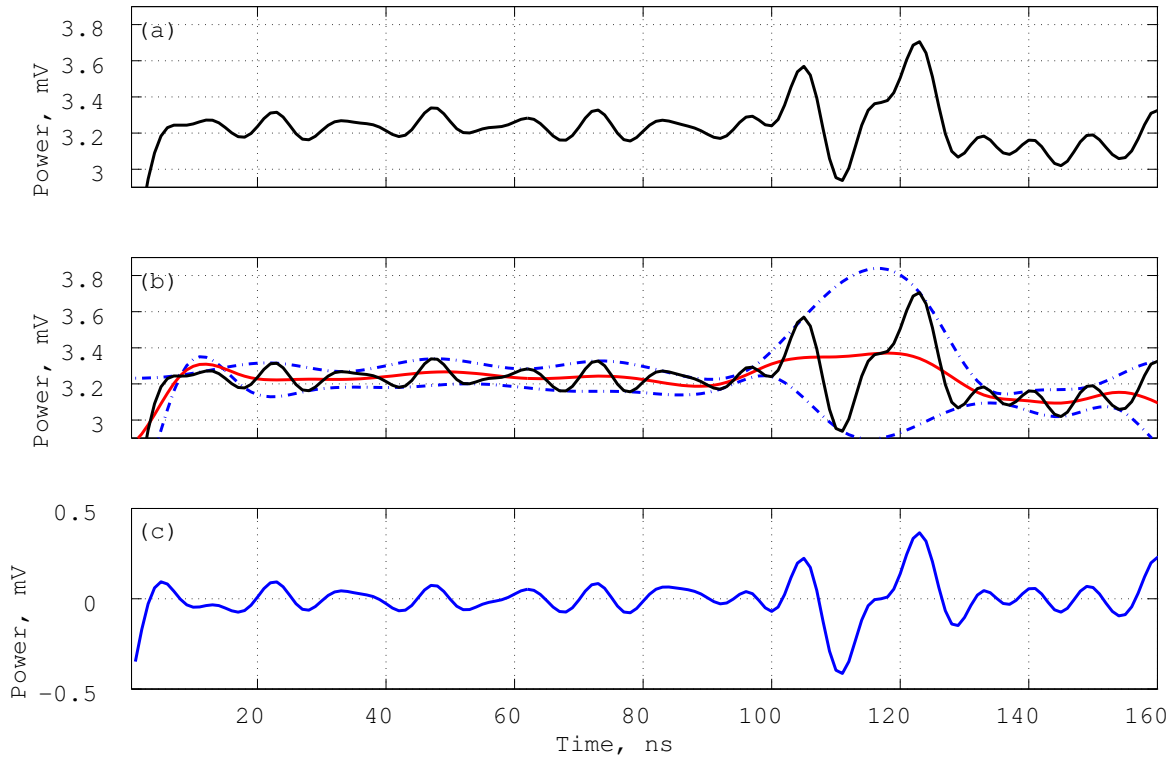


Figure 4.22: Illustration of the EMD: (a) is the original signal $u(t)$; (b) $u(t)$ in thin solid black line, upper and lower envelopes are dot-dashed with their mean $m_{i,j}$ in thick solid red line; (c) shows the difference between $u(t)$ and the envelope's mean.

EMD involves approximation with splines. By Definition 4.1, EMD uses local maxima and minima separately. All the local signal's maxima are connected by a cubic spline to define an upper envelope. The same procedure is repeated for the local minima to yield a lower envelope. The first EMD component $h_{1,0}(t)$ is obtained by subtraction from $u(t)$ the envelopes' mean $m_{1,0}(t)$ (see Fig. 4.22):

$$h_{1,0}(t) = u(t) - m_{1,0}(t) \tag{4.8}$$

Ideally, $h_{1,0}(t)$ should be an IMF. In reality this is not always the case and EMD has to be applied to $h_{1,0}(t)$ as well:

$$h_{1,1}(t) = h_{1,0}(t) - m_{1,1}(t) \tag{4.9}$$

EMD is iterated k times, until an IMF $h_{1,k}(t)$ is reached, that is

$$h_{1,k}(t) = h_{1,k-1}(t) - m_{1,k}(t) \tag{4.10}$$

Then, $h_{1,k}(t)$ is defined as the first IMF component $c_1(t)$.

$$c_1(t) \stackrel{\text{def}}{=} h_{1,k}(t) \tag{4.11}$$

Next, the IMF component $c_1(t)$ is removed from $u(t)$

$$r_1(t) = u(t) - c_1(t) \tag{4.12}$$

and the procedure is iterated on all the subsequent residues, until the residue $r_n(t)$ becomes a monotonic function from which no further IMFs can be extracted.

$$\begin{cases} r_2(t) = r_1(t) - c_2(t) \\ \dots \\ r_n(t) = r_{n-1}(t) - c_n(t) \end{cases} \tag{4.13}$$

Finally, the initial signal $u(t)$ is re-written as a sum:

$$u(t) = \sum_{j=1}^{n} c_j(t) + r_n(t), \quad \text{for} \quad 1 \leq t \leq N \tag{4.14}$$

where, $c_j(t)$ are IMFs and $r_n(t)$ is a constant or a monotonic residue.

Second step: Representation. The second HHT step is the representation of the initial signal in the time-frequency domain. All components $c_j(t)$, $j{\in}[1,n]$ obtained during the first step are transformed into analytical functions $c_j(t) + iH_{c_j}(t)$, allowing the computation of instantaneous frequencies by formula (4.7). The final transform $U(t,w)$ of $u(t)$ is:

$$U(t,w) = \sum_{j=1}^{n} a_j(t) \exp\left(i \sum_{\ell=1}^{t} w_j(\ell)\right) \tag{4.15}$$

where $j{\in}[1,n]$ is indexing components, $t{\in}[1,N]$ represents time and:

$a_j(t) = \sqrt{c_j^2(t) + H_{c_j}^2(t)}$ is the *instantaneous amplitude*;
$w_j(t) = \arctan\left(\frac{H_{c_j}(t+1)}{c_j(t+1)}\right) - \arctan\left(\frac{H_{c_j}(t)}{c_j(t)}\right)$ is the *instantaneous frequency*;

Equation (4.15) represents the amplitude and the instantaneous frequency as a function of time in a three-dimensional plot, in which amplitude can be contoured on the frequency-time plane. This frequency-time amplitude distribution is called the *Hilbert amplitude spectrum* $U(t,w)$, or simply the *Hilbert spectrum* [HSL+98]. In addition to the Hilbert spectrum, we define the *marginal spectrum* or *HHT power spectral density* $h(w)$, as

$$h(w_j) = \sum_{t=1}^{T} U(t, w_j) \tag{4.16}$$

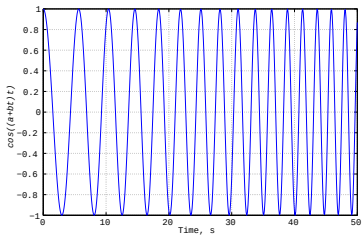


Figure 4.23: The increasing frequency function $\cos((a+bt)t)$.

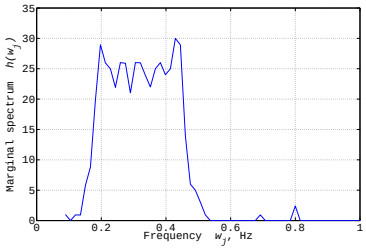


Figure 4.24: Analysis of the function $\cos((a+bt)t)$: Marginal Hilbert spectrum of Fig. 4.23.

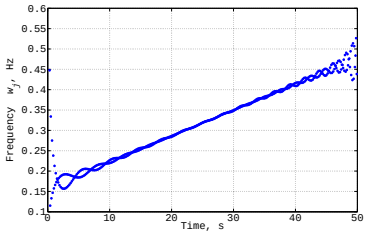


Figure 4.25: Analysis of the function $\cos((a+bt)t)$: Hilbert's amplitude spectrum contour of Fig. 4.23.

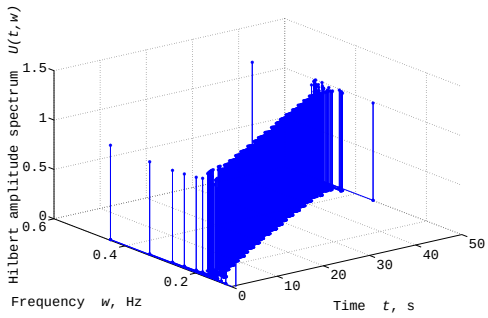


Figure 4.26: Analysis of the function $\cos((a+bt)t)$: Hilbert's amplitude spectrum contour of Fig. 4.23.

The marginal spectrum measures the total amplitude (or energy) contributed by each frequency value.

To illustrate HHT decomposition consider the function $u(t) = \cos(t\,(a+bt))$. In Fig. 4.23 parameters a and b were arbitrarily set to $a = 1$ and $b = 0.02$. Fig. 4.23 shows that the cosine's frequency increases progressively. Fig. 4.24 presents the Hilbert marginal spectrum of the signal $u(t) = \cos((1+0.02t)t)$. Fig. 4.25 shows the contour of Hilbert's amplitude spectrum, i.e., frequency evolution in time, and this evolution is indeed nearly linear. The 3D Hilbert amplitude spectrum is illustrated in Fig. 4.26.

4.6.2.2 AES Hardware Implementation

The AES-128 implementation used for our experiments runs on an Altera Cyclone II FPGA development board clocked by an external 50MHz oscillator. The AES architecture uses a 128-bit datapath. Each AES round is completed in one clock cycle and key schedule is performed during encryption. The

substitution box is described as a VHDL table mapped into combinational logic after FPGA synthesis. Encryption is triggered by a high `start` signal. After completing the rounds the device halts and drives a `done` signal high.

The implementation has no side-channel countermeasures. To simulate DVS, 200,000 physically acquired power consumption traces were processed by Algorithm 7. Algorithm 7 splits a time-series into segments and adds a uniformly distributed random voltage offset to each segment.

The rationale for simulating a DVS by processing a real signal (rather than adding a simple DVS module to the FPGA) is the desire to work with a rigorously modeled signal, free of the power consumption artifacts created by the DVS module itself.

4.6.3 Hilbert Huang Transform and Frequency Leakage

4.6.3.1 Why Should Instantaneous Frequency Variations Leak Information?

Most of the power consumed by a digital circuit is dissipated during rising or falling clock edges when registers are rewritten with new values. This activity is typically reflected in the power consumption trace as spikes occurring exactly during clock rising edges. Spike frequency, computed by the Fourier transform, is usually assumed to be constant because clock frequency is stable. In reality, this assumption is incorrect since each spike has its own duty cycle and consequently its own assortment of frequencies.

Differences in duty cycle come from the fact that the circuit's power supply must be restored to its nominal value after switching. Bigger amplitude spikes take more time to resorb than smaller amplitude ones.

To illustrate these spike differences, consider the simple circuit in Fig. 4.27. Each parallel branch has a resistor r, a switch S_i and a capacitor C that simulate a single inverter when switched from low to high. Resistor R_s and the current i_s represent the circuit's static current and R_a is the resistor used for acquisition. Initially all the switches $S_1 \dots S_k$ are open, so the current flowing through R_a is simply i_s.

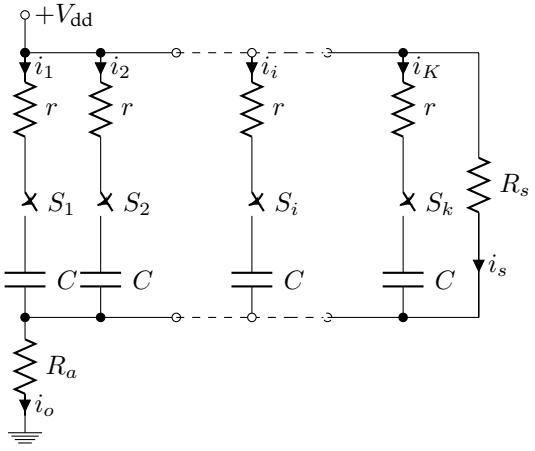


Figure 4.27: Inverters switch simulation.

Assume that at $t_0 = 0$ all the switches $S_1 \dots S_k$ are suddenly closed. All capacitors start charging and current flowing through R_a rises according to the following equation:

$$i_o(t) = i_s + k\left(\frac{V_{\text{dd}}}{r} e^{-\frac{t}{rC}}\right) \tag{4.17}$$

Equation (4.17) shows that current amplitude depends on the number of closed switches. However, there is one more parameter in the equation, namely the time t that characterizes the switching spike.

The current i_o needs some time to "practically" reach an asymptotic nominal value i_s and this time depends on the number of closed switches k. Consider the time T_k required by $i_o(t)$ to reach $\Gamma\%$ of its asymptotic value, i.e., $\frac{\Gamma}{100} i_s$:

$$i_o(T_k) = i_s - k\left(\frac{V_{\text{dd}}}{r} e^{-\frac{T_k}{rC}}\right) = \frac{\Gamma}{100} i_s \tag{4.18}$$

This is equivalent to:

$$T_k = rC \ln\left(\frac{100}{100-\Gamma}\frac{V_{\text{dd}}}{i_s r}\right) + rC\ln(k) = \alpha + \beta\ln(k) \tag{4.19}$$

Equation (4.19) shows that convergence time has a constant part α and a variable part $\beta\ln(k)$ that depends on the number of closed switches k. Equation (4.19) shows that both spike period and spike frequency depend on the processed data and could hence in principle be used as side-channel carriers. Nevertheless, power consumption is a non-stationary signal, which justifies the use of HHT.

Intuitively and dirtily, if we assimilate the curves in Fig. 4.24 to sines, we see that the instantaneous frequency $\frac{1}{T_k}$ reflects the number of closed switches k.

The dependency between the number of switches and spike period in equation (4.19) is non-linear and hard to formalize as a simple formula for a real circuit. Section 4.6.3.2 shows that the standard Hamming distance model can be used in conjunction with instantaneous frequency.

4.6.3.2 Power consumption of one AES round

The relationship between processed data and power amplitude is a well understood phenomenon [AARR03, BCO04, GBTP08, KJJ99]. However, to the best of our knowledge the dependency of instantaneous frequency on processed data has not been explored so far. This may be partially explained by the fact that Fourier Transform, previously used in some papers, is not inherently adapted to non-stationary and non-linear signals. Fourier analysis cannot extract frequency variations from a signal because frequency is defined as a constant parameter of the underlying sine function spanning the whole data-set $u(t)$. By opposition, HHT allows extracting instantaneous frequencies and exploiting them for subsequent cryptanalytic purposes.

To illustrate information leakage through frequency variation, the AES last rounds' power consumption was measured using a Picoscope 3207A with 250 MHz bandwidth at 10 G/s equivalent time sampling rate. Every signal had 1,000 samples and 100,000 traces were acquired for various input plaintexts. A power consumption example of the four last rounds is shown on the Fig. 4.28.

The AES last round was extracted from each power trace as shown on Fig. 4.29. The number of bits switches in the AES last round was computed with the known key. Afterwards the traces with the same number of bits switches were averaged.

In classic side-channel models [BCO04], flipping more bits would consume more energy. Fig. 4.29, 4.30 and 4.31 show that such is indeed the case for power consumption of 55, 65 and 75 bit flips where $v_{75} > v_{65} > v_{55}$. As per our assumption, the *frequency signatures of these three operations are also different*.

To show that HHT can detect frequency differences consider the power spectral density (PSD) of signals during 55, 65 and 75 bits switching (Fig. 4.31). The maximal spectral amplitude of the 55 bit change is located at 51.18 MHz (point f_{55}), that of the 65 bit change is at 51.12 MHz (point f_{65}) and that of the 75 bit change is at 50.73 MHz (point f_{75}) which is supportive of the hypothesis that HHT can distinguish frequency variations even in non-stationary signals because $f_{55} > f_{65} > f_{75}$.

This shows that not only amplitude but also frequency varies during register switch. Logically, power consumption increases as more bits are flipped. However, HHT was previously applied only for one

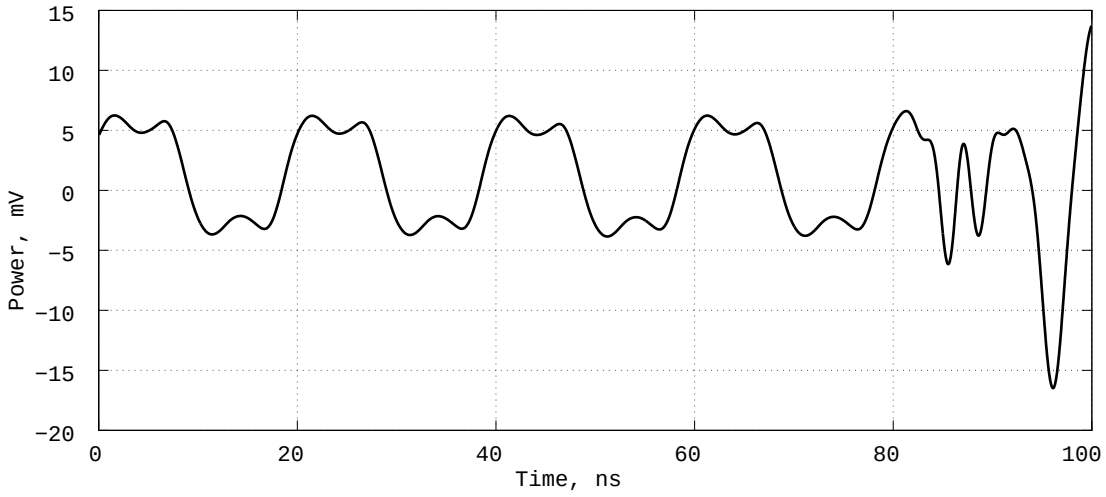


Figure 4.28: Four AES last rounds.

AES round and HHT's applicability for the entire AES power traces must be verified. That is why the next section carefully examines the effect of register alteration on IF when AES FPGA implementation is sampled at a smaller rate.

4.6.3.3 Hilbert Huang Transform of an AES Power Consumption Signal

We start by performing a Hilbert Huang decomposition of a real signal. The analysis was performed on the power trace of the previously described AES-128 implementation. The acquisition was performed 1 G/s real time rate with 1 GHz differential probe. Signals were averaged 10 times and had 1,000 samples (Fig. 4.32).

EMD decomposed the power trace to five IMFs and a residue, shown in Fig. 4.33. After decomposition, each IMF was Hilbert Transformed to derive the power signal's time-frequency representation. Fig. 4.34 is an IF distribution of Fig. 4.32.

Amplitude combination over frequency gave the power spectral density plot shown in blue on Fig. 4.35. An important observation in Fig. 4.35 is that HHT spectrum shows the distribution of a periodic variable over the main peak frequencies. Notably, the peak near 50 MHz that corresponds to the board's oscillator is not represented by a single point, but by a set of points. This data scatter can be explained by the fact that the IF of AES rounds varies, and HHT distinguishes this variation.

The main difference between HHT and FFT spectra (see plot shown in red on Fig. 4.35) is that HHT defines frequency as the speed of phase change and can hence detect intra-time-series deviations from the carrier's oscillation, whereas FFT frequency stems from the sine function, which is independent of the signals' shape.

So far, it was shown that IF varies for different rounds even within a given trace. However, an attack is only possible when IF depends on the data's Hamming weight.

The dependency is apparent in Fig. 4.36 showing the relationship between Hamming distance of the 9th and 10th AES round states and IF, taken from the first IMF component at the beginning of the 10th round. Fig. 4.36 was drawn using 200,000 HHT-processed power traces. The thin solid line in Fig. 4.36 represents the mean IF value, obtained from the first IMF component, as a function of Hamming distance.

The principal trend is the ascending line. Fig. 4.36 corresponds well to the simulation of a register's power consumption since frequency is decreasing due to the increase in Hamming distance. The relationship in Fig. 4.36 between Hamming distance and IF looks linear and therefore the Pearson correlation coefficient can be used as an SCA distinguisher.

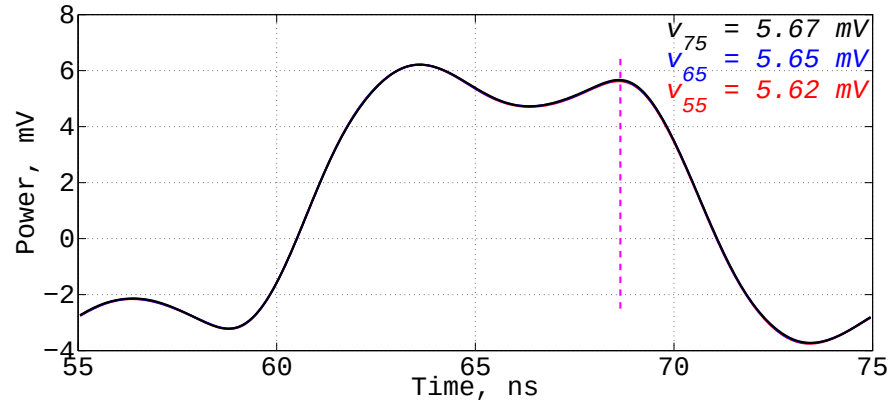


Figure 4.29: AES last round power consumption for 55 (red), 65 (blue) and 75 (black) register's flip-flops: Full voltage range.

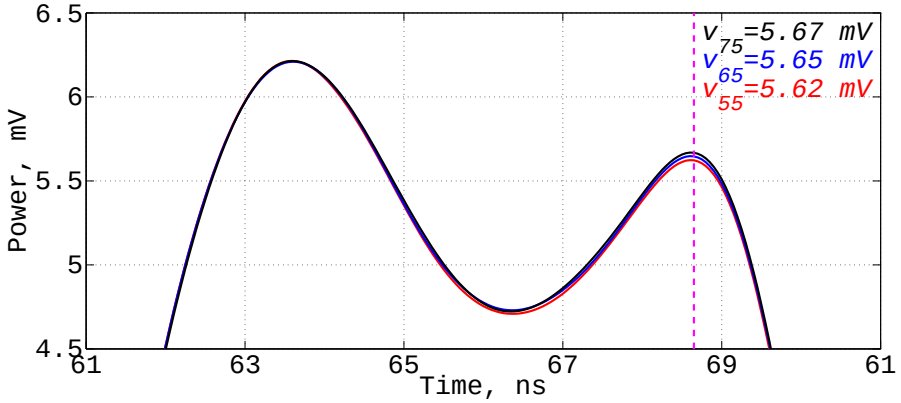


Figure 4.30: AES last round power consumption for 55 (red), 65 (blue) and 75 (black) register's flip-flops: Zoomed voltage range.

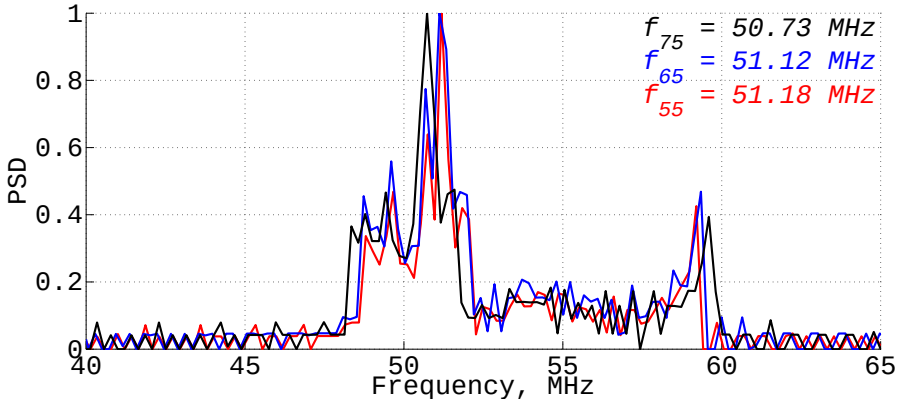


Figure 4.31: AES last round power consumption for 55 (red), 65 (blue) and 75 (black) register's flip-flops: Power spectra density for the signals shown on Fig. 4.29.

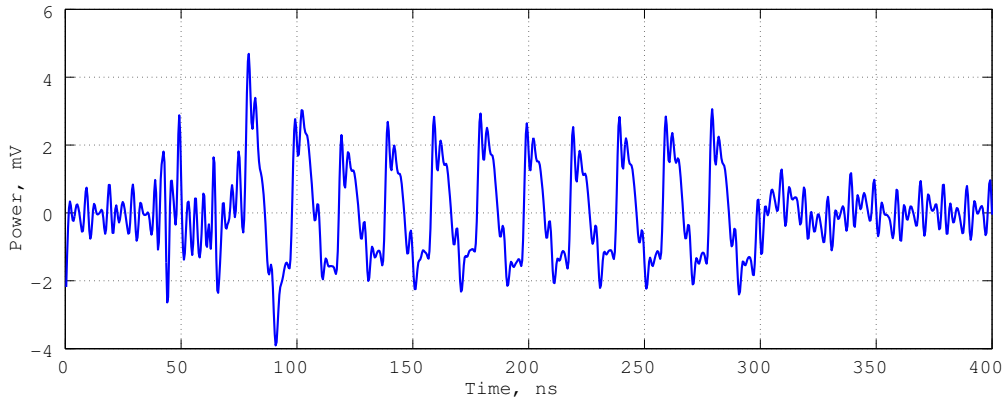


Figure 4.32: Power consumption of our experimental AES-128 implementation: initial signal $u(t)$.

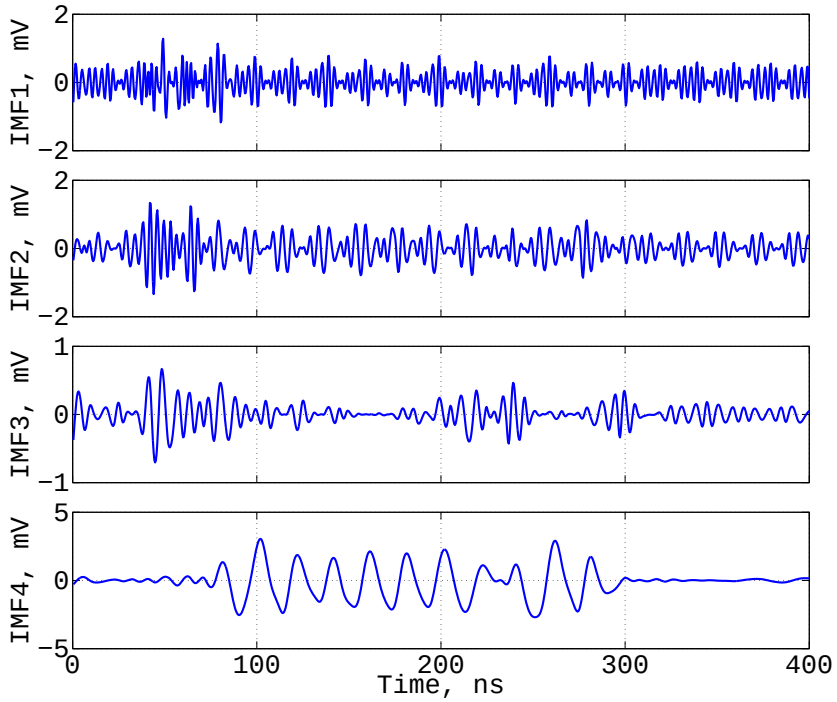


Figure 4.33: Power consumption of our experimental AES-128 implementation: The Empirical Mode Decomposition of signal $u(t)$.

IF adoption for side-channel attacks presents some particularities. The disadvantage of the method is that data scatter is higher than in usual DPA and hence the attack requires more power traces. Another issue is that each time-series will be decomposed into a set of IMFs, hence every sample will be wrapped-up with a set of IFs virtually multiplying the amount of data to be processed. However, the advantage is that because frequency based analysis is independent of local amplitude, CIFA can still be attempted in the presence of certain countermeasures.

4.6.4 Correlation Instantaneous Frequency Analysis

This section introduces Correlation Instantaneous Frequency Analysis (CIFA) and compares its performance with Correlation Power Analysis (CPA) and to Correlation Spectral Based Analysis (CSBA).

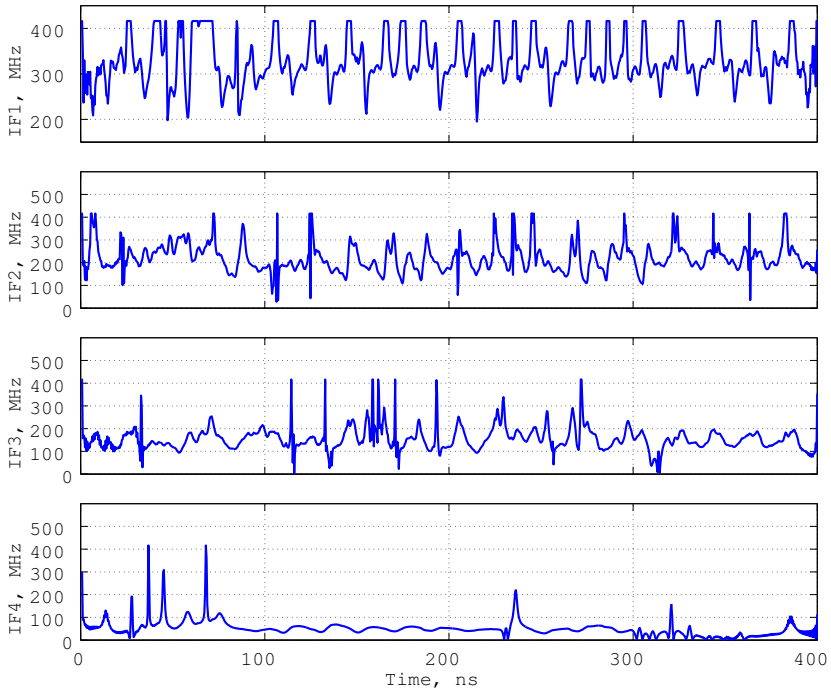


Figure 4.34: Power consumption of our experimental AES-128 implementation: IF distribution over time for the different IMFs of Fig. 4.33.

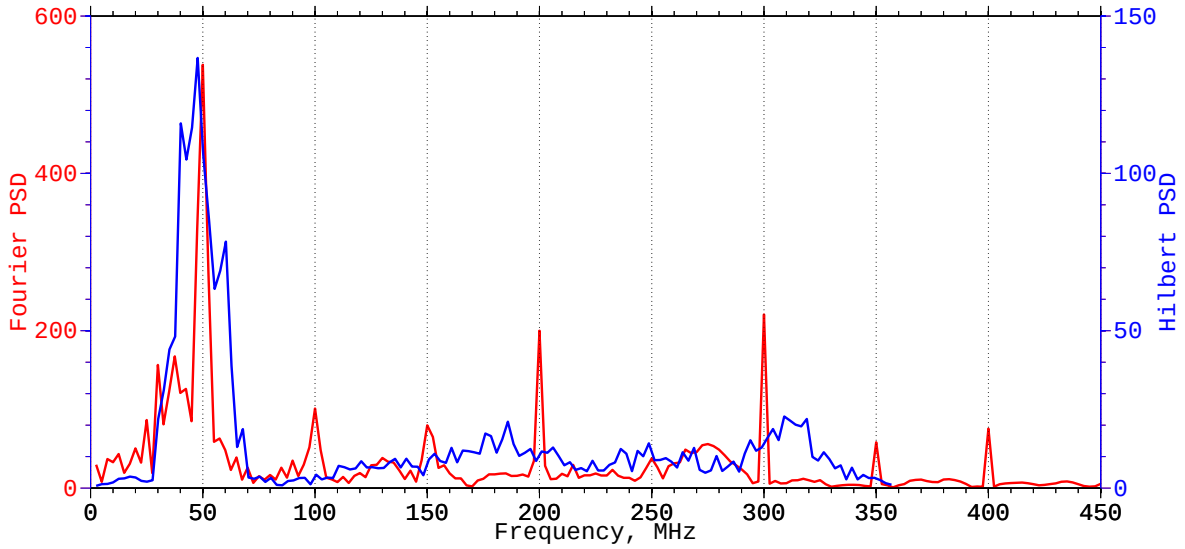


Figure 4.35: Fourier and Hilbert power spectrum density of Fig. 4.32.

4.6.4.1 Correlation Instantaneous Frequency Analysis on Unprotected Hardware

During the acquisition step 200,000 power traces were acquired at a sampling rate of 2.5 GS/s. Each power signal was averaged 10 times to reduce noise. All traces were HHT-processed using the Matlab HHT code of [BKMG07, BKMG12]. Most traces were decomposed into 6 components, but 5 and 7 IMFs occurred as well. To reduce the amount of processed information only the first four IMFs were used.

Generally, each higher rank IMF carries information present in smaller instantaneous frequencies (Fig. 4.34), this is why IMFs from different power traces were aligned index-wise, i.e., all first IMFs from every encryption were analyzed first, then all second IMFs and so on.

We chose the Hamming distance model and Pearson's correlation coefficient to investigate CIFA's prop-

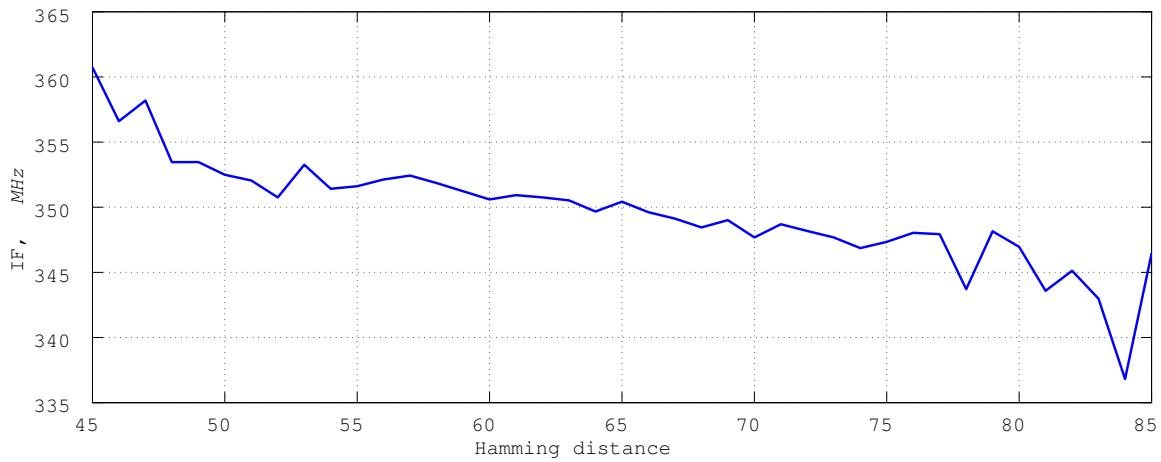


Figure 4.36: Dependency between the Hamming distance of 9th and 10th AES round states and the IF of the first IMF component at time 276 ns (corresponding to the beginning of the last AES round).

erties and compare CIFA with other attacks.

CPA. CPA applied to power traces produces Fig. 4.37(a). Clearly, CPA outperforms CIFA. CIFA's poorer performance can be partially attributed to the power model, because IF is not linearly dependent on the Hamming distance.

CSBA. Fig. 4.37(b) presents CSBA applied against Fourier power trace spectra with the same power model and distinguisher. The correct key byte can be distinguished from 2000 power traces and on.

CIFA. The application of the selected power model and of the distinguisher to IFs yields Fig. 4.37(c) where the correct key byte emerges from 16,000 power traces and on.

The three experiments seem to suggest that CSBA is superior to CIFA but inferior to CPA. That is CIFA $<$ CSBA $<$ CPA.

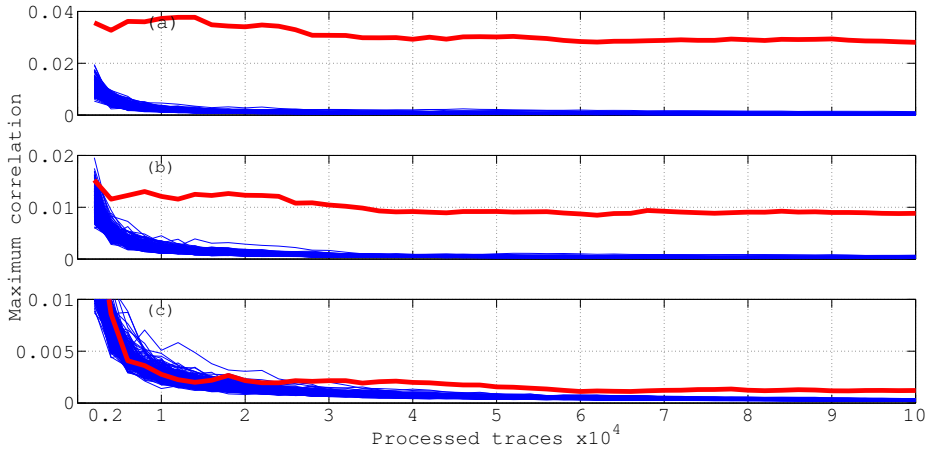


Figure 4.37: Maximum correlation coefficients for a byte of the last round AES key in an unprotected implementation. Although the three attacks eventually succeed CPA>CSBA>CIFA. (a) CPA (b) CSBA (c) CIFA.

While it appears that CPA and CSBA outperform CIFA in the absence of countermeasures, we will now see that CIFA survives countermeasures that derail CPA and CSBA.

4.6.4.2 Correlation Instantaneous Frequency Analysis in the Presence of DVS

As mentioned previously DVS alters power supply to reduce dependency between data and consumed power. According to [BZ07,KGS+11] DVS is cheap in terms of area overhead since only a voltage controller and a random number generator must be added to the protected design.

To simulate DVS all the traces of the unprotected AES were modified by Algorithm 7. Each power trace was partitioned into γ segments of normally distributed lengths covering the whole dataset. [1]. Each segment was lifted by a uniformly distributed random offset ℓ that did not exceed a predetermined value D set to $D = 12$ mV.

Algorithm 7 Dynamic Voltage Scrambling (DVS) simulator.

Require:
A power trace $u(1), \ldots, u(N)$;
γ : the number of segments;
m : mean value of segment length $m \stackrel{\text{def}}{=} N/\gamma$;
σ : standard deviation of segment length;
D : maximum offset for segment lifting;
Ensure:
a DVS-protected power trace $u'(1), \ldots, u'(N)$;

$\tau_0 \leftarrow 1$
$\tau_\gamma \leftarrow N$
for $i = 1$ **to** $\gamma - 1$ **do**
  $\tau_i \leftarrow \tau_{i-1} + \mathcal{N}(m, \sigma)$
end for
for $s = 1$ **to** γ **do**
  $\ell_s \in_R [0, D]$
  for $t = \tau_{s-1}$ **to** τ_s **do**
    $u'(t) \leftarrow u(t) + \ell_s$
  end for
end for

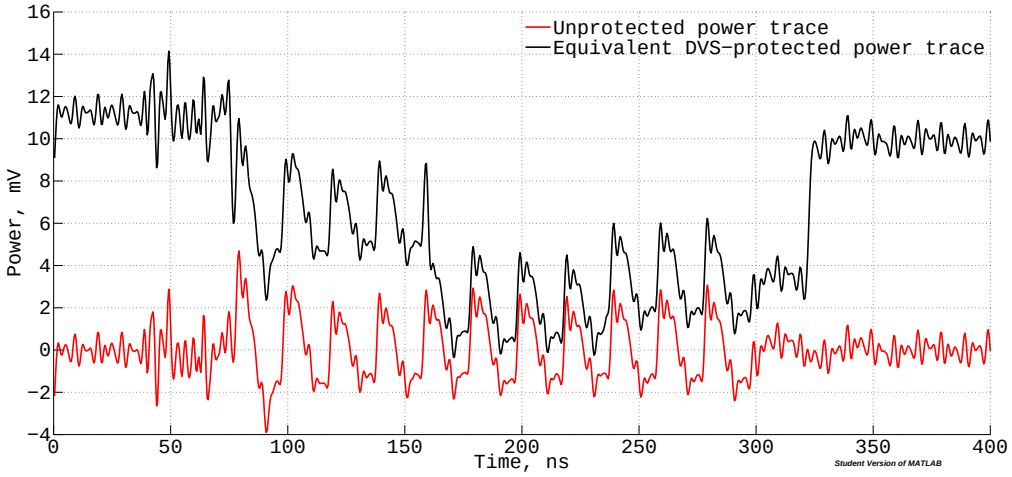


Figure 4.38: Power traces of the FPGA AES implementation. The unprotected signal is shown in red. The DVS-protected signal is shown in black.

A trace modification example is presented in Fig. 4.38, in which the trace of Fig. 4.32 was processed by Algorithm 7.

Logically, DVS decreases power analysis performance by reducing the attacker's SNR. We disposed of

1. The mean m and the standard deviation σ were arbitrary set to $m = 40$ ns and $\sigma = 5$ ns in our experiment

200,000 DVS-modified power traces. All of which were used to mount power analysis attacks under the same conditions as before, i.e., using Pearson's correlation coefficient and the Hamming distance model.

The same final round key byte used for attacks against the unprotected implementation was targeted. CPA and CSBA failed to detect the correct key byte even with 150,000 traces (Fig. 4.39(a),4.39(b). This confirms the intuition that DVS has a beneficial effect on the required number of power traces.

However CIFA was able to recover the byte from 60,000 traces and on (Fig. 4.39(c)). This illustrates that whilst CIFA is usually outperformed by CPA and CSBA, CIFA is much more resilient to DVS, to which CPA and CSBA are very sensitive.

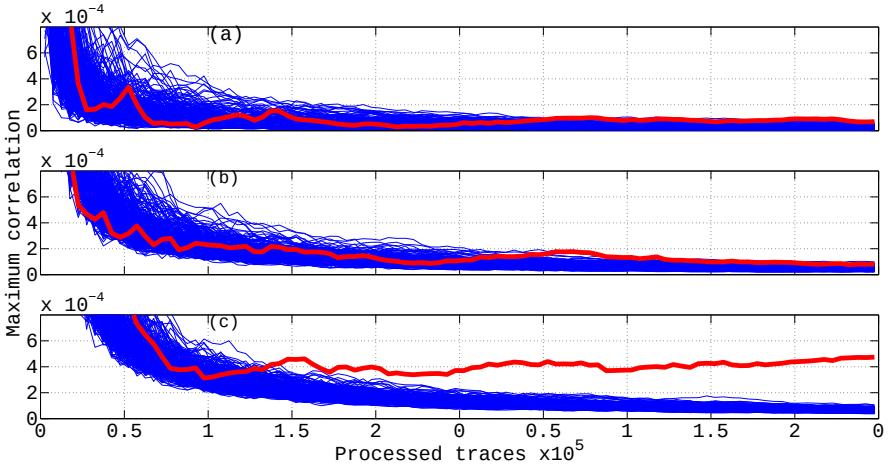


Figure 4.39: Maximum correlation coefficient for a byte of the last round AES key with simulated DVS. (a) CPA (b) CSBA (c) CIFA.

Chapter 5

Zero-Knowledge Protocols and Authenticated Encryption

Summary

This chapter presents protocols and algorithms specifically designed to achieve identification and authentication. Section 5.1 explores zero-knowledge protocols applied to the authentication of wireless sensor networks. While previous works focus the efforts of zero-knowledge protocols to authenticate single nodes in a sensor network, our work proposes collective authentication of the whole network. Section 5.1.2 recalls the Fiat-Shamir authentication scheme and present a distributed algorithm for topology-aware networks. A distributed Fiat-Shamir protocol for IoT authentication is detailed in Section 5.1.3. Lastly, we analyze the security of the proposed protocol in Section 5.1.4.

Section 5.2 presents OMD, a novel authentication encryption scheme that delivers confidentiality and integrity altogether. OMD is a keyed compression function mode of operation for nonce-based AEAD. This algorithm was proposed for consideration to CAESAR [1], a cryptographic competition for authenticated encryption focusing in security, applicability, and robustness. Section 5.2.1 introduces the novel algorithm and describes its most important features. Section 5.2.2 defines OMD's mathematical notations and syntax. The specification of the OMD cipher is explained in details in Section 5.2.3.

1. Until the publication of this thesis, the CAESAR competition has accepted our proposal through the first and second rounds. The winner remains to be announced.

5.1 Public-Key Based Lightweight Swarm Authentication

5.1.1 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 therefore susceptible to physical attacks, hard weather and communication interference.

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

This paper describes an authentication protocol based on the zero-knowledge paradigm 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.

A basic motivation for zero-knowledge protocols (ZKP) is that when a claimant A (called a *prover* in the context of zero-knowledge protocols) gives the verifier B her password, B can therefore impersonate A. Instead, ZKP addresses this concern by allowing a prover to demonstrate knowledge of a secret without revealing no useful information to the verifier in conveying this demonstration of knowledge to other parties. This is most useful in the context of wireless sensor networks and the Internet of Things, 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.

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 painfully large number of interactions to try and authenticate the network as a whole. What we propose instead is a collective perspective on authentication and not an isolated one.

5.1.2 Preliminaries

5.1.2.1 Fiat-Shamir Authentication Protocol

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 actually revealing it [GMR89, FFS88].

The algorithm first runs a one-time setup, whereby a trusted authority publishes an RSA-like 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 registers v as its public key on a trusted public server.

When a verifier $\mathcal{V}$ wishes to query $\mathcal{P}$, he uses the protocol of Fig. 5.1. $\mathcal{V}$ may run this protocol several times until he is convinced that $\mathcal{P}$ indeed knows the square root s of v modulo n.

Fig. 5.1 describes the original Fiat-Shamir authentication round [FS87], which is *honest verifier* zero-knowledge [2], and whose security is proven assuming the hardness of computing arbitrary square roots modulo a composite n, which is equivalent to factoring n.

2. 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.

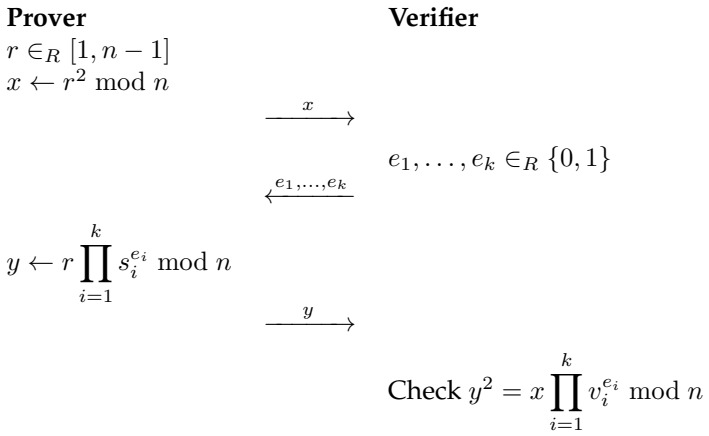


Figure 5.1: Fiat-Shamir authentication round.

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.

5.1.2.2 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 neighbors, 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 transceiver), 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 algorithm. We now describe the Mooij-Goga-Wesselink distributed algorithm for topology-aware networks [MGW]. We assume that nodes can locally detect whether a neighbor has appeared or disappeared, i.e., graph edge deletion and additions.

Each node 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:

— *Basic*: maintains a spanning tree as long as no edge is removed (it is a variant of the union-find algorithm). When a new neighbor 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 8. Note that a node only becomes active upon an event like an arriving M-message or a topology change.

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

Algorithm 8 Mooij-Goga-Wesselink algorithm, basic part.

Require: A M-message (r, d) coming from a neighbor w

1: **if** $(r, d+1) < (\mathsf{root}, \mathsf{dist})$ **then**
2: $\mathsf{parent} \leftarrow w$
3: $\mathsf{root} \leftarrow r$
4: $\mathsf{dist} \leftarrow d+1$
5: Send M-message $(\mathsf{root}, \mathsf{dist})$ to all neighbors except w
6: **end if**

The algorithm has converged once all topology change events have been processed. At that point we have a spanning tree [MGW].

For our purposes, we may assume that the network was setup and that such an algorithm 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.

5.1.3 Distributed Fiat-Shamir Authentication

5.1.3.1 The Approach

Given a network, we may consider its nodes as users and the base station as a trusted center $\mathcal{T}$. Let the number of nodes be k. In this context, each one will be given only one[3] 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 T is constructed with the algorithms presented in Section 5.1.2.2. When that happens, the base station sends an authentication request message (AR-message) to all the nodes 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 node n_i generates a private r_i and computes $x_i \leftarrow r_i^2 \bmod n$. The node n_i then sends an A-message to all its children, if any. When they respond, n_i multiplies all the x_j sent by its children together, and with its own x_i, and sends it 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 $d = \deg T$). This is illustrated for a simple network including 4 nodes and a base station in Fig. 5.2a.

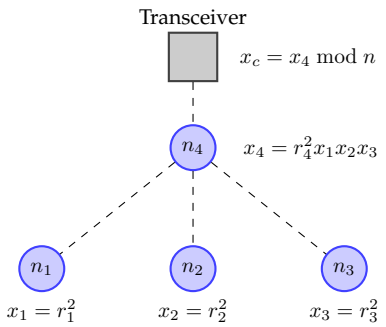
— *Step 2:* The transceiver sends a random e as an authentication challenge message (AC-message) to the nodes directly connected to it.

— *Step 3:* Upon receiving an AC-message e, each nodes n_i computes $y_i \leftarrow r_i s_i^{e_i}$. The node n_i then sends the AC-message to all its children, if any. When they respond, 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 the transceiver. This is illustrated in Fig. 5.2b.

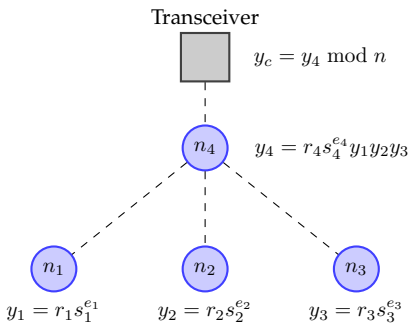
— *Step 4:* Upon receiving y_c, the transceiver checks that $y_c^2 = x_c \prod v_i^{e_i}$, where $v_1, \ldots, v_k$ are the public keys corresponding to $s_1, \ldots, s_k$ respectively.

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

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



(a) The construction of x_c.



(b) The construction of y_c.

Figure 5.2: The proposed algorithm running on a network. Each parent node aggregates the values computed by its children before transmitting it upwards to the base station.

5.1.3.2 Back-up Authentication

Network authentication may fail because of many reasons described and analyzed in detail in Section 5.1.4.3. As a consequence of the distributed nature of the algorithm we just described, a single defective node suffices for authentication to fail.

This is the intended behavior; 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.

5.1.4 Security

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

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

5.1.4.1 Soundness

Lemma 5.1 (Soundness) *If authentication succeeds then with overwhelming probability the network nodes are genuine.*

Proof: Assume that an adversary $\mathcal{A}$ has taken control of communications over 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. □

5.1.4.2 Zero-knowledge

Lemma 5.2 (Zero-knowledge) *The distributed authentication protocol 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 $\overline{e} \in_R \{0,1\}^k$ and $\overline{y} \in_R [0, n-1]$ 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 real ones.

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 $\overline{e} \in_R \{0,1\}^k$ and $\overline{y} \in_R [0, n-1]$ 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 having ℓ out of k nodes under its control still has to face the original Fiat-Shamir protocol.

5.1.4.3 Security Analysis

Choice of parameters. Let λ be a security parameter. In order 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 5.1), 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.

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

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

In average, each individual node 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 $\lambda = 128$, $k = 1024$ nodes and $t = 1$, we have $n \geq 2^{512}$, and up to 5 modular operations per node.

Root causes for 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, the transceiver 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 happen in the low battery case, i.e., almost down, or when errors appear within the algorithm the node has to perform (fault injection, malfunctioning, etc.). In that case it is expected that authentication fails.

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 5.2 proves that a passive adversary cannot learn anything valuable, and Lemma 5.1 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.

5.2 The Offset Merkle-Damgård Authenticated Cipher

5.2.1 Introduction

An authenticated encryption (AE) scheme delivers on two complementary data security goals: confidentiality (privacy) and integrity (authenticity). Historically, these goals were achieved by combining separate cryptographic primitives, one to ensure confidentiality and another to guarantee integrity [BN00]. This generic composition paradigm is neither most efficient (for instance, it requires processing the input stream at least twice) nor most robust to implementation errors [Vau02, CHVV03]. To address these concerns, the notion of AE which simultaneously achieves confidentiality and integrity was put forward [KY01, BN00, BR00] and further developed [RBBK01, Rog02, Rog04b, RS06, FFLW11] as a desirable primitive to be exposed by libraries and APIs to the end developer. Providing direct access to AE rather than requiring developers to make calls to several lower-level functions is seen as a step towards improving quality of security-critical code.

This section describes our proposal of a new authenticated cipher for consideration in the CAESAR competition. Our scheme, called Offset Merkle-Damgård (OMD), is a keyed compression function mode of operation for nonce-based AEAD. The syntax and security notions for nonce-based AEAD schemes were formalized by Rogaway in [Rog02, Rog04b]. To instantiate the OMD mode, we recommend two specific compression functions to be keyed and used in OMD, namely, the compression functions of the standard SHA-256 and SHA-512 hash functions. OMD parameterized with these two compression functions is called OMD-SHA256 and OMD-SHA512, respectively. The former is intended for 32-bit implementations and is our primary recommended algorithm, while the latter could be used specifically for 64-bit machines and is our secondary algorithm.

We believe that an AE scheme whose security is proved by a modular and easy to verify security reduction, only relying on some widely-verified standard assumption(s) on its underlying primitive(s), can get more confidence on its security compared to a scheme that demands strong and idealistic properties from its underlying primitive(s) or is not supported by a formal security proof. Provable security helps cryptanalysis efforts to be focused on analyzing the simpler underlying primitives rather than the whole scheme; hence, building confidence on the security of the scheme becomes easier if it uses well-analyzed and verified primitives.

Setting provable security in the standard model as one of our main design aims, OMD is designed as a scheme with its security goals achieved provably, based on the sole assumption that its underlying keyed compression function is a PRF; an assumption which is among the most well-known and widely-used assumption; for example, the security of the widely-employed standard HMAC algorithm is also based on this assumption [Bel06]. From a theoretical point of view, this is an advantage for OMD compared to the recently proposed permutation-based AE schemes in the literature whose security proofs rely on the *ideal* permutation assumption.

Unlike the mainstream AE schemes which are block-cipher based or permutation-based schemes, OMD is designed to be a compression function based scheme. The cryptographic community has spent more than two decades on public research and standardization activities on hash functions resulting to development of a rich source of secure and efficient compression functions. The recent announcement by Intel in July 2013 [Int13] about Intel SHA Extensions, supporting performance acceleration of the SHA family of functions (more precisely, SHA-1 and SHA-256), further encourages the decision to design a compression function based scheme. The SHA family of algorithms is heavily used in many of the most common cryptographic applications. For example, every secure web session initiation includes SHA-1, and the latest protocols involve SHA-256 as well. We believe that having a diverse set of AE schemes based on different primitives can be interesting from a practical viewpoint, providing the opportunity to choose among the AE algorithms based on what primitives have already been available and implemented and to reuse them.

OMD is patent free and suitable for widespread adoption. Our primary recommended scheme, OMD-SHA256, uses the compression function of SHA-256 [SHA95] and has features offering the following advantages over the AES-GCM scheme:

Higher quantitative security level. The proven security of OMD-SHA256 falls off, as usual for birthday-type security bounds, in $\frac{\sigma^2}{2^{256}}$ where σ is the total number of calls to the compression function; while, for the same key size and tag size, the proven security of AES-GCM [IOM12] falls off in about $\frac{\sigma'^2}{2^{128}}$ where σ' is the total number of calls to AES. That is, with the same key length and tag length, OMD-SHA256 offers higher security level than that of AES-GCM.

More flexible key size. AES-GCM only supports three different key lengths, namely 128, 192 and 256 bits. OMD-SHA256 can support any key length between 80 bits and 256 bits.

Simpler operations. OMD-SHA256 only needs the compression function of SHA-256 plus the simple operations of bitwise XOR and bitwise AND of two binary strings and (left and right) shifting a binary string. In comparison, AES-GCM in addition to calling AES requires multiplication of two arbitrary elements in $GF(2^{128})$. The field multiplication operation demand extra resources and is a complicated operation in contrast with the basic operations used in OMD-SHA256. This is important, in particular, if one does not have access to Intel CPUs supporting the PCLMULQDQ instruction for implementing AES-GCM, e.g. on low-end devices.

Resistance against software-level timing attacks. Most AES software implementations risk leaking their keys through cache timing [Ber05] unless they are implemented on machines with Intel® CPUs supporting the constant-time AES-NI and PCLMULQDQ instructions. In comparison, we note that the only operations in OMD-SHA256 are: bitwise XOR, AND and OR of two binary strings (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), fixed-distance (left and right) shift of a binary string (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), and 32-bit addition (of words in the compression function of SHA-256). These operations have the virtue of taking constant time on typical CPUs in which case the implementations can avoid timing attacks.

Using only a single well-known primitive. OMD is designed as a mode of operation for a keyed compression function. Together with block ciphers and permutations, compression functions are among the most well-known and widely used symmetric key primitives. We have a rich source of secure compression functions thanks to more than two decades of public research and standardization activities on hash functions.

Provable security based on a single widely-accepted standard assumption. The security goals of privacy and authenticity for OMD are achieved provably in the sense of reduction-based cryptography; that is, any attack against these security goals will imply an attack against the classical PRF property of the underlying compression function. We note that any keyed compression function (either a dedicated-key one or keyed via some part of its input) must provide the classical PRF property when its key is secret as otherwise it will be considered useless for any secret key application, e.g. for being used as a MAC. That is, the base PRF assumption on the compression function upon which the security of OMD relies is highly assured for compression functions of the practical, standard hash functions, thanks to the vast amount of cryptanalytic work on these functions.

Requiring minimal basic operations in addition to the core primitive. The only operations that OMD needs *in addition to* its core compression function are the basic operations of bitwise XORing two binary strings and shifting a binary string.

Integrated (one-pass) AEAD scheme. In OMD the mechanisms for providing privacy and authenticity of the message are coupled in a single pass of (a variant of) the Merkle-Damgård iteration of the compression function. This is aimed to make OMD as much efficient as possible (up to the limits that are inherent to any compression function based AEAD scheme).

Online Encryption. OMD encryption is online; that is, it outputs a stream of ciphertext as a stream of plaintext arrives with a constant latency and using constant memory. After receiving an indication that the plaintext is over, the final part of ciphertext together with the tag is output.

Internally Online Decryption. OMD decryption is *internally* online: one can generate a stream of plaintext bits as the stream of ciphertext bits comes in, but no part of the plaintext stream will be revealed before the whole ciphertext stream is decrypted and the tag is verified to be correct. That is, nothing about the decrypted plaintext should be made available to adversaries if the tag is incorrect signifying that the queried ciphertext is invalid.

Flexible key size. OMD-SHA256 can support any key length between 80 bits and 256 bits. This will be useful for applications requiring unconventional key lengths, e.g. 96-bit keys.

Efficient. If implemented with a member of the SHA family, OMD can take advantage of the newly introduced Intel® instructions that support performance acceleration of the Secure Hash Algorithm (SHA) on Intel® Architecture processors. In particular, our main recommended scheme for CAESAR, called OMD-SHA256, is aimed to get the most out of these new performance accelerating instructions.

5.2.2 Preliminaries

NOTATIONS. If $\mathcal{S}$ is a finite set, $x \xleftarrow{\$} S$ means that x is chosen from S uniformly at random. $X \leftarrow Y$ is used for denoting the assignment statement where the value of Y is assigned to X. The set of all binary strings of length n bits (for some positive integer n) is denoted as $\{0,1\}^n$, the set of all binary strings whose lengths are variable but upper-bounded by L is denoted by $\{0,1\}^{\leq L}$ and the set of all binary strings of arbitrary but finite length is denoted by $\{0,1\}^*$. For two strings X and Y we use $X||Y$ and XY analogously to denote the string obtained by concatenating Y to X. For an m-bit binary string $X = X_{m-1} \cdots X_0$ we denote the left-most bit by $\mathsf{msb}(X) = X_{m-1}$ and the right-most bit by $\mathsf{lsb}(X) = X_0$; let $X[i \cdots j] = X_i \cdots X_j$ denote a substring of X, for $0 \leq j \leq i \leq (m-1)$. Let $1^n 0^m$ denote concatenation of n ones by m zeros. For a non-negative integer i let $\langle i \rangle_m$ denote binary representation of i by an m-bit string.

For a binary string $X = X_{m-1} \cdots X_0$, let $X \ll n$ denote the left-shift operation, where the n left-most bits are discarded and the n vacated right bits are set to 0; that is, $X \ll n = X_{m-n-1} \cdots X_0 0^n$. We let $X \gg n$ denote the (unsigned) right-shift operation where the n right-most bits are discarded and the n vacated left bits are set to 0; i.e., $X \gg n = 0^n X_{m-1} \cdots X_n$. We let $X \gg_s n$ denote the *signed* right-shift operation where the n right-most bits are discarded and the n vacated left bits are filled with the left-most bit (which is considered as the sign bit); for example, $1001100 \gg_s 3 = 1111001$. If the left-most bit of X is 0 then we have $X \gg_s n = X \gg n$.

$\neg X$ means bitwise complement of X. For two binary strings X and Y, let $X \wedge Y$ and $X \vee Y$ denote, respectively, bitwise AND and bitwise OR of the strings.

The special symbol $\perp$ means that the value of a variable is undefined; we also overload this symbol and use it to signify an error. Let $|Z|$ denote the number of elements of Z if Z is a set, and the length of Z in bits if Z is a binary string. The empty string is denoted by ε and we let $|\varepsilon| = 0$. For $X \in \{0,1\}^*$ let $X[1]||X[2]\cdots||X[m] \xleftarrow{b} X$ denote partitioning X into blocks $X[i]$ such that $|X[i]| = b$ for $1 \leq i \leq m-1$ and $|X[m]| \leq b$; let $m = |X|_b$ denote length of X in b-bit blocks.

For two binary strings $X = X_{m-1} \cdots X_0$ and $Y = Y_{n-1} \cdots Y_0$, the notation $X \oplus Y$ denotes bitwise XOR of $X_{m-1} \cdots X_{m-1-\ell}$ and $Y_{n-1} \cdots Y_{n-1-\ell}$ where $\ell = min\{m-1, n-1\}$. That is, $X \oplus Y$ is a binary string whose length is equal to the length of the shorter operand and is obtained by XORing the shorter operand with an equal length left-most substring of the longer operand consisting of its left-most bits. Clearly, if X and Y have the same length then $X \oplus Y$ simply means their usual bitwise XOR. For any string X, define $X \oplus \varepsilon = \varepsilon \oplus X = \varepsilon$.

THE FIELD WITH 2^n POINTS. Let $(GF(2^n), \oplus, .)$ denote the Galois Field with 2^n points. When considering a point α in $GF(2^n)$ it can be represented in any of the following equivalent ways: (1) as an integer between 0 and 2^n, (2) as a binary string $\alpha_{n-1} \cdots \alpha_0 \in \{0,1\}^n$, or (3) as a formal polynomial $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ with binary coefficients. For example, in $GF(2^{256})$: the string $0^{254}10$, the number 2 and the polynomial X are different representations of the same field element; the string $0^{254}11$, the number 3 and the polynomial $X+1$ represent the same field element, and so forth.

The addition "$\oplus$" and multiplication "." of two elements in $GF(2^n)$ are defined as follows. The addition of two elements $\alpha, \beta \in GF(2^n)$ simply means the element obtained by bitwise XORing their representations as binary strings. For example, $2 \oplus 1 = 0^{n-2}10 \oplus 0^{n-2}01 = 0^{n-2}11 = 3$, $2 \oplus 3 = 1$, $1 \oplus 1 = 0$, and so forth. (Note that the addition operation in $GF(2^n)$ is *different* from the addition of integers module 2^n.) To multiply two elements, first choose and fix an irreducible polynomial $P_n(X)$ of degree n over $GF(2)$; for example, choose the lexicographically first polynomial among the irreducible polynomials of degree n over $GF(2)$ with a minimum number of nonzero coefficients. For example, for $n = 256$ we use $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, for $n = 512$ we use $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$.

To multiply two elements α and β in $GF(2^n)$ denoted by $\alpha \cdot \beta$ consider them as polynomials $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ and $\beta(X) = \beta_{n-1}X^{n-1} + \cdots + \beta_1 X + \beta_0$, form their product in $GF(2)$ to get $\gamma(X)$ and take the remainder of dividing $\gamma(X)$ by the irreducible polynomial $P_n(X)$.

It is easy to multiply an arbitrary field element α by the element 2 (i.e., X). We describe this for $GF(2^{256})$and $GF(2^{512})$. Let $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ then multiplying by X we get $\alpha_n X^n + \alpha_{n-1}X^{n-1} \cdots + \alpha_1 X + \alpha_0 X$; so if $\mathsf{msb}(\alpha) = 0$ then $2.\alpha = X.\alpha = \alpha \ll 1$. If $\mathsf{msb}(\alpha) = 1$ then we need to reduce the result by module $P_n(X)$, i.e., we have to add X^n to $\alpha \ll 1$. For $n = 256$ using $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, we have $X^{256} = X^{10} + X^5 + X^2 + 1 = 0^{245}10000100101$, so adding X^{256} means XORing with $0^{245}10000100101$. For $n = 512$ using $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$, we have $X^{512} = X^8 + X^5 + X^2 + 1 = 0^{503}100100101$, so adding X^{512} means XORing with $0^{503}100100101$.

In summary, for $GF(2^{256})$

$$2.\alpha = \begin{cases} \alpha \ll 1 & \text{if } \mathsf{msb}(\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{245}10000100101 & \text{if } \mathsf{msb}(\alpha) = 1 \end{cases} \tag{5.1}$$

$$= (\alpha \ll 1) \oplus ((\alpha \gg_s 255) \wedge 0^{245}10000100101) \tag{5.2}$$

and for $GF(2^{512})$

$$2.\alpha = \begin{cases} \alpha \ll 1 & \text{if } \mathsf{msb}(\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{503}100100101 & \text{if } \mathsf{msb}(\alpha) = 1 \end{cases} \tag{5.3}$$

$$= (\alpha \ll 1) \oplus ((\alpha \gg_s 511) \wedge 0^{503}100100101) \tag{5.4}$$

We note that the results computed in (5.1) and (5.2) are the same but an implementation using (5.2) will not be susceptible to the timing attacks unlike one which uses (5.1). Similarly, an implementation using (5.4) is aimed for timing attack resistance.

SYNTAX OF KEYED AND KEYLESS COMPRESSION FUNCTIONS. We denote a keyed compression function by $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$, where m and n are two positive integers, and the keyspace $\mathcal{K}$ is a non-empty set of strings. We write $F_K(H, M) = F(K; H, M)$ for every $K \in \mathcal{K}$, $H \in \{0,1\}^n$ and $M \in \{0,1\}^m$. We can alternatively think of F_K as a single argument function whose domain is $\{0,1\}^{n+m}$ and write $F_K(H||M) = F_K(H, M)$. If $|\mathcal{K}| = 1$ we assume that $\mathcal{K} = \{\varepsilon\}$, i.e., it only consists of the empty string, and in this case we call F a keyless compression function. Time_F denotes the time complexity of computing $F_K(X)$ for any $K \in \mathcal{K}$ and $X \in \{0,1\}^{n+m}$, plus the time complexity for sampling from $\mathcal{K}$.

Given a keyless compression function $F' : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ (e.g. `SHA-256` : $\{0,1\}^{256} \times \{0,1\}^{512} \to \{0,1\}^{256}$) we convert it to a keyed compression function F by borrowing k bits of its b-bit input block; i.e., we define $F_K(H, M) = F'(H, K||M)$.

CONCRETE SECURITY CONVENTIONS. As usual in the concrete-security definitions, we use the resource parameterized function $\mathbf{Adv}^{\mathsf{xxx}}_{\Pi}(\mathbf{r})$ to denote the maximal value of the adversarial advantage (i.e., $\mathbf{Adv}^{\mathsf{xxx}}_{\Pi}(\mathbf{r}) = max_{\boldsymbol{A}}\{\mathbf{Adv}^{\mathsf{xxx}}_{\Pi}(\boldsymbol{A})\}$) over all adversaries $\boldsymbol{A}$, against the xxx property of a primitive or scheme Π, that use resources bounded by $\mathbf{r}$. The resource parameter $\mathbf{r}$, depending on the notion, may include time complexity (t), length of queries and number of queries that an adversary makes to its oracles. If a resource parameter is irrelevant in the context then we omit it; e.g. for information-theoretic security bounds the time complexity t is omitted.

Let $\boldsymbol{A}$ be an adversary that returns a binary value; by $\boldsymbol{A}^{f(.)}(X) \Rightarrow 1$ we refer to the event that $\boldsymbol{A}$ on input X and access to an oracle function $f(.)$ returns 1. By time complexity of an algorithm we mean the running time, relative to some fixed model of computation plus the size of the description of the algorithm using some fixed encoding method.

PSEUDORANDOM FUNCTIONS (PRFS) AND TWEAKABLE PRFS. Let $\mathrm{Func}(m,n) = \{f : \{0,1\}^m \rightarrow \{0,1\}^n\}$ be the set of all functions from m-bit strings to n-bit strings. A random function (RF) R with m-bit input and n-bit output is a function selected uniformly at random from $\mathrm{Func}(m,n)$. We denote this by $R \xleftarrow{\$} \mathrm{Func}(m,n)$.

Let $\mathrm{Func}^{\mathcal{T}}(m,n)$ be the set of all functions $\left\{\widetilde{f} : \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n\right\}$, where $\mathcal{T}$ is a set of tweaks. A tweakable RF with the tweak space $\mathcal{T}$, m-bit input and n-bit output is a map $\widetilde{R} : \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n$ selected uniformly at random from $\mathrm{Func}^{\mathcal{T}}(m,n)$; i.e., $\widetilde{R} \xleftarrow{\$} \mathrm{Func}^{\mathcal{T}}(m,n)$. Clearly, if $\mathcal{T} = \{0,1\}^t$ then $|\mathrm{Func}^{\mathcal{T}}(m,n)| = |\mathrm{Func}(m+t,n)|$, and hence, $\widetilde{R}$ can be instantiated using a random function R with $(m+t)$-bit input and n-bit output. We use $\widetilde{R}^{\langle T\rangle}(.)$ and $\widetilde{R}(T,.)$ interchangeably, for every $T \in \mathcal{T}$. Notice that each tweak T names a random function $\widetilde{R}^{\langle T\rangle} : \{0,1\}^m \rightarrow \{0,1\}^n$ and distinct tweaks name distinct (independent).

Let $F : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$ be a keyed function and let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n$ be a keyed and tweakable function, where the key space $\mathcal{K}$ is some nonempty set. Let $F_K(.) = F(K,.)$ and $\widetilde{F}_K^{\langle T\rangle}(.) = \widetilde{F}(K,T,.)$. Let $\boldsymbol{A}$ be an adversary.

Then:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prf}}_{F}(\boldsymbol{A}) &= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{F_K(.)} \Rightarrow 1\right] - \Pr\left[R \xleftarrow{\$} \mathrm{Func}(m,n) : \boldsymbol{A}^{R(.)} \Rightarrow 1\right] \\
\mathbf{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(\boldsymbol{A}) &= \Pr\left[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{\widetilde{F}_K^{\langle.\rangle}(.)} \Rightarrow 1\right] - \Pr\left[\widetilde{R} \xleftarrow{\$} \mathrm{Func}^{\mathcal{T}}(m,n) : \boldsymbol{A}^{\widetilde{R}^{\langle.\rangle}(.)} \Rightarrow 1\right]
\end{aligned}$$

The resource parameterized advantage functions are defined accordingly, considering that the adversarial resources of interest here are the time complexity (t) of the adversary and the total number of queries (q) asked by the adversary (note that we just consider fixed-input-length functions, so the lengths of queries are fixed and known). We say that F is $(t,q;\epsilon)$-PRF if $\mathbf{Adv}^{\mathrm{prf}}_{F}(t,q) \leq \epsilon$. We say that $\widetilde{F}$ is $(t,q;\epsilon)$-tweakable PRF if $\mathbf{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(t,q) \leq \epsilon$.

5.2.2.1 Security Definitions and Goals

Conventions and Preliminary Definitions OMD is a nonce-based AEAD. Therefore, we aim to achieve the security notions for AEAD schemes as formalized in [Rog02].

NONCE RESPECTING ADVERSARIES. Let $\boldsymbol{A}$ be an adversary. We say that $\boldsymbol{A}$ is nonce-respecting if it never repeats a nonce in its *encryption* queries. That is, if $\boldsymbol{A}$ queries the encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ on $(N_1,A_1,M_1)\cdots(N_q,A_q,M_q)$ then $N_1,\cdots,N_q$ must be distinct.

In the following, we define the conventional security properties of an AEAD; namely, the privacy notion ("confidentiality for the plaintext") and the authenticity notion ("integrity for the nonce, associated data, and plaintext").

PRIVACY OF AEAD SCHEMES. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a nonce-based AEAD scheme. Let $\boldsymbol{A}$ be a nonce-respecting adversary. $\boldsymbol{A}$ is provided with an oracle which can be either a real encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ such that on input (N, A, M) returns $\mathbb{C} = \mathcal{E}_K(N, A, M)$, or a fake encryption oracle $\$(\cdot,\cdot,\cdot)$ which on any input (N, A, M) returns $|\mathbb{C}|$ fresh random bits. The advantage of $\boldsymbol{A}$ in mounting a chosen plaintext attack (CPA) against the privacy property of Π is measured as follows:

$$\mathbf{Adv}^{\mathrm{priv}}_{\Pi}(\boldsymbol{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[\boldsymbol{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1].$$

This privacy notion, also called indistinguishability of ciphertext from random bits under CPA (IND-CPA), is defined originally in [RBBK01] and is a stronger variant of the classical IND-CPA notion [BDJR97, BN00] for conventional symmetric-key encryption schemes.

RESOURCE PARAMETERS FOR THE CPA ADVERSARY. Let the CPA-adversary $\boldsymbol{A}$ make queries $(N_1, A_1, M_1) \cdots (N_{q_e}, A_{q_e}, M_{q_e})$. We define the resource parameters of $\boldsymbol{A}$ as $(t, q_e, \sigma_A, \sigma_M, L_{\max})$ where t is the time complexity, q_e is the total number of encryption queries, $\sigma_A = \sum_{i=1}^{q} |A_i|$ is the total length of associated data in bits, $\sigma_M = \sum_{i=1}^{q} |M_i|$ is the total length of messages in bits, and $L_{\max}$ is the maximum length of each query in bits.

We remind that absence of a resource parameter means that the parameter is irrelevant in the context and hence omitted.

AUTHENTICITY OF AEAD SCHEMES. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a nonce-based AEAD scheme. Let $\boldsymbol{A}$ be a nonce-respecting adversary. We stress that nonce-respecting is only regarded for the encryption queries; that is, $\boldsymbol{A}$ can repeat nonces during its decryption queries and it can also ask an encryption query with a nonce that was already used in a decryption query. Let $\mathcal{A}$ be provided with the encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ and the decryption oracle $\mathcal{D}_K(\cdot,\cdot,\cdot)$; that is, we consider adversaries that can mount chosen ciphertext attacks (CCA). We say that $\boldsymbol{A}$ forges if it makes a decryption query $(N, A, \mathbb{C})$ such that $\mathcal{D}_K(N, A, \mathbb{C}) \neq \perp$ and no previous encryption query $\mathcal{E}_K(N, A, M)$ returned $\mathbb{C}$.

$$\mathbf{Adv}^{\mathrm{auth}}_{\Pi}(\boldsymbol{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot),\, \mathcal{D}_K(\cdot,\cdot,\cdot)} \text{ forges}].$$

This authenticity notion, also called integrity of ciphertext (INT-CTXT) under CCA attacks, is defined originally in [BN00].

RESOURCE PARAMETERS FOR THE CCA ADVERSARY. Let the CCA-adversary $\boldsymbol{A}$ make encryption queries $(N_1, A_1, M_1) \cdots (N_{q_e}, A_{q_e}, M_{q_e})$ and decryption queries $(N'_1, A'_1, \mathbb{C}'_1) \cdots (N'_{q_v}, A'_{q_v}, \mathbb{C}'_{q_v})$. We define the resource parameters of $\boldsymbol{A}$ as $(t, q_e, q_v, \sigma_A, \sigma_M, \sigma_{A'}, \sigma_{\mathbb{C}'}, L_{\max})$, where t is the time complexity, q_e and q_v are respectively the total number of encryption queries and decryption queries, $L_{\max}$ is the maximum length of each query in bits, $\sigma_A = \sum_{i=1}^{q_e} |A_i|$, $\sigma_M = \sum_{i=1}^{q_e} |M_i|$, $\sigma_{A'} = \sum_{i=1}^{q_v} |A'_i|$ and $\sigma_{\mathbb{C}'} = \sum_{i=1}^{q_v} (|\mathbb{C}'_i| - \tau)$.

We remind that absence of a resource parameter means that the parameter is irrelevant in the context and hence omitted.

Remark *The use of the aforementioned privacy (IND-CPA) and authenticity (INT-CTXT) goals to define security of AE schemes dates back to [BN00] where it was shown that if an AE scheme satisfies the combination of IND-CPA and INT-CTXT properties then it will also fulfill indistinguishability under the strongest form of chosen-ciphertext attack (IND-CCA) which, in turn, is equivalent to non-malleability under chosen-ciphertext attack (NM-CCA).*

Remark *The nonce-respecting assumption on the adversary is justified as follows. The nonce would typically be a counter (message number) maintained by the sender who encrypts the messages. In practice, an implementation must make sure that no nonce gets repeated within a session (i.e., the lifetime of the current encryption key). As the nonce N is needed both to encrypt and to decrypt; it would be typically communicated* in clear *between the sender and the receiver. Note that nonce-respecting is only assumed with respect to the encryption queries, reflecting the fact that the sender who encrypts a message is the party that is responsible for providing fresh nonces and the receiver may be stateless.*

Remark *OMD v1.0 requires the nonce-respecting condition: it does not provide security if the nonce is repeated.*

5.2.2.2 Quantitative Security Level of OMD-SHA256

Using the concrete security bounds in Section 5.2.2.4 and letting $n = 256$ (the hash size for SHA-256) one can calculate the quantitative security (privacy and authenticity) levels of OMD-SHA256 for any set of fixed values for the adversarial resource parameters. For this purpose, we make the assumption that the function $F_K(H, M) = \text{SHA-256}(H, K||0^{256-k}||M)$ is a PRF providing a k-bit security; as (to the best of our knowledge) there is no known attack with complexity less than 2^k against it. We note that having only a single (input, output) pair for F_K one can mount an *offline* exhaustive search attack with time complexity 2^k.

For the privacy property of OMD-SHA256 (i.e., "confidentiality for the plaintext") the security bound falls off in $\frac{3\sigma_e^2}{2^{256}}$; that is, if the adversary has *online* data complexity about $\sigma_e = 2^{127}$, where σ_e denotes the total number of blocks in all inputs for encryption and decryption as defined in Section 5.2.2.1. We note that, giving a single measure for the bit security level of OMD-SHA256 is a bit tricky as the terms determining the security bound and the resources are different in nature (e.g. we have both offline complexity and online complexity); nevertheless, one can roughly consider $min\{k, 127\}$ as the bit security.

For the authenticity property of OMD-SHA256 (i.e., "integrity for the public message number, the associated data and the plaintext") the security bound falls off in $\frac{3\sigma^2}{2^{256}} + \frac{q_v \ell_{\max}}{2^{256}} + \frac{q_v}{2^\tau}$; that is, if the adversary has *online* data complexity about $\sigma_e = 2^{127}$, or $q_v \ell_{\max} = 2^{256}$, or $q_v = 2^\tau$ (we refer to Section 5.2.2.1 for definitions of the resource parameters). As a single measure for the bit security of OMD-SHA256 for the authenticity goal, one can roughly consider $min\{k, 127, \tau\}$.

Remark *We note that a single measure for the "bit security level" should be interpreted carefully regarding the different online/offline nature of the resources used for complexity measures. For example, just based on our bit security levels for OMD-SHA256 one may think that a key length (k) larger than 127 bits or larger than the tag length (τ) is not useful, but this is not true because, for example, while the role of τ is to prevent online attacks, a large k can help prevent (mainly) offline key recovery attacks (that may only use one online query).*

5.2.2.3 Quantitative Security Level of OMD-SHA512

Using the concrete security bounds in Section 5.2.2.4 and letting $n = 512$ (the hash size for SHA-512) one can calculate the quantitative security (privacy and authenticity) levels of OMD-SHA512 for any set of fixed values for the adversarial resource parameters. For this purpose, we make the assumption that the function $F_K(H, M) = \text{SHA-512}(H, K||0^{512-k}||M)$ is a PRF providing a k-bit security; as (to the best of our knowledge) there is no known attack with complexity less than 2^k against it. We note that having only a single (input, output) pair for F_K one can mount an *offline* exhaustive search attack with time complexity 2^k.

For the privacy property of OMD-SHA512 (i.e., "confidentiality for the plaintext") the security bound falls off in $\frac{3\sigma_e^2}{2^{512}}$; that is, if the adversary has *online* data complexity about $\sigma_e = 2^{255}$, where σ_e denotes the total number of blocks in all inputs for encryption and decryption as defined in Section 5.2.2.1. We note that, giving a single measure for the bit security level of OMD-SHA512 is a bit tricky as the terms determining the security bound and the resources are different in nature (e.g. we have both offline complexity and online complexity); nevertheless, one can roughly consider $min\{k, 255\}$ as the bit security.

For the authenticity property of OMD-SHA512 (i.e., "integrity for the public message number, the associated data and the plaintext") the security bound falls off in $\frac{3\sigma^2}{2^{512}} + \frac{q_v \ell_{\max}}{2^{512}} + \frac{q_v}{2^\tau}$; that is, if the adversary has *online* data complexity about $\sigma_e = 2^{255}$, or $q_v \ell_{\max} = 2^{512}$, or $q_v = 2^\tau$ (we refer to Section 5.2.2.1 for definitions of the resource parameters). As a single measure for the bit security of OMD-SHA512 for the authenticity goal, one can roughly consider $min\{k, 255, \tau\}$.

Remark *We note that a single measure for the "bit security level" should be interpreted carefully regarding the different online/offline nature of the resources used for complexity measures. For example, just based on our bit security levels for OMD-SHA512 one may think that a key length (k) larger than 255 bits or larger than the tag length (τ) is not necessary, but this is not true because, for example, while the role of τ is to prevent online attacks, a large k can help prevent (mainly) offline key recovery attacks (that may only use one online query).*

5.2.2.4 Security Proofs

Theorem 5.3 provides the security bounds of OMD.

Theorem 5.3 *Fix $n \geq 1$ and $\tau \in \{0, 1, \cdots, n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ be a PRF, where the key space $\mathcal{K} = \{0,1\}^k$ for $k \geq 1$ and $1 \leq m \leq n$. Then*

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{OMD}[F,\tau]}(t, q_e, \sigma_e, \ell_{\max}) &\leq \mathbf{Adv}^{\mathrm{prf}}_{F}(t', 2\sigma_e) + \frac{3\sigma_e^2}{2^n} \\
\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{OMD}[F,\tau]}(t, q_e, q_v, \sigma, \ell_{\max}) &\leq \mathbf{Adv}^{\mathrm{prf}}_{F}(t', 2\sigma) + \frac{3\sigma^2}{2^n} + \frac{q_v \ell_{\max}}{2^n} + \frac{q_v}{2^\tau}
\end{aligned}$$

where q_e and q_v are, respectively, the number of encryption and decryption queries, ℓ_{max} denotes the maximum number of m-bit blocks in an encryption or decryption query, $t' = t + cn\sigma$ for some constant c, and σ_e and σ are the total number of calls to the underlying compression function F in all queries asked by the CPA *and* CCA *adversaries against the privacy and authenticity of the scheme, respectively.*

The proof is obtained by combining Lemma 5.4 in subsection 5.2.2.5 with Lemma 5.5 and Lemma 5.6 in subsection 5.2.2.6.

Note. Referring to subsection 5.2.2.1 for definitions of the resource parameters, it can be seen that: $\sigma_e = \lceil \sigma_M/m \rceil + \lceil \sigma_A/(n+m) \rceil + q_e + 2$; $\sigma = \lceil (\sigma_M + \sigma_{\mathbb{C}'})/m \rceil + \lceil (\sigma_A + \sigma_{A'})/(n+m) \rceil + q + 2$; and $\ell_{\max} = \lceil L_{\max}/m \rceil$.

5.2.2.5 Generalization of OMD Based on Tweakable Random Functions

Fig. 5.3 shows the $\mathbb{OMD}[\widetilde{R}, \tau]$ scheme which is a generalization of OMD$[F, \tau]$ using a tweakable random function $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$. The tweak space $\mathcal{T}$ consists of five mutually exclusive sets of tweaks, namely $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathbb{N} \times \{0\} \cup \mathbb{N} \times \{1\}$, where $\mathcal{N} = \{0,1\}^{|N|}$ is the set of nonces and $\mathbb{N}$ is the set of positive integers.

Lemma 5.4 *Let $\mathbb{OMD}[\widetilde{R}, \tau]$ be the scheme shown in Fig. 5.3. Then*

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{priv}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e, \sigma_e, \ell_{\max}) &= 0 \\
\mathbf{Adv}^{\mathrm{auth}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e, q_v, \sigma, \ell_{\max}) &\leq \frac{q_v \ell_{\max}}{2^n} + \frac{q_v}{2^\tau}
\end{aligned}$$

where q_e and q_v are, respectively, the number of encryption and decryption queries, $\ell_{\max}$ denotes the maximum number of m-bit blocks in an encryption or decryption query, and σ_e and σ are the total number of calls to the underlying tweakable random function $\widetilde{R}$ in all queries asked by the CPA *and* CCA *adversaries against the privacy and authenticity of the scheme, respectively.*

The proof of the privacy bound is straightforward. Let $\mathcal{A}$ be a CPA adversary that asks (encryption) queries $(N^1, A^1, M^1), \cdots, (N^{q_e}, A^{q_e}, M^{q_e})$ where all N^x values (for $1 \leq x \leq q_e$) are distinct due to the nonce-respecting assumption on the adversary $\mathcal{A}$. Referring to Fig. 5.3, this means that we are applying

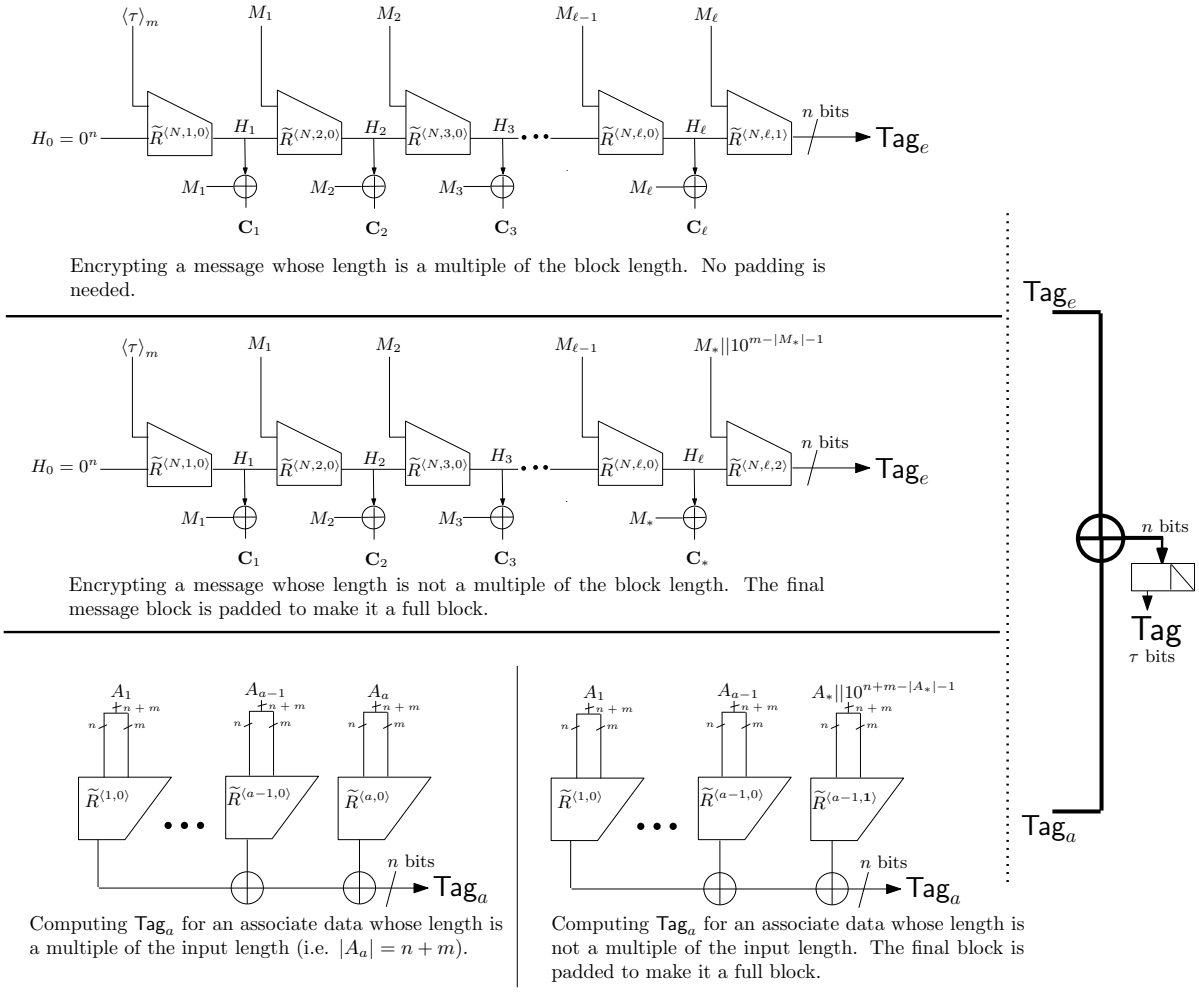


Figure 5.3: The $\mathbb{OMD}[\widetilde{R}, \tau]$ scheme using a tweakable random function $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ (i.e., $\widetilde{R} \xleftarrow{R} \mathrm{Func}^{\mathcal{T}}(n+m, n)$). The tweak space $\mathcal{T}$ consists of five mutually exclusive sets of tweaks, namely $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathbb{N} \times \{0\} \cup \mathbb{N} \times \{1\}$, where $\mathcal{N} = \{0,1\}^{|N|}$ is the set of nonces, $\mathbb{N}$ is the set of positive integers.

independent random functions $\widetilde{R}^{N_x,i,j}$ each to a single point, hence the images that the adversary sees (i.e., $\mathbb{C}^x$ for $1 \le x \le q_e$) are fresh uniformly random values.

The authenticity bound can be shown by a straightforward but lengthy case analysis. First we consider the single verification case where the adversary only makes one decryption (verification) query and then we will use the generic result of Bellare et al. [BGM04] to get a bound against adversaries that make multiple (say q_v) verification queries. Let $\mathcal{A}$ be a CCA adversary making encryption queries $(N^1, A^1, M^1), \cdots, (N^{q_e}, A^{q_e}, M^{q_e})$. Let $M^i = M^i_1, \cdots, M^i_{\ell_i}$ or $M^i = M^i_1, \cdots, M^i_{\ell_i - 1} M^i_*$ be the message queries and $A^i = A^i_1, \cdots, A^i_{a_i}$ or $A^i = A^i_1, \cdots, A^i_{a_i - 1} A^i_*$ be the associated data queries. Let $\mathbb{C}^i = C^i || \mathsf{Tag}^i$ be the ciphertext received for query (N^i, A^i, M^i). That is, we use superscripts to indicate query numbers and subscripts to denote the block indices in each query.

Let $(N, A, \mathbb{C})$ be the forgery attempt by the adversary, where $N \in \{0,1\}^{|N|}$ is the nonce, $A = A_1, \cdots, A_a$ or $A = A_1, \cdots, A_{a-1} A_*$ is the associate data, $\mathbb{C} = C || \mathsf{Tag}$ is the ciphertext where $C = C_1, \cdots, C_\ell$ (where $|C_i| = m$ for $1 \le i \le \ell$) or $C = C_1, \cdots, C_{\ell-1} C_*$ (where $|C_i| = m$ for $1 \le i \le \ell - 1$ and $|C_*| < m$), and $\mathsf{Tag} = (\mathsf{Tag}_e \oplus \mathsf{Tag}_a)[n-1, \cdots, n-\tau] \in \{0,1\}^\tau$ is the tag. Let $M = M_1, \cdots, M_\ell$ or $M = M_1, \cdots, M_{\ell-1} M_*$ denote the corresponding decrypted messages, respectively. Note that no superscripts are used for the strings in the alleged forgery by the adversary. We have the following disjoint cases:

① $N \notin \{N^1, \cdots, N^{q_e}\}$. Adversary has to find a correct Tag that is the first τ bits of the value

$\widetilde{R}^{\langle N,x,y\rangle}(\text{final input}) \oplus \mathsf{Tag}_a$ but has not seen any image under $\widetilde{R}^{\langle N,x,y\rangle}(.)$, hence the probability that the adversary can succeed in doing this is $2^{-\tau}$. By "final input" we mean $H_\ell||M_\ell$ or $H_\ell||M_*||10^{m-|M_*|-1}$ when $|C| \neq 0$ in which case the final tweak used to generate Tag_e will be either $\langle N,\ell,1\rangle$ or $\langle N,\ell,2\rangle$ (depending on whether the final block is a full block or not); otherwise (i.e., for empty message) the "final input" will be $H_0||\langle\tau\rangle_m$ and hence the final tweak used to generate Tag_e will be $\langle N,1,0\rangle$.

② $N = N^i$, $|C| \neq |C^i|$, and one of $|C|$ and $|C^i|$ is a multiple of m but the other is not. We can ignore all queries other than the i^{th} query since the responses to such queries are random and unrelated (because of using different nonces) to the adversary's task to make the alleged forgery $N, A, \mathbb{C}$ with $N = N^i$. That is, we can assume that adversary has only made a single encryption query (N^i, A^i, M^i) and received $C^i||\mathsf{Tag}^i$. Then as in Case 1 the adversary has to find a correct Tag, i.e., the first τ bits of the value $\widetilde{R}^{\langle N,x,y\rangle}(\text{final input}) \oplus \mathsf{Tag}_a$, but has not seen any image under $\widetilde{R}^{\langle N,x,y\rangle}(.)$. Note that we can even give Tag_a to the adversary. More precisely, consider the case that $|C^i|$ is a multiple of m but $|C|$ is not; then adversary must guess the first τbits of the value $\widetilde{R}^{\langle N,\ell,2\rangle}(\text{final input}) \oplus \mathsf{Tag}_a$, but has seen no image under $\widetilde{R}^{\langle N,\ell,2\rangle}(.)$. Similarly, in the case that $|C|$ is a multiple of m but $|C^i|$ is not, the adversary must guess the first τ bits of the value $\widetilde{R}^{\langle N,x,y\rangle}(\text{final input}) \oplus \mathsf{Tag}_a$ for $(N,x,y) = (N,1,0)$ if $|C| = 0$ or $(N,x,y) = (N,\ell,1)$ if $|C| \neq 0$, but the adversary has seen no image under $\widetilde{R}^{\langle N,x,y\rangle}(.)$ under either case. Therefore, the probability that the adversary can succeed in guessing Tag is $2^{-\tau}$.

③ $N = N^i$, $|C| \neq |C^i|$, and either both $|C|$ and $|C^i|$ are multiple of m or none of them is. We may ignore all queries but the i^{th} query as responses to such queries are unrelated to the adversary's task at hand. If both $|C|$ and $|C^i|$ are multiple of m then $|C| \neq |C^i|$ means that $\ell \neq \ell^i$, so from (the top of) Fig. 5.3 it can be easily seen that in this case even if the adversary knows Tag_a it must still guess the first τ bits of the output of the random function $\widetilde{R}^{\langle N,\ell,1\rangle}$ while it has seen no image of this function; the probability to succeed in guessing Tag is clearly $2^{-\tau}$. Now, let's consider the case that neither $|C|$ nor $|C^i|$ is a multiple of m then $|C| \neq |C^i|$ means that we have two cases: (1) $\ell \neq \ell^i$, and (2) $\ell = \ell^i$ but $|C_*| \neq |C^i_*|$. In the first case, it can be seen the adversary must guess the first τ bits of the random function $\widetilde{R}^{\langle N,\ell,2\rangle}$ while has seen no image of this function; the chance to do so is clearly $2^{-\tau}$. In the second case, the adversary must guess the first τ bits of $\widetilde{R}^{\langle N,\ell,2\rangle}((M_* \oplus C_*)||(M_*||10^{m-|M_*|-1}))$ while it has seen (τ bits of) a single image of this function for one different domain point, namely $((M^i_* \oplus C^i_*)||(M^i_*||10^{m-|M^i_*|-1})$; the probability to succeed in this case is again $2^{-\tau}$. (Note that $|M_*| = |C_*|$. Using 10^* padding for processing messages whose length is not a multiple of m is essential for this part.)

④ $N = N^i$, $|C| = |C^i|$, and $A \neq A^i$. We can ignore all queries except the i^{th} query because the responses to such queries are random and unrelated to the adversary's task to make the alleged forgery $N, A, \mathbb{C}$ with $N = N^i$. That is, we can assume that adversary has only made a single encryption query (N^i, A^i, M^i) and received $C^i||\mathsf{Tag}^i$. It aims to forge using the same nonce but a different associated data A. The adversary must find a correct $\mathsf{Tag} = (\mathsf{Tag}_e + \mathsf{Tag}_a)[n-1,\cdots,n-\tau]$. We consider two sub-cases: ④a $|A| \neq 0$ and ④b $|A| = 0$.

④a In this case, let's assume that we even provide the adversary with all the functions $\widetilde{R}^{\langle N,x,y\rangle}(.)$, so that the adversary can compute the correct value of Tag_e. Then the adversary's task will reduce to guessing a correct value for the first τ bits of Tag_a. The only relevant information that the adversary has is the first τ bits of Tag^i_a. We show that even if the whole Tag^i_a is given to the adversary, the chance to correctly guess the first τ bits of Tag_a is still $2^{-\tau}$. This is done by a simple case analysis:

1. if only one of $|A|$ and $|A^i|$ is a multiple of $n+m$ then it is easy to see from Fig. 5.3 that the probability to guess the first τ bits of Tag_a is still $2^{-\tau}$;
2. if $a \neq a_i$ then again from Fig. 5.3 we can see that the probability to guess the first τ bits of Tag_a is $2^{-\tau}$;
3. otherwise, we have $a = a_i$ and either both $|A|$ and $|A^i|$ are multiple of $n+m$ or neither of them is a multiple of $n+m$. These two cases are similar. Let's consider the first one. As we have $A \neq A^i$ then it must be the case that for some j we have $A_j \neq A^i_j$. So, the j^{th} value XORed to Tag_a, i.e., $\widetilde{R}^{\langle j,0\rangle}(A_j)$ is a fresh n-bit random value; hence the adversary's chance to guess the first τ bits of Tag_a is $2^{-\tau}$.

(4b) In this case the adversary has seen $C^i||\mathsf{Tag}^i$, where $\mathsf{Tag}^i = (\mathsf{Tag}_e^i \oplus \mathsf{Tag}_a^i)[n-1,\cdots,n-\tau]$. To get the forged tuple $(N,\varepsilon,C||\mathsf{Tag})$ be accepted and decrypted, it must find the value of $\mathsf{Tag} = \mathsf{Tag}_e[n-1,\cdots,n-\tau]$ (as $\mathsf{Tag}_a = 0^n$ in this case). Now let's give the adversary all functions $\widetilde{R}^{\langle N,x,0\rangle}(.)$ for $1 \le x \le \ell$. Even in this case, the adversary has seen no image of the function $\widetilde{R}^{\langle N,x,j\rangle}(.)$ for $j \in \{1,2\}$, since the value $\mathsf{Tag}^i = \mathsf{Tag}_e^i \oplus \mathsf{Tag}_a^i$ that adversary has seen does not reveal any information about Tag_e^i noting that Tag_a^i is random and unrevealed to the adversary. So, the probability that the adversary can correctly guess the first τ bits of $\mathsf{Tag}_e = \widetilde{R}^{\langle N,\ell,j\rangle}$(final input) for $j = \{1,2\}$ is $2^{-\tau}$. (Note that $j = 1$ when $|C|$ is a multiple of m and $j = 2$ when $|C|$ is not a multiple of m).

(5) $N = N^i$, $A = A^i$, and $|C| = |C^i| = \ell m$ is a multiple of m. We can again ignore all queries except the i^{th} query. Let's assume that we make all functions $\widetilde{R}^{\langle x,y\rangle}$ (for $x \ge 1$ and $y \in \{0,1\}$) used in processing the associate data public to the adversary; i.e., assume that the adversary even knows the values of Tag_a and Tag_a^i. Now remember that the adversary must not repeat the known tuple $(N^i, A^i, C^i||\mathsf{Tag}^i)$ as its decryption query, so it must be the case that $C \neq C^i$ as otherwise any $\mathsf{Tag} \neq \mathsf{Tag}^i$ will be incorrect and rejected. Therefore, we may assume that the alleged forgery will be of the form $(N,A,C||\mathsf{Tag})$ such that $C_j \neq C_j^i$ for some $1 \le j \le \ell$. Now referring to (the top of) Fig. 5.3 it is easy to see that if $C_\ell \neq C_\ell^i$ then the probability that the adversary can correctly guess the value of Tag is $2^{-\tau}$; otherwise there are two cases: (1) if $H_\ell \neq H_\ell^i$ the chance that Tag is correct is $2^{-\tau}$; (2) if the event $H_\ell = H_\ell^i$ happens then adversary can simply use $\mathsf{Tag} = \mathsf{Tag}^i$, but this event only happens with probability at most $\ell 2^{-n}$ noting that $|H_i| = n$ (note that we credit the adversary for any possible collision in the iteration, there are ℓ blocks and the probability of each collision under the random function is 2^{-n}). So, the total success probability in this case is bounded by $\dfrac{1}{2^\tau} + \dfrac{\ell}{2^n}$.

(6) $N = N^i$, $A = A^i$, and $|C| = |C^i|$ is not a multiple of m. It is easy to see from Fig. 5.3 that the analysis of this case is the same as that of Case 5 and the success probability of the adversary is bounded by $\dfrac{1}{2^\tau} + \dfrac{\ell}{2^n}$.

Finally, using the results of Bellare et al. [BGM04] we get the bound against adversaries that make q_v decryption (verification) queries as $\dfrac{q_v}{2^\tau} + \dfrac{q_v \ell}{2^n}$.

5.2.2.6 Instantiating Tweakable RFs with PRFs

We proceed to complete the proof of Theorem 5.3 in two steps.

(1) Replace the tweakable RF $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ in $\mathbb{OMD}$ with a tweakable PRF $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^k$. The following lemma states the classical bound on the security loss induced by this replacement step. The proof is a straightforward reduction and omitted here.

Lemma 5.5 *Let $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a tweakable RF and $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a tweakable PRF. Then*

$$\begin{aligned}
\mathbf{Adv}^{\text{priv}}_{\mathbb{OMD}[\widetilde{F},\tau]}(t,q_e,\sigma_e,\ell_{\max}) &\le \mathbf{Adv}^{\text{priv}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e,\sigma_e,\ell_{\max}) + \mathbf{Adv}^{\widetilde{\text{prf}}}_{\widetilde{F}}(t',\sigma_e) \\
\mathbf{Adv}^{\text{auth}}_{\mathbb{OMD}[\widetilde{F},\tau]}(t,q_e,q_v,\sigma,\ell_{\max}) &\le \mathbf{Adv}^{\text{auth}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e,q_v,\sigma,\ell_{\max}) + \mathbf{Adv}^{\widetilde{\text{prf}}}_{\widetilde{F}}(t'',\sigma)
\end{aligned}$$

where q_e and q_v are, respectively, the number of encryption and decryption queries, $q = q_e + q_v$, $\ell_{\max}$ denotes the maximum number of m-bit blocks in an encryption or decryption query, $t' = t + cn\sigma_e$ and $t'' = t + c'n\sigma$ for some constants c, c', and σ_e and σ are the total number of calls to the underlying compression function F in all queries asked by the CPA *and* CCA *adversaries against the privacy and authenticity of the scheme, respectively.*

② We instantiate a tweakable PRF using a PRF by means of XORing (part of) the input by a mask generated as a function of the key and tweak as shown in Fig. 5.4. This method to tweak a PRF is (essentially) the XE method of [Rog04a]. In OMD the tweaks are of the form $T = (\alpha, i, j)$ where $\alpha \in \mathcal{N} \cup \{\varepsilon\}$, $1 \leq i \leq 2^{n-8}$ and $j \in \{0, 1, 2\}$. We note that not all combinations are used; for example, if $\alpha = \varepsilon$ (empty) which corresponds to processing of the associate data in Fig. 5.5 then $j \neq 2$. The masking function $\Delta_K(T) = \Delta_K(\alpha, i, j)$ outputs an n-bit mask such that the following two properties hold for any fixed string $H \in \{0,1\}^n$:

1. $\Pr[\Delta_K(\alpha, i, j) = H] \leq 2^{-n}$ for any (α, i, j)
2. $\Pr[\Delta_K(\alpha, i, j) \oplus \Delta_K(\alpha', i', j') = H] \leq 2^{-n}$ for $(\alpha, i, j) \neq (\alpha', i', j')$

where the probabilities are taken over random selection of the secret key K.

It is easy to verify that these two properties are satisfied by the specific masking scheme of OMD as described in Section 5.2.3.

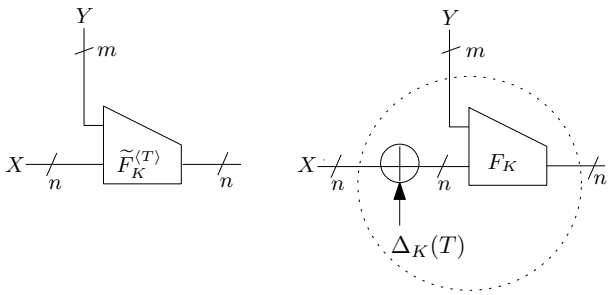


Figure 5.4: Building a tweakable PRF $\widetilde{F}_K^{\langle T\rangle} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ using a PRF $F_K : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. There are several efficient ways to define the masking function $\Delta(T)$ [Rog04a, CS07, KR11]. We use the method of [KR11].

Lemma 5.6 *Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a function family with key space $\mathcal{K}$. Let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be defined by $\widetilde{F}_K^{\langle T\rangle}(X||Y) = F_K((X \oplus \Delta(T))||Y)$ for every $T \in \mathcal{T}, K \in \mathcal{K}$, $X \in \{0,1\}^n, Y \in \{0,1\}^m$ and $\Delta_K(T)$ is the masking function of OMD as defined in Section 5.2.3. If F is PRF then $\widetilde{F}$ is tweakable PRF. More precisely*

$$\mathbf{Adv}^{\widetilde{\text{prf}}}_{\widetilde{F}}(t, q) \leq \mathbf{Adv}^{\text{prf}}_{F}(t', 2q) + \frac{3q^2}{2^n}$$

.

The proof is a simple adaptation of a similar result on the security of the XE construction (to tweak a block-cipher) in [KR11]. As we use a PRF rather than PRP, our bound has two main terms. The first term is a single birthday bound loss of $\frac{0.5q^2}{2^n}$ to take care of the case that a collision might happen when computing the initial mask $\Delta_{N,0,0} = F_K(N||10^{n-1-|N|}, 0^m)$ using a PRF (F) rather than a PRP (as in [KR11]). The analysis of the remaining term (i.e., $\frac{2.5q^2}{2^n}$) is essentially the same as the similar part in [KR11], but we note that in the context of our construction as we are directly dealing with PRFs unlike [KR11] in which PRPs are used, the bound obtained here does not have any loss terms caused by the switching (PRF/PRP) lemma. Therefore, instead of the $\frac{6q^2}{2^n}$ bound in [KR11] (from which $\frac{3.5q^2}{2^n}$ is due to using the switching lemma) our bound has only $\frac{2.5q^2}{2^n}$.

5.2.3 Specification of OMD

OMD is a compression function mode of operation for nonce-based authenticated encryption with associated data (AEAD). We use the syntax of AEAD schemes following [Rog02].

SYNTAX OF AN AEAD SCHEME. A nonce-based authenticated encryption with associated data, AEAD for short, is a symmetric key scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space $\mathcal{K}$ is some non-empty finite set. The encryption algorithm $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C} \cup \{\perp\}$ takes four arguments, a secret key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, an associated data (a.k.a. header data) $A \in \mathcal{A}$ and a message $M \in \mathcal{M}$, and returns either a ciphertext $\mathbb{C} \in \mathcal{C}$ or a special symbol $\perp$ indicating an error. The decryption algorithm $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ takes four arguments $(K, N, A, \mathbb{C})$ and either outputs a message $M \in \mathcal{M}$ or an error indicator $\perp$.

For correctness of the scheme, it is required that $\mathcal{D}(K, N, A, \mathbb{C}) = M$ for any $\mathbb{C}$ such that $\mathbb{C} = \mathcal{E}(K, N, A, M)$. It is also assumed that if algorithms $\mathcal{E}$ and $\mathcal{D}$ receive parameter not belonging to their specified domain of arguments they will output $\perp$. We write $\mathcal{E}_K(N, A, M) = \mathcal{E}(K, N, A, M)$ and similarly $\mathcal{D}_K(N, A, \mathbb{C}) = \mathcal{D}(K, N, A, \mathbb{C})$.

We assume that the message and associated data can be any binary string of arbitrary but finite length; i.e., $\mathcal{M} = \{0,1\}^*$ and $\mathcal{A} = \{0,1\}^*$, but the key and nonce are some fixed-length binary strings, i.e., $\mathcal{N} = \{0,1\}^{|N|}$ and $\mathcal{K} = \{0,1\}^k$, where the positive integers $|N|$ and k are respectively the nonce length and the key length of the scheme in bits. We assume that $|\mathcal{E}_K(N, A, M)| = |M| + \tau$ for some positive fixed constant τ; that is, we will have $\mathbb{C} = C||\mathsf{Tag}$ where $|C| = |M|$ and $|\mathsf{Tag}| = \tau$. We call C the core ciphertext and Tag the tag.

Remark *According to the CAESAR call for proposals "An authenticated cipher is a function with five byte-string inputs and one byte-string output. The five inputs are a variable-length plaintext, variable-length associated data, a fixed-length secret message number, a fixed-length public message number, and a fixed-length key. The output is a variable-length ciphertext." OMD considers the "public message number" as the nonce and does not support a secret message number.*

5.2.3.1 The OMD Mode of Operation

To use OMD one must specify a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ and a tag length $\tau \leq n$; where the key space $\mathcal{K} = \{0,1\}^k$ and $m \leq n$ where the case $m = n$ is the optimal choice from efficiency viewpoint. At first glance, requiring $m \leq n$ may look a bit odd as usually a compression function has a larger input block length than its output (hash) length, so we first explain this restriction based on the following two observations:

— It will be clear from the description of OMD in the sequel that at each call to the compression function only n random bits (namely, the output bits of the compression function) are available for encrypting an m-bit message block, hence we must have $m \leq n$. The optimal case is when $m = n$, so no random bits are wasted. We notice that this limitation applies to any compression function based AE, therefore a compression function based AE scheme (like OMD) will be usually less efficient than a block-cipher based AE (like OCB) "unless" one uses a dedicated compression function which is more efficient than the block-cipher.

— In practice, the compression function of standard hash functions (e.g. SHA-1 or the SHA-2 family) are keyless, i.e., do not have a dedicated key input, therefore one will need to use k bits of their b-bit message block to get a keyed function. So, there will be no efficiency waste in each call to the compression function if $m = n$ and $b = n + k$; for example, when the key length is 256 bits and the compression function of SHA-256 is used.

We let OMD-F denote the OCB mode of operation using a keyed compression function $F_K : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ with $m \leq n$ and an unspecified tag length. We let OMD$[F, \tau]$ denote the OMD mode of operation using keyed compression function F_K and tag length τ. The encryption algorithm of OMD$[F, \tau]$ inputs four arguments (secret key $K \in \{0,1\}^k$, nonce $N \in \{0,1\}^{|N|}$, associated data $A \in \{0,1\}^*$, message $M \in \{0,1\}^*$) and outputs $\mathbb{C} = C||\mathsf{Tag} \in \{0,1\}^{|M|+\tau}$. The decryption algorithm of OMD$[F, \tau]$ inputs four arguments (secret key $K \in \{0,1\}^k$, nonce $N \in \{0,1\}^{|N|}$, associated data $A \in \{0,1\}^*$, ciphertext $C||\mathsf{Tag} \in \{0,1\}^*$) and either outputs the whole $M \in \{0,1\}^{|C|-\tau}$ at once or an error message ($\perp$). Note that we have either $C = C_1 \cdots C_\ell$ or $C = C_1 \cdots C_{\ell-1} C_*$ depending on whether the message length in bits is a multiple of the block length m or not, respectively.

Fig. 5.5 depicts the construction of the encryption algorithm of OMD$[F, \tau]$. The construction of the decryption algorithm is straightforward and almost the same as the encryption algorithm except a tag comparison (verification) at the end of the decryption process. Fig. 5.6 describes the encryption and decryption algorithms of OMD$[F, \tau]$. We remind that for two binary strings $X = X_{m-1} \cdots X_0$ and $Y = Y_{n-1} \cdots Y_0$, the notation $X \oplus Y$ denotes bitwise XOR of $X_{m-1} \cdots X_{m-1-\ell}$ and $Y_{n-1} \cdots Y_{n-1-\ell}$ where $\ell = min\{m-1, n-1\}$.

COMPUTING THE MASKING VALUES. As seen from the description of OMD in Fig. 5.5, before each call to the underlying keyed compression function we XOR a masking value denoted as $\Delta_{N,i,j}$ (the top and middle parts of Fig. 5.5) and $\bar{\Delta}_{i,j}$ (the bottom part of Fig. 5.5). In the following, we describe how these masks are generated. We note that there are both security and efficient related criteria to be satisfied by the method to compute the masking values. We only explain the efficiency criterion for computing the masks here; the security related properties will be made clear in Section 5.2.2.4. By an efficient masking scheme, we mean a scheme in which the mask value needed for processing a block can be efficiently computed from the mask value used for processing the previous block.

There are different ways to compute the masking values to satisfy both the security and efficiency criteria; for example, we refer to [Rog04a, CS07, KR11]. We use the method proposed in [KR11].

In the following, all multiplications are in $GF(2^n)$, $\mathtt{ntz}(i)$ denotes the number of trailing zeros (i.e., the number of rightmost bits that are zero) in the binary representation of a positive integer i.

Initialization. $\Delta_{N,0,0} = F_K(N||10^{n-1-|N|}, 0^m)$; $\bar{\Delta}_{0,0} = 0^n$; $L_* = F_K(0^n, 0^m)$; $L[0] = 4.L_*$, and $L[i] = 2.L[i-1]$ for $i \geq 1$. We note that the values $L[i]$ can be preprocessed and stored (for a fast implementation) in a table for $0 \leq i \leq \lceil \log_2(\ell_{\max}) \rceil$, where $\ell_{\max}$ is the bound on the maximum number of blocks in any input that can be encrypted or decrypted. Alternatively, (if there is a memory restriction) they can be computed on-the-fly for $i \geq 1$. It is also possible to precompute and store some values and then compute the others as needed on-the-fly.

Masking sequence for processing the message. For $i \geq 1$: $\Delta_{N,i,0} = \Delta_{N,i-1,0} \oplus L[\mathtt{ntz}(i)]$; $\Delta_{N,i,1} = \Delta_{N,i,0} \oplus 2.L_*$; and $\Delta_{N,i,2} = \Delta_{N,i,0} \oplus 3.L_*$.

Masking sequence for processing the associate data. $\bar{\Delta}_{i,0} = \bar{\Delta}_{i-1,0} \oplus L[\mathtt{ntz}(i)]$ for $i \geq 1$; and $\bar{\Delta}_{i,1} = \bar{\Delta}_{i,0} \oplus L_*$ for $i \geq 0$.

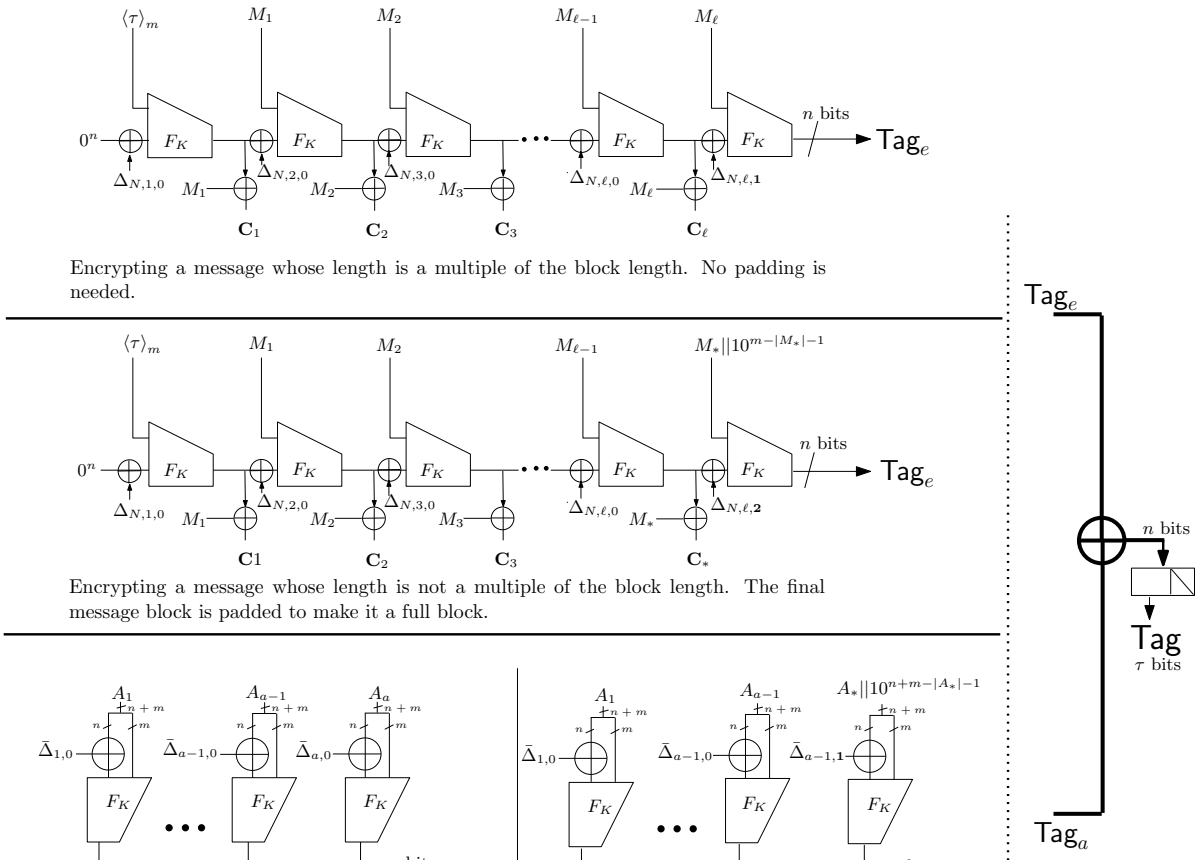
5.2.3.2 OMD-SHA256: Primary Recommendation for Instantiating OMD

Our primary recommendation to instantiate OMD is called OMD-SHA256 and uses the underlying compression function of SHA-256 [SHA95]. This is intended to be the appropriate choice for implementations on 32-bit machines. The compression function of SHA-256 is a map SHA-256 : $\{0,1\}^{256} \times \{0,1\}^{512} \rightarrow \{0,1\}^{256}$. On input a 256-bit chaining block X and a 512-bit message block Y, it outputs a 256-bit digest Z, i.e., let $Z = \text{SHA-256}(X, Y)$. The description of SHA-256 is provided in subsection B.1.

To use OMD with SHA-256, we use the first 256-bit argument X for chaining values as usual. In our notation (see Fig. 5.5) this means that $n = 256$. We use the 512-bit argument Y (the message block in SHA-256) to input both a 256-bit message block and the key K which can be of any length $k \leq 256$ bits. If $k < 256$ then let the key be $K||0^{256-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{256} \times \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ needed in OMD as $F_K(H, M) = \text{SHA-256}(H, K||0^{256-k}||M)$.

The parameters of OMD-SHA256 are as follows:

— The message block length in bits is $m = 256$, i.e., $|M_i| = 256$. *If needed*, we pad the final block of the message with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 256 bits.
— The key length in bits can be $80 \leq k \leq 256$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{256-k} to make its length exactly 256 bits.



Encrypting a message whose length is a multiple of the block length. No padding is needed.

Encrypting a message whose length is not a multiple of the block length. The final message block is padded to make it a full block.

Computing Tag_a for an associate data whose length is a multiple of the input lenght (i.e. $|A_a| = n + m$).

Computing Tag_a for an associate data whose length is not a multiple of the input length. The final block is padded to make it a full block.

Figure 5.5: The encryption process of OMD$[F, \tau]$ using a keyed compression function $F_K : (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ with $m \leq n$. (**Top**) The encryption process when the message length is a multiple of the block length m and no padding is required. (**Middle**) The encryption process when the message length is not a multiple of the block length and the final block M_* is padded to make a full block $M_*||10^{m-|M_*|-1}$. (**Bottom, Left**) Computing the intermediate value T_a when the bit length of the associated data is a multiple of the **input** length $n + m$. (**Bottom, Right**) Computing T_a when the bit length of the associated data is not a multiple of $n + m$ and the final block is padded to make a full block $A_*||10^{n+m-|A_*|-1}$ is needed. The output ciphertext is $C||\mathsf{Tag}$. For operation $\oplus$ see our convention in Section 5.2.2. Five types of key-dependent masking values (corresponding to five mutually exclusive tweak sets) are used; these are denoted by $\Delta_{N,i,0}, \Delta_{N,i,1}, \Delta_{N,i,2}, \bar{\Delta}_{i,0}$ and $\bar{\Delta}_{j,1}$, for $i \geq 1$ and $j \geq 0$, where N is the nonce. Note that the masks used in computing T_a do not depend on the nonce.

— The nonce (public message number) length in bits can be $96 \leq |N| \leq 255$. We *always* pad the nonce with $10^{255-|N|}$ to make its length exactly 256 bits.
— The secret message number length in bits is 0; that is, our scheme does not support secret message numbers.
— The associated data block length in bits is $2n = 512$, i.e., $|A_i| = 512$. *If needed*, we pad the final block of the associated data with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 512 bits.
— The tag length in bits can be $32 \leq \tau \leq 256$; but it must be noted that the selection of the tag length directly affects the achievable security level. We refer to Section 5.2.2.4 for the security bounds.

Algorithm INITIALIZE(K)
  $L_* \leftarrow F_K(0^n, 0^m)$
  $L[0] \leftarrow 4.L_*$ ▷ $2.(2.L_*)$, doubling in $GF(2^n)$
  for $i \leftarrow 1$ **to** $\lceil \log_2(\ell_{\max}) \rceil$ **do**
    $L[i] = 2.L[i-1]$ ▷ doubling in $GF(2^n)$
  end for
  return
end Algorithm

Algorithm $\mathrm{HASH}_K(A)$
  $b \leftarrow n + m$
  $A_1||A_2 \cdots A_{\ell-1}||A_\ell \xleftarrow{b} A$, where $|A_i| = b$ for $1 \le i \le \ell - 1$ and $|A_\ell| \le b$
  $\mathsf{Tag}_a \leftarrow 0^n$
  $\Delta \leftarrow 0^n$
  for $i \leftarrow 1$ **to** $\ell - 1$ **do**
    $\Delta \leftarrow \Delta \oplus L[\mathrm{ntz}(i)]$
    $\mathsf{Left} \leftarrow A_i[b-1 \cdots m]$; $\mathsf{Right} \leftarrow A_i[m-1 \cdots 0]$
    $\mathsf{Tag}_a \leftarrow \mathsf{Tag}_a \oplus F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
  end for
  if $|A_\ell| = b$ **then**
    $\Delta \leftarrow \Delta \oplus L[\mathrm{ntz}(\ell)]$
    $\mathsf{Left} \leftarrow A_\ell[b-1 \cdots m]$; $\mathsf{Right} \leftarrow A_\ell[m-1 \cdots 0]$
    $\mathsf{Tag}_a \leftarrow \mathsf{Tag}_a \oplus F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
  else
    $\Delta \leftarrow \Delta \oplus L_*$
    $\mathsf{Left} \leftarrow A_\ell||10^{b-|A_\ell|-1}[b-1 \cdots m]$
    $\mathsf{Right} \leftarrow A_\ell||10^{b-|A_\ell|-1}[m-1 \cdots 0]$
    $\mathsf{Tag}_a \leftarrow \mathsf{Tag}_a \oplus F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
  end if
  return Tag_a
end Algorithm

Algorithm $\mathcal{E}_K(N, A, M)$
  if $|N| > n - 1$ **then**
    return $\perp$
  end if
  $M_1||M_2 \cdots M_{\ell-1}||M_\ell \xleftarrow{m} M$, where $|M_i| = m$ for $1 \le i \le \ell - 1$ and $|M_\ell| \le m$
  $\Delta \leftarrow F_K(N||10^{n-1-|N|}, 0^m)$ ▷ initialize $\Delta_{N,0,0}$
  $H \leftarrow 0^n$
  $\Delta \leftarrow \Delta \oplus L[0]$ ▷ compute $\Delta_{N,1,0}$
  $H \leftarrow F_K(H \oplus \Delta, \langle \tau \rangle_m)$
  for $i \leftarrow 1$ **to** $\ell - 1$ **do**
    $C_i \leftarrow H \oplus M_i$
    $\Delta \leftarrow \Delta \oplus L[\mathrm{ntz}(i+1)]$
    $H \leftarrow F_K(H \oplus \Delta, M_i)$
  end for
  $C_\ell \leftarrow H \oplus M_\ell$
  if $|M_\ell| = m$ **then**
    $\Delta \leftarrow \Delta \oplus 2.L_*$
    $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
  else if $|M_\ell| \neq 0$ **then**
    $\Delta \leftarrow \Delta \oplus 3.L_*$
    $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell||10^{m-|M_\ell|-1})$
  else
    $\mathsf{Tag}_e \leftarrow H$
  end if
  $\mathsf{Tag}_a \leftarrow \mathrm{HASH}_K(A)$
  $\mathsf{Tag} \leftarrow (\mathsf{Tag}_e \oplus \mathsf{Tag}_a)[n-1 \cdots n-\tau]$
  $\mathbb{C} \leftarrow C_1||C_2||\cdots||C_\ell||\mathsf{Tag}$
  return $\mathbb{C}$
end Algorithm

Algorithm $\mathcal{D}_K(N, A, \mathbb{C})$
  if $|N| > n - 1$ **or** $|\mathbb{C}| < \tau$ **then**
    return $\perp$
  end if
  $C_1||C_2 \cdots C_{\ell-1}||C_\ell||\mathsf{Tag} \xleftarrow{m} \mathbb{C}$, where $|C_i| = m$ for $1 \le i \le \ell - 1$, $|C_\ell| \le m$ and $|\mathsf{Tag}| = \tau$
  $\Delta \leftarrow F_K(N||10^{n-1-|N|}, 0^m)$ ▷ initialize $\Delta_{N,0,0}$
  $H \leftarrow 0^n$
  $\Delta \leftarrow \Delta \oplus L[0]$ ▷ compute $\Delta_{N,1,0}$
  $H \leftarrow F_K(H \oplus \Delta, \langle \tau \rangle_m)$
  for $i \leftarrow 1$ **to** $\ell - 1$ **do**
    $M_i \leftarrow H \oplus C_i$
    $\Delta \leftarrow \Delta \oplus L[\mathrm{ntz}(i+1)]$
    $H \leftarrow F_K(H \oplus \Delta, M_i)$
  end for
  $M_\ell \leftarrow H \oplus C_\ell$
  if $|C_\ell| = m$ **then**
    $\Delta \leftarrow \Delta \oplus 2.L_*$
    $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
  else if $|C_\ell| \neq 0$ **then**
    $\Delta \leftarrow \Delta \oplus 3.L_*$
    $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell||10^{m-|M_\ell|-1})$
  else
    $\mathsf{Tag}_e \leftarrow H$
  end if
  $\mathsf{Tag}_a \leftarrow \mathrm{HASH}_K(A)$
  $\mathsf{Tag}' \leftarrow (\mathsf{Tag}_e \oplus \mathsf{Tag}_a)[n-1 \cdots n-\tau]$
  if $\mathsf{Tag}' = \mathsf{Tag}$ **then**
    return $M \leftarrow M_1||M_2||\cdots||M_\ell$
  else
    return $\perp$
  end if
end Algorithm

Figure 5.6: Definition of OMD$[F, \tau]$. The function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ is a keyed compression function with $\mathcal{K} = \{0,1\}^k$ and $m \le n$. The tag length is $\tau \in \{0, 1, \cdots, n\}$. Algorithms $\mathcal{E}$ and $\mathcal{D}$ can be called with arguments $K \in \mathcal{K}$, $N \in \{0,1\}^{\le n-1}$, and $A, M, \mathbb{C} \in \{0,1\}^*$. $\ell_{\max}$ is the bound on the maximum number of blocks in any input to the encryption or decryption algorithms.

5.2.3.3 OMD-SHA512: Secondary Recommendation for Instantiating OMD

Our secondary recommendation to instantiate OMD is called OMD-SHA512 and uses the underlying compression function of SHA-512 [SHA95]. This is intended to be the appropriate choice for implementations on 64-bit machines. The compression function of SHA-512 is a map SHA-512 : $\{0,1\}^{512} \times \{0,1\}^{1024} \to \{0,1\}^{512}$. On input a 512-bit chaining block X and a 1024-bit message block Y, it outputs a 512-bit digest Z, i.e., let $Z = \text{SHA-512}(X, Y)$. The description of SHA-512 is provided in subsection B.1.

To use OMD with SHA-512, we use the first 512-bit argument X for chaining values as usual. In our notation (see Fig. 5.5) this means that $n = 512$. We use the 1024-bit argument Y (the message block in SHA-512) to input both a 512-bit message block and the key K which can be of any length $k \leq 512$ bits. If $k < 512$ then let the key be $K||0^{512-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{512} \times \{0,1\}^{512} \rightarrow \{0,1\}^{512}$ needed in OMD as $F_K(H, M) = \text{SHA-512}(H, K||0^{512-k}||M)$.

The parameters of OMD-SHA512 are set as follows:

— The message block length in bits is $m = 512$, i.e., $|M_i| = 512$. *If needed*, we pad the final block of the message with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 512 bits.
— The key length in bits can be $80 \leq k \leq 512$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{512-k} to make its length exactly 512 bits.
— The nonce (public message number) length in bits can be $96 \leq |N| \leq 511$. We *always* pad the nonce with $10^{511-|N|}$ to make its length exactly 512 bits.
— The secret message number length in bits is 0; that is, our scheme does not support secret message numbers.
— The associated data block length in bits is $2n = 1024$, i.e., $|A_i| = 1024$. *If needed*, we pad the final block of the associated data with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 1024 bits.
— The tag length in bits can be $32 \leq \tau \leq 512$; but it must be noted that the selection of the tag length directly affects the achievable security level. We refer to Section 5.2.2.4 for the security bounds.

5.2.3.4 Compression Functions of SHA-256 and SHA-512

The CAESAR call for submissions has mentioned that "The cipher definition is required to be self-contained, including all information necessary to implement the cipher from scratch, except that the following functions are free to be used without being defined: AES-128 with 128-bit key, 128-bit input, and 128-bit output; AES-192 with 192-bit key, 128-bit input, and 128-bit output; AES-256 with 256-bit key, 128-bit input, and 128-bit output."

Therefore, in this section we include a description of the compression functions of the standard SHA-256 and SHA-512 hash functions from NIST FIPS PUB 180-4 [SHA95]. We refer to the underlying compression functions of these standard hash functions as SHA-256 and SHA-512, respectively.

Preliminaries. In the following, by "word" we mean a group of either 32 bits (4 bytes) or 64 bits (8 bytes), depending on the compression function algorithm. Namely, in SHA-256 each word is a 32-bit string and in SHA-512 each word is a 64-bit string.

$\mathbf{ROTR^n(x)}$: The *rotate right* (circular right shift) operation, where x is a w-bit word and n an integer with $0 \leq n < w$, is defined by $\text{ROTR}^n(x) = (x \gg n) \vee (x \ll w - n)$

$\mathbf{SHR^n(x)}$: The *right shift* operation, where x is a w-bit word and n an integer with $0 \leq n < w$, is defined by $\text{SHR}^n(x) = (x \gg n)$.

The addition $x+y$ of two w-bit words x and y is defined as follows. The words x and y represent integers X and Y, where $0 \leq X < 2^w$ and $0 \leq Y < 2^w$. Compute $Z = (X + Y) \bmod 2^w$. Then $0 \leq Z < 2^w$. Convert the integer Z to a word z and define $z = x + y$.

The SHA-256 Compression Function. SHA-256 uses six logical functions, where each function operates on 32-bit words, which are represented as x, y, and z and outputs a 32-bit word as a result. These functions are defined as follows:

$$Ch : \{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}, \qquad Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj : \{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}, \quad Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\Sigma_0^{\{256\}} : \{0,1\}^{32} \to \{0,1\}^{32}, \quad \Sigma_0^{\{256\}}(x) = \text{ROTR}^2(x) \oplus \text{ROTR}^{13}(x) \oplus \text{ROTR}^{22}(x)$$
$$\Sigma_1^{\{256\}} : \{0,1\}^{32} \to \{0,1\}^{32}, \quad \Sigma_1^{\{256\}}(x) = \text{ROTR}^6(x) \oplus \text{ROTR}^{11}(x) \oplus \text{ROTR}^{25}(x)$$
$$\sigma_0^{\{256\}} : \{0,1\}^{32} \to \{0,1\}^{32}, \quad \sigma_0^{\{256\}}(x) = \text{ROTR}^7(x) \oplus \text{ROTR}^{18}(x) \oplus \text{SHR}^3(x)$$
$$\sigma_1^{\{256\}} : \{0,1\}^{32} \to \{0,1\}^{32}, \quad \sigma_1^{\{256\}}(x) = \text{ROTR}^{17}(x) \oplus \text{ROTR}^{19}(x) \oplus \text{SHR}^{10}(x)$$

During the process of compression, a sequence of 64 constant 32-bit words, $K_0^{\{256\}}, ..., K_{63}^{\{256\}}$ are used. These 32-bit words represent the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers. In hex, these constant words are (from left to right):

```
428a2f98  71374491  b5c0fbcf  e9b5dba5  3956c25b  59f111f1  923f82a4  ab1c5ed5
d807aa98  12835b01  243185be  550c7dc3  72be5d74  80deb1fe  9bdc06a7  c19bf174
e49b69c1  efbe4786  0fc19dc6  240ca1cc  2de92c6f  4a7484aa  5cb0a9dc  76f988da
983e5152  a831c66d  b00327c8  bf597fc7  c6e00bf3  d5a79147  06ca6351  14292967
27b70a85  2e1b2138  4d2c6dfc  53380d13  650a7354  766a0abb  81c2c92e  92722c85
a2bfe8a1  a81a664b  c24b8b70  c76c51a3  d192e819  d6990624  f40e3585  106aa070
19a4c116  1e376c08  2748774c  34b0bcb5  391c0cb3  4ed8aa4a  5b9cca4f  682e6ff3
748f82ee  78a5636f  84c87814  8cc70208  90befffa  a4506ceb  bef9a3f7  c67178f2
```

The Compression Process. The SHA-256 compression function is defined as follows:

$$\text{SHA-256} : \{0,1\}^{256} \times \{0,1\}^{512} \longrightarrow \{0,1\}^{256}, \qquad \text{SHA-256}(H,M) = C$$

Let H be the 256-bit *hash input* (chaining input) and M be the 512-bit *message input*. These two inputs are represented respectively by an array of 8 32-bit words $H_0 \cdots H_7$ and an array of 16 32-bit words $M_0 \cdots M_{15}$. The 256-bit output value C is also represented as an array of 8 32-bit words $C_0 \cdots C_7$.

The compression function processes as below:

1. Prepare the message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

2. Initialize the eight working variables, a, b, c, d, e, f, g and h with the hash input value H:

$a = H_0$
$b = H_1$
$c = H_2$
$d = H_3$
$e = H_4$
$f = H_5$
$g = H_6$
$h = H_7$

3. For $t = 0$ to 63, do:

{

$$
\begin{aligned}
T_1 &= h + \Sigma_1^{\{256\}}(e) + \mathsf{Ch}(e,f,g) + K_t^{\{256\}} + W_t \\
T_2 &= \Sigma_0^{\{256\}}(a) + \mathsf{Maj}(a,b,c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

}

4. Compute the 256-bit output (hash) value $C = C_0 \cdots C_7$ as:

$$
\begin{aligned}
C_0 &= a + H_0 \\
C_1 &= b + H_1 \\
C_2 &= c + H_2 \\
C_3 &= d + H_3 \\
C_4 &= e + H_4 \\
C_5 &= f + H_5 \\
C_6 &= g + H_6 \\
C_7 &= h + H_7
\end{aligned}
$$

The SHA-512 Compression Function. SHA-512 uses six logical functions, where each function operates on 64-bit words, which are represented as x, y, and z and outputs a 64-bit word as a result. These functions are defined as follows:

$$
\begin{aligned}
Ch &: \{0,1\}^{64} \times \{0,1\}^{64} \times \{0,1\}^{64} \rightarrow \{0,1\}^{64}, \quad Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) \\
Maj &: \{0,1\}^{64} \times \{0,1\}^{64} \times \{0,1\}^{64} \rightarrow \{0,1\}^{64}, \quad Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_0^{\{512\}} &: \{0,1\}^{64} \rightarrow \{0,1\}^{64}, \quad \Sigma_0^{\{256\}}(x) = \text{ROTR}^{28}(x) \oplus \text{ROTR}^{34}(x) \oplus \text{ROTR}^{39}(x) \\
\Sigma_1^{\{512\}} &: \{0,1\}^{64} \rightarrow \{0,1\}^{64}, \quad \Sigma_1^{\{256\}}(x) = \text{ROTR}^{14}(x) \oplus \text{ROTR}^{18}(x) \oplus \text{ROTR}^{41}(x) \\
\sigma_0^{\{512\}} &: \{0,1\}^{64} \rightarrow \{0,1\}^{64}, \quad \sigma_0^{\{256\}}(x) = \text{ROTR}^{1}(x) \oplus \text{ROTR}^{8}(x) \oplus \text{SHR}^{7}(x) \\
\sigma_1^{\{512\}} &: \{0,1\}^{64} \rightarrow \{0,1\}^{64}, \quad \sigma_1^{\{256\}}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^{6}(x)
\end{aligned}
$$

During the process of compression, a sequence of 80 constant 64-bit words $K_0^{\{512\}}, ..., K_{79}^{\{512\}}$ is used. These 64-bit words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. In hex, these constant words are (from left to right):

```
428a2f98d728ae22  7137449123ef65cd  b5c0fbcfec4d3b2f  e9b5dba58189dbbc
3956c25bf348b538  59f111f1b605d019  923f82a4af194f9b  ab1c5ed5da6d8118
d807aa98a3030242  12835b0145706fbe  243185be4ee4b28c  550c7dc3d5ffb4e2
72be5d74f27b896f  80deb1fe3b1696b1  9bdc06a725c71235  c19bf174cf692694
e49b69c19ef14ad2  efbe4786384f25e3  0fc19dc68b8cd5b5  240ca1cc77ac9c65
2de92c6f592b0275  4a7484aa6ea6e483  5cb0a9dcbd41fbd4  76f988da831153b5
983e5152ee66dfab  a831c66d2db43210  b00327c898fb213f  bf597fc7beef0ee4
c6e00bf33da88fc2  d5a79147930aa725  06ca6351e003826f  142929670a0e6e70
27b70a8546d22ffc  2e1b21385c26c926  4d2c6dfc5ac42aed  53380d139d95b3df
650a73548baf63de  766a0abb3c77b2a8  81c2c92e47edaee6  92722c851482353b
a2bfe8a14cf10364  a81a664bbc423001  c24b8b70d0f89791  c76c51a30654be30
d192e819d6ef5218  d69906245565a910  f40e35855771202a  106aa07032bbd1b8
19a4c116b8d2d0c8  1e376c085141ab53  2748774cdf8eeb99  34b0bcb5e19b48a8
391c0cb3c5c95a63  4ed8aa4ae3418acb  5b9cca4f7763e373  682e6ff3d6b2b8a3
748f82ee5defb2fc  78a5636f43172f60  84c87814a1f0ab72  8cc702081a6439ec
90befffa23631e28  a4506cebde82bde9  bef9a3f7b2c67915  c67178f2e372532b
ca273eceea26619c  d186b8c721c0c207  eada7dd6cde0eb1e  f57d4f7fee6ed178
06f067aa72176fba  0a637dc5a2c898a6  113f9804bef90dae  1b710b35131c471b
28db77f523047d84  32caab7b40c72493  3c9ebe0a15c9bebc  431d67c49c100d4c
4cc5d4becb3e42b6  597f299cfc657e2a  5fcb6fab3ad6faec  6c44198c4a475817
```

The Compression Process. The SHA-512 compression function is defined as follows:

$$\text{SHA-512} : \{0,1\}^{512} \times \{0,1\}^{1024} \longrightarrow \{0,1\}^{512}, \qquad \text{SHA-512}(H, M) = C$$

Let H be the 512-bit *hash input* (chaining input) and M be the 1024-bit *message input*. These two inputs are represented respectively by an array of 8 64-bit words $H_0 \cdots H_7$ and an array of 16 64-bit words $M_0 \cdots M_{15}$. The 512-bit output value C is also represented as an array of 8 64-bit words $C_0 \cdots C_7$.

The compression function processes as described below:

1. Preparing the message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t & 0 \le t \le 15 \\ \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases}$$

2. Initialize the eight working variables, a, b, c, d, e, f, g and h with the hash input value H:

$$\begin{aligned} a &= H_0 \\ b &= H_1 \\ c &= H_2 \\ d &= H_3 \\ e &= H_4 \\ f &= H_5 \\ g &= H_6 \\ h &= H_7 \end{aligned}$$

3. For $t = 0$ to 79, do:

{

$$
\begin{aligned}
T_1 &= h + \Sigma_1^{\{512\}}(e) + Ch(e,f,g) + K_t^{\{512\}} + W_t \\
T_2 &= \Sigma_0^{\{512\}}(a) + Maj(a,b,c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

}

4. Computing the 512-bit output (hash) value $C = C_0 \cdots C_7$ as:

$$
\begin{aligned}
C_0 &= a + H_0 \\
C_1 &= b + H_1 \\
C_2 &= c + H_2 \\
C_3 &= d + H_3 \\
C_4 &= e + H_4 \\
C_5 &= f + H_5 \\
C_6 &= g + H_6 \\
C_7 &= h + H_7
\end{aligned}
$$

CHAPTER 6

CONCLUSION

In the quite recent past, we have experienced a new era of connectivity: devices have become smaller and cheaper, and pretty much every single chip has the capacity of connecting to another device or to the Internet. Sensitive data and its computation, which were first centralized to a protected environment inside the chip, now are routed and transmitted over several different protocols, via wired or wireless links, to remote databases. This has started a revolution in security implementations and protocols: how to protect privacy, integrity and authentication over this fuzz of connected smart devices.

The concept called the "Internet of Things" (IoT) is becoming a hot topic in media and within tech companies. A broad definition of IoT is the interconnected network of physical devices such as cars, objects, buildings and other smart systems by means of electronics, sensors, software and network protocols that enable these components to collect and exchange information. IoT brings connectivity and data to smart objects, which pervasively monitor the environment and act accordingly. Ultimately, IoT integrates the physical world into computer-based systems, improving efficiency, accuracy, and bringing economic benefits to the processes involved.

In this context, new application areas emerge, pushing the limits of hardware and software systems to generate, index, store and process larger amounts of data from diverse locations, with the consequently demand of higher performance of transmission protocols. In conjunction with cloud computing, IoT continually demands hardware systems to be smaller, faster and smarter. Cryptographic designs face several tradeoffs, as depicted in Fig. 6.1.

An IoT computing environment is also generically referred to as a *smart object*. Smart objects interact with the environment and with humans, sense the environment, take actions based on stimuli and communicate with other smart objects to fulfill their application. However these devices are constrained and have limited hardware resources, making them vulnerable. Each IoT node is a smart object that can potentially compromise the whole network. Should one of these nodes fails and dies, the network as whole must continue to work properly. If one of these nodes is attacked, the security of the whole network might be compromised. Typically, smart objects do not have enough resources to handle complex cryptographic primitives; they must implement lightweight versions instead. *Lightweight cryptography* (LWC) is a cryptographic algorithm or protocol tailored for implementation in constrained environments including RFID tags, sensors, contactless smart cards, health-care devices and many more. These are ultimately nodes of the IoT network, the smart objects.

Two main characteristics of lightweight cryptography that make it suitable for IoT are:

Efficiency of end-to-end communication. To achieve end-to-end security, end nodes have an implementation of a symmetric key algorithm. For the low resource-devices, e.g. battery-powered devices, the cryptographic operation with a limited amount of energy consumption is important. Application of the lightweight symmetric key algorithm allows reducing the energy consumption for end devices.

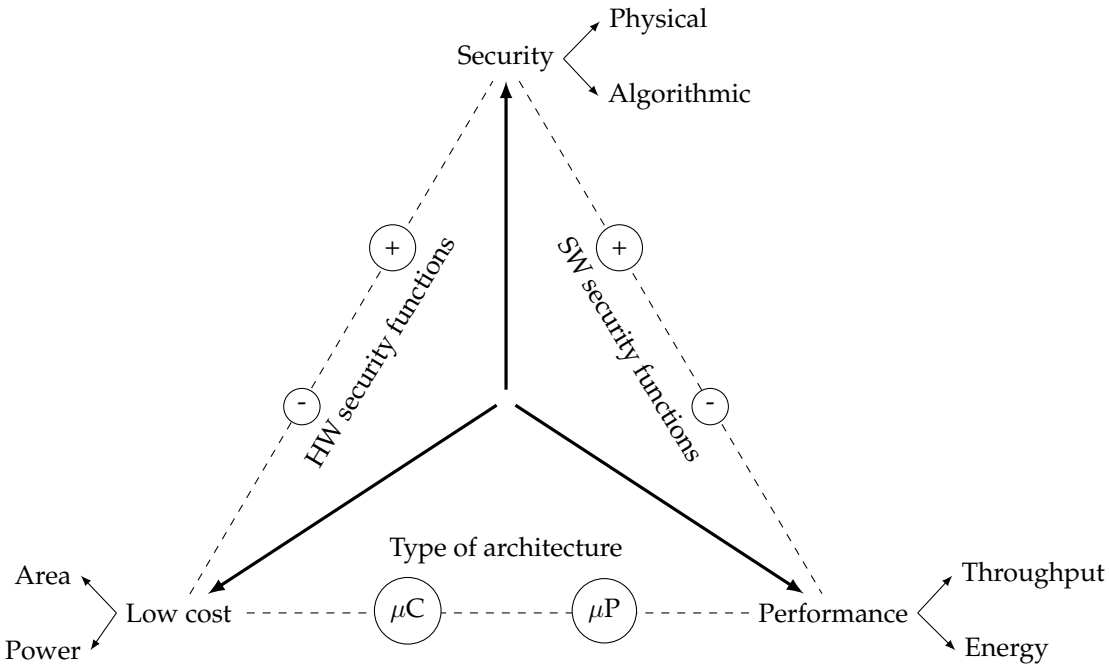


Figure 6.1: Tradeoffs in cryptography.

Applicability to lower resource devices. The footprint of the lightweight cryptographic primitives is smaller than the conventional cryptographic ones. The lightweight cryptographic primitives would open possibilities of more network connections with lower resource devices.

An important discernment in modern cryptography is to divide the algorithm's security analysis in two parts. Initially, cryptanalysts analyze the cryptographic algorithm, protocol or construction with ideal primitives. This initial steps has the goal of finding security flaws inherent to the algorithm's mathematical construction. After this first validation by means of mathematical proofs, a second analysis is conducted. This one has the objective of finding security flaws on actual implementations of cryptographic algorithms. At this second step, side-channel analysis is a very powerful tool used by cryptanalysts to find weaknesses.

To evaluate the efficiency of hardware-based cryptosystem implementations, we must rely on efficiency metrics. These metrics can be, for example, the number of cipher rounds, the number of modular exponentiations, the size of the substitution table, or the complexity of the mathematical operations required to perform one computation round. Care must be taken when choosing these metrics, as they are not always straightforward to interpret. Certain designs must be more efficient for long messages, although systems using LWC usually have shorter message lengths. Some cryptographic constructions rely on nonces (numbers used only once) for security reasons, which could be prohibitive on low-end systems. A cryptosystem relying on a nonce must instantiate a non-volatile memory to store a counter or a hardware source of randomness. Metrics such as number of primitive calls per block and the possibility to apply parallelism are likely important in constrained cryptosystems that apply LWC. Ultimately, the power consumption metric is a very important metric to evaluate how lightweight a cryptosystem is.

Perhaps most importantly, we should point out that lightweight cryptography is intended for applications with very stringent requirements that conventional algorithms fail to satisfy. As such, we focus specifically at applications where cryptography is the bottleneck. Conventional algorithms were of course never designed to be inefficient, but typically aim to have good performance on a very wide range of platforms. For lightweight cryptography, it seems that the goal is then to focus on algorithms that are tailored to more specific use cases, at the cost of having a more narrow range of applicability.

Two important and well studied block ciphers designed to be lightweight are CLEFIA [SSA+07] and PRESENT [BKL+07]. Both algorithms are considered suitable for lightweight cryptography under ISO/IEC 29192 certification. In the hash function world, QUARK [AHMN13] is one example of a cryptographic hash function designed to be lightweight. Recent cryptographic competitions such as the

SHA-3 [Nat12] and CAESAR competitions [CAE] aim at designing trusted cryptography for standardization that is fast, efficient, and resists the latest advances in cryptanalysis.

Lightweight implementations. BCH codes features a large class of error-correcting codes and are widely adopted in the industry. 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); the BCH(511, 493) was adopted by the International Telecommunication Union as a standard for video conferencing and video phone codecs (Rec. H.26). Section 3.1 presented a faster BCH implementation that makes use of a public-key cryptography technique. Because BCH codes require repeated polynomial reductions modulo a constant polynomial, it is possible to apply the Barrett's modular reduction algorithm to achieve better performance.

Five BCH(15, 7, 2) implementations were used in order to evaluate the performance of different approaches:

— `BCH-standard`
— `BCH-LFSR`
— `BCH-LFSR-improved`
— `BCH-Barrett`
— `BCH-Barrett-pipelined`

This study showed that the `BCH-Barrett` implementation presented comparable area size when compared to the LFSR options, typically used in BCH implementations for its very small area overhead. Apart from that, the `BCH-Barrett` architecture is also more suitable for lightweight hardware designs because results showed much lower power consumption (around 45% reduction) when compared to LFSR-based architectures (`BCH-LFSR` and `BCH-LFSR-improved`). Moreover, `BCH-Barrett` was approximately 9 and 4 times faster than `BCH-LFSR` and `BCH-LFSR-improved`, respectively.

Consequently, this thesis presented BCH design techniques that highlight the importance of finding new ways of transposing known techniques from an application field to another. Commercial BCH hardware cores are mostly implemented using LFSR-based approaches. Well known to security hardware engineers and cryptographers, BCH codes leverages the performance of Barrett's modular reduction. Lightweight hardware implementations are ultimately about applying new ideas to push their own limits.

When security designers implement standard cryptographic algorithms, especially in hardware, they have to keep in mind the hardware constraints and environment limitations in which the cryptosystems will be integrated to. Therefore, whenever possible, designers push the limits of EDA tools in search of the best implementation. New techniques and novel ways of bringing lightweight implementations to real life problems are constantly needed.

Spatially distributed nodes form the networks that enable the introduction of the Internet of things. Consequently, wireless networks must be studied in terms of security and reliability. As already stated, 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 and communication interference.

Section 5.1 introduces an authentication protocol based on the zero-knowledge paradigm that establishes network integrity, and leverages the distributed nature of computing nodes to alleviate individual computational effort. This enables the base station to identify nodes needing replacement or attention. In this work we described 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.

Power minimization. Smart objects connected to an IoT infrastructure are typically limited in hardware resources. Mobile devices and smart sensors are also constrained in power. A right balance between data processing, computational power and energy resources is therefore crucial for these devices.

Section 3.2 presented a smart energy management system capable of turning on and off an SoC or an embedded system according to a proposed model. This model assumes that the system has two possible states, A and B, in which the system is available to respond to new requests or is in idle mode, respectively.

The importance of having such smart energy manager is to endow that the system can go idle when the incoming request probability is low enough to take such risk. While reducing unresponsiveness penalties, our model ensures a better power usage, which translates to a better battery lifetime. Interestingly our proposed strategy does not depend on the energy cost necessary to process an incoming request, which makes it clearly optimal since we show that at any time the model tends to minimize the increase in the power consumption function.

While energy harvesting technologies and remote power generation are still not a reality for sensors and embedded systems, smart energy policies ensure the success of deploying the IoT everyday devices.

Hardware countermeasures. Before being standardized, cryptographic algorithms are extensively studied by the academic community, where cryptanalysts test theoretical attacks against cryptographic constructions. In addition to the mathematical analysis of cryptographic algorithms, cryptanalysts also study and apply side-channel attacks that exploit weaknesses in physical implementations.

Section 4.4 presented a lightweight side-channel protection for a proposed AES implementation. Two physical threats were analyzed: power and fault attacks. The proposed architecture leveraged the AES algorithms structure to create low-cost protections against these attacks. This allowed very flexible runtime configurability without significantly affecting performance.

First, the unprotected AES implementation was sliced into four clock cycles per round. Making use of this approach, we built on top of the unprotected core two power scrambling ideas to thwart side-channel attacks, such as CPA. We also demonstrated how the design can also prevent fault injection by recomputing its internal *state* values or by sacrificing one out of four blocks at each clock to compute the encryption of a known plaintext. We then exhibited simulation results and compared the unprotected core with the protected core. The results confirm that the overhead in terms of area, power and performance is small, making this countermeasure attractive.

Moreover, the proposed AES architecture provides different options to tune the design into the user's need. Among 29 different configurations, examples include: turn the proposed AES into a 4-stage pipeline (i.e., compute four different plaintexts per execution), use three blocks to generate noise against power attacks, or to use one inactive block in the chain to recompute encryption and ascertain its correctness. In addition to the proposed AES implementation, we presented a simple scheme halving the number of memory cells required for storing *subkeys* during AES decryption.

Applying cryptography knowledge to real case scenarios where security threats are also a concern is another topic addressed in this thesis. Section 4.5 presented a hardware protection scheme specifically applied to system-on-chips. In this context, attacks are divided into three main categories:

– **Hijacking** writing to restricted addresses to change the system's configuration;

– **Extraction of secret information** reading from secure addresses to retrieve sensitive data;

– **Denial of Service** reducing the system's throughput by replaying or forging request over the NoC.

If an attacker can successfully glitch a packet header or impersonate an initiator (by changing the initiator ID of a packet request to the firewall, for example), the attacker will be able to read from or write to a protected memory area. Although potentially dangerous, this type of attack is a single-event exploit and is rather limited. Typically, the request path and the response path are two physically different buses in the interconnect, so the packet response will be routed to the initiator that was impersonated, instead of being routed to the malicious initiator. Emerging solutions to maintain the reliability and the integrity of the request path will also make successful modification of an in-flight request harder.

However, escalating privileges in the access control block or firewalls presents a much higher threat. For instance, this access can be achieved by impersonating the reprogramming agent or by glitching the

bus during reprogramming to load unintended rules. If attackers can modify access policies for a given resource, they can gain permanent access and therefore read or modify content at will, such as digital rights managements (DRM) content, banking or personal information. Recent work demonstrated how a malicious SoC hardware IP could compromise critical data [LG14], and how Internet of Things (IoT) devices could not be trusted with secure applications if firewalling and partitioning is not maintained properly.

Our work introduced the Cryptographically Secure Access Control (CSAC) as a security layer over the existing network-on-chip hardware management of virtualization of secure/non-secure environments. CSAC implements two security features. First, it cryptographically authenticates the reprogramming agent and ensures the integrity of its reprogramming sequences. Second, CSAC ensures the integrity of the access policies over the SoC by checking and hashing rules per access. The CSAC engine is based on a Keyed-Hash Message Authentication Code (HMAC) [Nat08], where the key is shared between the reprogramming agent and CSAC core. This key is programmed for each session and the programming is part of the hardware root-of-trust of the SoC. CSAC supports both hardware and software key delivery, which are performed at secure boot of the SoC.

CSAC was synthesized in five different versions using a digital library with technology node of $45nm$. The synthesis results were obtained with Cadence Encounter RTL Compiler v13.10. The results show the impact of including security features in the CSAC core. Each version was split into two firewall settings: one composed of four regions providing access rights to six initiators and address a space of 4GB; the other enabling the use of 14 initiators and covering a target address space of 64GB, with a total of 8 protection regions. These results focused on comparing the cost of the enforcement logic and authentication engine.

Results showed a high overhead, specially in power consumption, which can be restrictive for IoT deployment. By using two clock domains one for configuration, the other for firewalling CSAC could speed up the firewall logic and avoid NoC clock frequency loss due to security. Since the authentication engine is only used when a new protection region is (re)programmed in CSAC, it does not impact the firewall enforcement logic in a two-clock domain scheme. As a future work in this direction, a full implementation of this two-clock architecture remains to be evaluated.

Yet another contribution of this thesis regarding hardware protection against side-channel attacks is the introduction of Correlation Instantaneous Frequency Analysis (CIFA), presented in Section 4.6. This works investigated the use of instantaneous frequency instead of power amplitude and power spectrum in side-channel analysis. By opposition to the constant frequency used in Fourier Transform, instantaneous frequency reflects local phase differences and allows the detecting of frequency variations. These variations depend on the processed binary data and are hence cryptanalytically useful. The relationship stems from the fact that after higher power drops more time is required to restore power back to its nominal value.

Energy consumed during each clock cycle creates a waveform in the power domain. A duty cycle, i.e., the time during which the power wave is unequal to its nominal value, can be considered as the execution time of a hardware implemented algorithm. The duty cycle may depend on the processed data. Fourier transform can not determine local duty cycles since frequency is defined for the sine or cosine function spanning the whole data length with constant period and amplitude. However, recent techniques can detect local frequencies and hence determine the wave's duty cycle. We showed that, in addition to the signal's amplitude and spectrum, traditionally used for side-channel analysis, instantaneous frequency variations may also leak secret data. To the authors' best knowledge, "pure" frequency leakage has not been considered as a side-channel vector so far. Hence a re-assessment of several countermeasures, especially, these based on amplitude alterations, seems in order. Our work also examined DVS, which makes AES implementation impervious to power and spectrum attacks while leaving it still vulnerable to CIFA.

IF analysis does not bring specific benefits when applied to unprotected designs on which CPA and CSBA yield better results. However, CIFA allows to discard the effect of amplitude modification countermeasures, *e.g.* DVS, because CIFA extracts from signal features are not exploited so far.

Authenticated encryption. Section 5.2 described our proposal of a new authenticated cipher for consideration in the CAESAR competition. Our scheme, called Offset Merkle-Damgård (OMD), is a keyed compression function mode of operation for nonce-based AEAD. The syntax and security notions for nonce-based AEAD schemes were formalized by Rogaway in [Rog02, Rog04b]. To instantiate the OMD mode, we recommend two specific compression functions to be keyed and used in OMD, namely, the compression functions of the standard SHA-256 and SHA-512 hash functions. OMD parameterized with these two compression functions is called OMD-SHA256 and OMD-SHA512, respectively. The former is intended for 32-bit implementations and is our primary recommended algorithm, while the latter could be used specifically for 64-bit machines and is our secondary algorithm.

We believe that an AE scheme whose security is proved by a modular and easy to verify security reduction, only relying on some widely-verified standard assumption(s) on its underlying primitive(s), can get more confidence on its security compared to a scheme that demands strong and idealistic properties from its underlying primitive(s) or is not supported by a formal security proof. Provable security helps cryptanalysis efforts to be focused on analyzing the simpler underlying primitives rather than the whole scheme; hence, building confidence in the security of the scheme becomes easier if the cryptosystem uses well-analyzed and verified primitives.

The main features that make OMD a very strong candidate for the CAESAR competition are:

– **Using only a single well-known primitive.** OMD is designed as a mode of operation for a keyed compression function. Together with block ciphers and permutations, compression functions are among the most well-known and widely used symmetric key primitives. We have a rich source of secure compression functions thanks to more than two decades of public research and standardization activities of hash functions.

– **Provable security based on a single widely-accepted standard assumption.** The security goals of privacy and authenticity for OMD are achieved provably in the sense of reduction-based cryptography; that is, any attack against these security goals will imply an attack against the classical PRF property of the underlying compression function. We note that any keyed compression function (either a dedicated-key one or keyed via some part of its input) must provide the classical PRF property when its key is secret as otherwise it will be considered useless for any secret key application, e.g. for being used as a MAC. That is, the base PRF assumption on the compression function upon which the security of OMD relies is highly assured for compression functions of the practical, standard hash functions, thanks to the vast amount of cryptanalytic work on these functions.

– **Requiring minimal basic operations in addition to the core primitive.** The only operations that OMD needs *in addition to* its core compression function are the basic operations of bitwise XORing two binary strings and shifting a binary string.

– **Integrated (one-pass) AEAD scheme.** In OMD the mechanisms for providing privacy and authenticity of the message are coupled in a single pass of (a variant of) the Merkle-Damgård iteration of the compression function. This is aimed to make OMD as much efficient as possible (up to the limits that are inherent to any compression function based AEAD scheme).

– **Online Encryption.** OMD encryption is online; that is, it outputs a ciphertext stream as a plaintext stream arrives with a constant latency and using constant memory. After receiving an indication that the plaintext is over, the final part of ciphertext together with the tag is output.

– **Internally Online Decryption.** OMD decryption is *internally* online: one can generate a stream of plaintext bits as the stream of ciphertext bits comes in, but no part of the plaintext stream will be revealed before the whole ciphertext stream is decrypted and the tag is verified to be correct. That is, nothing about the decrypted plaintext should be made available to adversaries if the tag is incorrect signifying that the queried ciphertext is invalid.

– **Flexible key size.** OMD-SHA256 can support any key length between 80 bits and 256 bits. This will be useful for applications requiring unconventional key lengths, e.g. 96-bit keys.

– **Efficient.** If implemented with a SHA family member, OMD can take advantage of the newly introduced Intel® instructions that support performance acceleration of the Secure Hash Algorithm

(SHA) on Intel® Architecture processors. In particular, our main recommended scheme for CAESAR, called OMD-SHA256, is aimed to get the most out of these new performance accelerating instructions.

– **Resistance against software-level timing attacks.** Most AES software implementations risk leaking their keys through cache timing [Ber05] unless they are implemented on machines with Intel® CPUs supporting the constant-time AES-NI and PCLMULQDQ instructions. In comparison, we note that the only operations in OMD-SHA256 are: bitwise XOR, AND and OR of two binary strings (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), fixed-distance (left and right) shift of a binary string (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), and 32-bit addition (of words in the compression function of SHA-256). These operations have the virtue of taking constant time on typical CPUs in which case the implementations can avoid software-level timing based side-channel leaks.

The main design rationales behind OMD are the following:

– **Provable security.** We aimed to have a scheme with a sound security guarantee in the style of reduction-based provable security relying only on a single well-established standard assumption on the underlying primitive, namely the PRF assumption on the keyed compression function. The security goals of privacy and authenticity for OMD are achieved provably; that is, any attack against these security goals will imply an attack against the classical PRF property of the underlying compression function. We note that any good keyed compression function (either a dedicated-key one or keyed via some part of its input) must provide the classical PRF property when its key is secret as otherwise it will be considered useless for almost any secret key application, e.g. for being used as the compression function of a hash function in the standard HMAC algorithm. That is, the base PRF assumption on the compression function upon which the security of OMD relies is highly assured for compression functions of the practical, standard hash functions, thanks to the vast amount of cryptanalytic work on these functions.

– **Simple structure.** Simplicity is important in any cryptographic algorithm: the easier an algorithm is to understand, the easier it is to analyze and to get confidence on its security, and also less prone it is to implementation errors. Therefore, simplicity was one of our core design goals. OMD's high level structure is quite simple and resembles the well-known structures for hash functions and MACs, namely, the part that is processing the message resembles the Merkle-Damgård iteration where at each iteration random bits are derived from the chaining values to be used for encryption and a key-dependent offset value is XORed to the chaining values. The part for processing the associated data is inspired by the XMACC scheme (counter-based XOR MAC scheme) [BGR95] and is a simple adaptation of the similar hashing process in the OCB3 algorithm [KR11]. We note that when the message is empty then OMD acts almost the same as XMACC on the associated data.

– **No trapdoor.** The designers have not hidden any weaknesses in this cipher. Any attack against security of OMD means an attack against the specific compression function that is used for instantiating OMD. For example, attacking OMD-SHA256 will imply attacking the compression function of SHA-256 in the PRF sense.

Final thoughts. There exists a huge gap between the scientific state of the art in cryptography and commercial cryptosystems. Outdated cryptographic protocols and primitives are still used nowadays despite all the proven attacks against these implementations. Companies are usually reluctant to change their cryptosystems for several reasons, of which cost is certainly the most important. Deployed cryptosystems that are no longer considered secure and pose a serious threat to users are continually maintained, and the responsible actors claim that *legacy application* is a good enough reason for that.

As a result, only a small fraction of recent research in cryptography is really used in practice; and the deployed cryptosystems lack implementation care and performance measures (and often lack of proper parameter selection).

Efforts in cryptology must be taken to ensure that future cryptography developments are well understood and implemented correctly and in a reasonable time frame. Advancements in lightweight cryptosystems will allow the deploying of more and more pervasive networks, ultimately allowing the broad concept of the Internet of things.

Therefore, novel ideas to improve cryptosystem performances like the ones presented in this thesis are always welcome, as they enable the evolution of electronic security and privacy.

INDEX

3D, 116
3DES, 51

time-memory tradeoff attack, 60

access control, 103–106, 157
AddRoundKey, 51, 53, 55, 93–97, 99, 100
adversary, 18, 130–132, 137–143
AE, 133, 138, 145, 159
AEAD, 22, 25, 133, 137, 138, 144, 145, 159
AES, 18, 25, 50–56, 65, 93, 94, 96–101, 112, 113, 116, 118–120, 123–125, 134
AES-128, 55, 116, 119, 121, 122, 149
AES-192, 149
AES-256, 149
AES-CCB, 105
AES-GCM, 133, 134
AES-NI, 134
API, 133
application-specific produced, 29, 30
application-specific programmable, 29
ASIC, 28, 29, 33, 41, 53–55, 63, 97
asymmetric, 13, 41
authenticated cipher, 22, 25, 133, 145, 159
authenticated encryption, 22, 105, 126, 133, 144, 145, 184
authenticated tag, 41, 105
authentication, 18, 22, 24, 104–109, 126–132, 158
authentication engine, 109, 110, 158
authenticity, 105, 130, 133, 138–141, 143

Barrett's algorithm, 66–69, 71–73, 177
Bazeries cylinder, 13
BCH, 23, 24, 66, 67, 75–80
BCH-Barrett, 80
BCH-Barrett-pipelined, 80
BCH-LFSR, 79, 80
BCH-LFSR-improved, 79, 80
BCH-standard, 78, 80
behavioral domain, 27
Berlekamp-Massey algorithm, 77
bipolar, 35
birthday attack, 59
birthday paradox, 59
bitcoin, 66
block-cipher, 14, 18, 25, 51, 62, 63, 133, 144, 145
brute-force attack, 16, 77

CAESAR, 22–25, 133, 145, 149, 159
carry-lookahead, 62
carry-propagate, 62
carry-save, 62–64
CCA, 138, 140, 141, 143
Chien's search, 77
CIFA, 112, 121–123, 125
ciphertext, 15, 18, 42, 50, 93, 97, 138, 141, 145, 147
circuit layout, 34
CLB, 97
clock domain, 109, 110
clock frequency, 27, 62, 80, 100, 101, 110, 117
closure, 19
CMOS, 30–37, 40, 66, 108, 112
codeword, 15, 75, 76, 79
collision resistance, 57
commutative, 19
compound gate, 34
compression, 25, 61, 62, 133, 134, 140, 143–146, 148–152, 159
confidentiality, 15, 104, 133, 138, 139
CPA, 62, 94, 121, 123, 125, 138, 140, 143
cryptanalysis, 13, 16–18, 133, 159
cryptoalgorithm, 15
cryptogram, 15
cryptographic engineering, 17, 18
cryptography, 13, 14, 18, 23, 50
cryptology, 13, 14, 19
cryptosystem, 13–19, 23, 26, 27, 41–43, 50, 106
CSAC, 106–111, 158
CSBA, 121, 123, 125

data integrity, 104
decomposition, 113, 114, 116, 119, 121, 122
delay balancing, 62, 63
denial-of-service, 103, 157
DES, 14, 18, 50, 51, 93
digest message, 64, 107, 146, 148
digital library, 30, 31, 80, 96, 100, 101, 109, 158
distinguisher, 112, 119, 123
distributive, 19, 128
DPA, 18, 94, 96, 97, 112, 121
DSBA, 112
DVS, 112, 113, 117, 124, 125

electromagnetic emanations, 18, 93
EMD, 113, 114, 119
encryption table, 42
endomorphic, 42
Enigma machine, 13

error-locator polynomial, 76–78
expansion, 61

fault attack, 24, 93, 108
fault detection, 93, 97, 98, 107
fault injection, 104, 110, 132
fault resistance, 93, 110
Feistel, 14, 50
Fiat-Shamir, 126–131
finite field, 19, 75
firewall, 102–106, 109–111, 157, 158
Fourier analysis, 118
Fourier transform, 24, 112, 117, 118
FPGA, 24, 29, 30, 33, 41, 53–56, 62, 63, 65, 94, 96, 97, 100, 110, 113, 116, 117, 119, 124
FreePDK, 80, 100, 101
frequency analysis, 24, 112, 113, 122, 124
full-custom, 29, 30

GaAs, 35
Gajski-Kuhn $\mathcal{Y}$-chart, 28
Galois Field, 20, 51, 55, 136
gate equivalent, 80, 109, 110
gate leakage, 40
gate-array, 29, 30
generator polynomial, 75, 79, 80
glitching, 103, 157

Hamming code, 67, 75
Hamming distance, 94, 118, 119, 122, 123, 125
Hamming weight, 119
hardcoded key, 108
hardware accelerator, 29, 66
hardware countermeasure, 86
hardware design, 17, 18, 28, 41, 42, 54, 61–63, 78, 104, 105, 112, 116
hardware multiplier, 67
hardware utilization, 54
hardwired, 28–30, 97, 98
HHT, 112–119, 122
hijacking, 103, 105, 157
HMAC, 106–111, 133, 158

index frequency, 115
indistinguishability, 131, 138
instantaneous amplitude, 113, 115
instantaneous frequency, 24, 112, 113, 115, 117, 118, 121, 122, 124
instantaneous phase, 113
integrated circuit, 18, 27, 30, 96
integrity, 13, 22, 24, 103–106, 108, 109, 127, 132, 133, 138, 139, 158
integrity check, 105, 106, 108–110
interconnect, 102–104, 107
inverse, 19, 53, 93
irreducible polynomial, 20, 136

junction leakage, 40

kernel, 41
key compression, 22, 25, 133, 136, 145–149, 159
key delivery, 106, 158
key expansion, 51, 54, 63, 99
key integrity, 108
key management system, 108
key register, 108
key schedule, 55, 116
key-setup latency, 53
keyless compression, 136

last round, 64, 93, 118–120, 123, 125
latency, 55, 56, 63, 65, 79, 102, 105
leakage, 18, 36, 40, 93–96, 101, 109, 112, 117, 118, 134
LFSR, 24, 79, 80, 95, 96, 100, 101
lightweight, 22, 24, 35, 66, 105, 127, 184
logic balancing, 56, 62, 63

man-in-the-middle, 132
mask programmable, 29
masking, 144, 146, 147
masking function, 144
message compression, 61
message schedule, 61, 150, 152
microelectronics, 27
minimal polynomial, 75
MixColumns, 51, 53–55, 93–97
modular division, 78, 79
modular multiplication, 131
modular reduction, 23, 24, 67, 133, 159
modular squaring, 131
modulo, 19, 20, 23, 61, 62, 67, 127
Mooij-Goga-Wesselink algorithm, 128, 129
Moore's Law, 27
MOSFET, 30
multivariate polynomial, 67, 71

netlist key, 108
network topology, 126, 128, 129, 131, 132
network-on-chip, 102–106, 110, 157
nMOS, 29–33, 35–38
NoC attacks, 103, 104
nonce, 22, 25, 105, 107, 133, 137–142, 144, 145, 147, 149, 159

OMD, 22, 24, 25, 133, 134, 137, 139, 140, 144–149, 159
on-chip fabric, 102
one-time pad, 14
one-wayness, 58
operating system, 41
oscilloscope, 89
overflow check, 108

padding, 61, 142, 146, 147, 149
parity check, 75, 76, 107–110
PCLMULQDQ, 134
permanent fault, 97–99
permutation, 25, 50, 51, 63, 99, 133

Peterson's algorithm, 77, 78
physical domain, 27, 28
pigeonhole principle, 59
pipeline, 24, 55, 56, 62–65, 79, 80, 94, 98, 108
plaintext, 15, 18, 42, 50, 55, 93, 94, 97, 100, 118, 138, 139, 145
pMOS, 29–33, 35–38
polynomial reduction, 23, 24, 67
polynomial time, 130, 131
power analysis, 18, 94, 112, 113, 121, 124, 125
power attack, 18, 24, 35, 93, 94, 112, 125
power consumption, 18, 23, 24, 27, 29, 30, 32, 34, 40, 63, 80–84, 93–96, 98–101, 109, 110, 112, 116–122, 124, 127, 130
power management, 102
power minimization, 66
power model, 94, 112, 123
power scrambling, 93, 94, 96–99
power trace, 18, 95–97, 113, 117–119, 121–125
private-key, 41, 50, 51, 63, 129, 131
programmable logic array, 29
programming sequence, 104–107, 109, 110, 158
pseudo-random, 50, 95, 137
pseudocode, 78
public-key, 13, 14, 23, 24, 67, 127, 129–131
pull-down, 31, 32, 34, 35
pull-up, 31, 32, 34, 35

QoS, 102, 103
quasi-pipelining, 62

replay attack, 103, 105, 107, 157
reprogramming agent, 103, 105, 106, 157, 158
request path, 103, 105
reverse engineer, 13
Rijndael, 51, 53, 93
RSA, 18, 51, 87, 89, 90, 127
RSA signature, 89
runtime configuration, 24, 93, 98, 99

S-Box, 50, 53, 94, 117
secrecy, 14, 15, 50
secret key, 13, 15, 18, 41, 50, 63, 108, 127, 131, 144, 145
semi-custom, 29, 30
session key, 107, 108
SHA, 25, 61, 62, 133
SHA-1, 25, 133, 145
SHA-2, 61–65, 145
SHA-256, 25, 61, 62, 64, 109–111, 133, 134, 139, 145, 146, 149, 150, 159
SHA-3, 25
SHA-512, 25, 62, 133, 139, 140, 148, 149, 151, 152, 159
ShiftRows, 51, 53, 55, 93–97
Shockley model, 36
side-channel, 18, 24, 41, 86, 87, 94, 112, 113, 117–119, 121
single-channel MOS, 35
smart-card, 41
SoC, 102–108, 158
software algorithm, 112
software certification, 41
software design, 17, 18, 25, 28, 41, 63, 134
software multiplier, 67
software polling, 110
software security, 41
soundness, 130
standard cell, 29, 30
standard circuit, 28, 29
structural domain, 27, 28
SubBytes, 51, 55, 93–97
substitution, 50, 51, 63
subthreshold conduction, 40
switching activity, 80, 112, 117
switching lemma, 144
symmetric, 13, 14, 41, 50, 63, 67
syndrome, 76, 77
syndrome circuit, 76
syndrome decoding, 76

tag, 134, 139–143, 145–149
technology node, 27, 29, 33, 35, 36, 40
throughput, 54–56, 62, 63, 80, 100, 101, 103, 157
timing attack, 112, 134, 136
topology-aware, 126, 128
transient fault, 97–99
transmission gate, 34
tri-state, 32, 34, 35, 94, 96, 97, 99–101
trusted code, 108
trusted flow, 102
trusted party, 127
trusted software, 23

unrolling, 55, 62, 63
untrusted flow, 102
untrusted software, 23

Verilog, 24, 44, 100
VHDL, 44, 117

working variable, 150, 152

zero-knowledge, 22, 24, 126, 127, 129, 130, 184

List of Main Abbreviations

A-message	Authentication Message
AC-message	Authentication Challenge Message
AR-message	Authentication Request Message
V_{DD}	Supply Voltage
V_{SS}	Ground Voltage
[	i
3D	Three-Dimensional
3DES	Triple Data Encryption Standard
AE	Authenticated Encryption
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-GCM	Galois/Counter Mode
AES-NI	Intel AES New Instructions
ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
AND	Conjunction Logic Gate
API	Application Programming Interface
ARM	Acorn RISC Machine
ASIC	Application-Specific Integrated Circuit
BA	Base Address
BCH	Bose-Chaudhuri-Hocquenghem
CAD	Computer-Aided Design
CAESAR	Competition for Authenticated Encryption: Security, Applicability, and Robustness
CBC	Cipher Block Chaining
CCA	Chosen-Ciphertext Attack
CFB	Cipher Feedback Mode
CHES	Cryptographic Hardware and Embedded Systems
CIFA	Correlation Instantaneous Frequency Analysis
CLA	Carry-Lookahead Addition
CLB	Configurable Logic Block
CMOS	Complementary Metal-oxide Semiconductor
CPA	Carry-Propagate Adder
CPA	Chosen-Plaintext Attack
CPA	Correlation Power Attack
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSA	Carry-Save Addition
CSAC	Cryptographically Secure Access Control
CSBA	Correlation Spectral Based Analysis
CTR	Counter Mode

DDR	Double Data Rate
DES	Data Encryption Standard
DFA	Differential Frequency Analysis
DFT	Design for Testability
DHT	Discrete Hilbert Transform
DM	Digest Message
DMA	Direct Memory Access
DPA	Differential Power Analysis
DRM	Digital Rights Management
DSBA	Differential Spectral Based Analysis
DSP	Digital Signal Processing
DVS	Dynamic Voltage Scrambling
ECB	Electronic Codebook
ECC	Error-Correcting Code
EMD	Empirical Mode Decomposition
EPROM	Erasable Programmable Read-Only Memory
FFT	Fast Fourier Transform
FIPS	Federal Information Processing Standards
FPGA	Field-Programmable Gate Array
FSR	Feedback Shift Register
G/s	Giga per Second
GCM	Galois Counter Mode
GE	Gate Equivalent
GF	Galois Field
GHz	Gigahertz
HD	Hamming Distance
HDL	Hardware Description Language
HHT	Hilbert Huang Transform
HMAC	Keyed-Hash Message Authentication Code
HW	Hamming Weight
I/O	Input/Output
IBM	International Business Machines
IC	Integrated Circuit
IDEA	International Data Encryption Algorithm
IF	Instantaneous Frequency
IMF	Intrinsic Mode Function
IND-CPA	Indistinguishability Under Chosen-Plaintext Attack
INT-CTXT	Integrity of Ciphertext
IoT	Internet of Things
KMS	Key Management System
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bits
LUT	Look Up Table
Mbit/s	Megabit per Second
MHz	Megahertz
MITM	Man-in-the-Middle
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MSB	Most Significant Bits
MUX	Multiplexer Logic Gate
NAND	Negative-AND Gate
NBS	National Bureau of Standards

NIST	National Institute of Standards and Technology
NM-CCA	Non-Malleability Under Chosen-Ciphertext Attack
NoC	Network-on-Chip
NOR	Negative OR Logic Gate
NP	Non-Polynomial
NRE	Non-Recurring Engineering
ns	Nanosecond
NSA	National Security Agency
NSR	Non-Secure Read
NSW	Non-Secure Write
OCB	Offset Codebook Mode
OFB	Output Feedback Mode
OMD	Offset Merkle-Damgård
OR	Disjunction Logic Gate
OTP	One-Time Programmable
PC	Parallel Counter
PC	Personal Computer
PCLMULQDQ	Intel Carry-Less Multiplication of two 64-bit Polynomials Over the Finite Field *GF(2)*
PDK	Process Design Kit
PLA	Programmable Logic Array
PRF	Pseudo Random Function
PRNG	Pseudo Random Number Generator
PRP	Pseudo Random Permutation
PSD	Power Spectral Density
QoS	Quality-of-Service
RAM	Random Access Memory
RF	Radio-Frequency
RF	Random Function
ROTR	Rotate No Carry
RSA	Rivest, Shamir and Adleman
RTL	Register-Transfer Level
S-Box	Substitution Box
SCA	Side-Channel Analysis
SHA	Secure Hash Algorithm
SHR	Right Logical Shift
SoC	System-on-Chip
SPA	Simple Power Analysis
SR	Secure Read
SW	Secure Write
TA	Top Address
TDMA	Time Division Multiple Access
VHDL	VHSIC Hardware Description Language
VLSI	Very-Large-Scale Integration
XOR	eXclusive OR Logic Gate
ZK	Zero Knowledge
ZKP	Zero-Knowledge Protocol

BIBLIOGRAPHY

[AARR03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45, Redwood Shores, California, USA, August 13–15, 2003. Springer, Heidelberg, Germany. 18, 88, 118

[ABK98] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A New Block Cipher Proposal. *NIST AES Proposal*, 1998. 44

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001. 41, 55, 93

[AG01] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and AES, secure against some attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318, Paris, France, May 14–16, 2001. Springer, Heidelberg, Germany. 93

[AHMN13] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. *Journal of Cryptology*, 26(2):313–339, April 2013. 155

[Ajt88] Miklós Ajtai. The complexity of the pigeonhole principle. In *29th Annual Symposium on Foundations of Computer Science*, pages 346–355, White Plains, New York, October 24–26, 1988. IEEE Computer Society Press. 59

[AR05] Dev Anshul and Suman Roy. A ZKP-based identification scheme for base nodes in wireless sensor networks. In Bart Preneel and Stafford Tavares, editors, *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, Kingston, Ontario, Canada, August 11–12, 2005. Springer, Heidelberg, Germany. 127

[ARM13] TZC-400 TrustZone Address Space Controller® Technical Reference Manual, 2013. 105

[Art14] FlexNoC® Interconnect IP, 2014. 105

[Bar87] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. 66, 67

[BBK+03] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Trans. Computers*, pages 492–505, 2003. 93, 98

[BCD+99] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr, Luke O'Connor, Mohammad Peyravian, Jr. Luke, O'connor Mohammad Peyravian, David Stafford, and Nevenko Zunic. Mars - a candidate cipher for aes. *NIST AES Proposal*, 1999. 44

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany. 118

[BD15] Kirti Bhanushali and William Rhett Davis. FreePDK15: An Open-Source Predictive Process Design Kit for 15nm FinFET Technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, pages 165–170, Monterey, CA, USA, 2015. 80

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press. 138

[BECN+04] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. Cryptology ePrint Archive, Report 2004/100, 2004. http://eprint.iacr.org/2004/100. 104

[Bel06] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. Cryptology ePrint Archive, Report 2006/043, 2006. http://eprint.iacr.org/2006/043. 133

[Ben73] Charles Bennett. Logical Reversibility of Computation. *IBM Journal of Research and Development*, pages 525–532, 1973. 112

[Ber05] Daniel Bernstein. Cache-Timing Attacks on AES, 2005. 134, 160

[BGM04] Mihir Bellare, Oded Goldreich, and Anton Mityagin. The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309, 2004. http://eprint.iacr.org/2004/309. 141, 143

[BGR95] Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, pages 15–28, 1995. 160

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466, Vienna, Austria, September 10–13, 2007. Springer, Heidelberg, Germany. 155

[BKMG07] Bradley Battista, Camelia Knapp, Tom McGee, and Vaughn Goebel. Application of the Empirical Mode Decomposition and Hilbert-Huang Transform to Seismic Reflection Data. In *Geophysics*, pages H29–H37. SEG, 2007. 122

[BKMG12] Bradley Battista, Camelia Knapp, Tom McGee, and Vaughn Goebel. Matlab Program Demonstrating Performing the Empirical Mode Decomposition and Hilbert-Huang Transform on Seismic Reflection Data, 2012. 122

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Cryptology ePrint Archive, Report 2000/025, 2000. http://eprint.iacr.org/2000/025. 133, 138

[Boa92] Boualem Boashash. Estimating and Interpreting the Instantaneous Frequency of a Signal. I. Fundamentals. In *Proceedings of the IEEE*, pages 520 –538, 1992. 113

[BR60] Raj Chandra Bose and Dwijendra Ray-Chaudhuri. On A Class of Error Correcting Binary Group Codes. *Information and Control*, pages 68–79, 1960. 67, 75

[BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany. 133

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991. 50

[BS93] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany. 88

[BWY06] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 245–254, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. 18

[BZ07] Karthik Baddam and Mark Zwolinski. Evaluation of Dynamic Voltage and Frequency Scaling as a Differential Power Analysis Countermeasure. In *Proceedings of the 20th International Conference on VLSI Design held jointly with the 6th International Conference: Embedded Systems*, VLSID '07, pages 854–862, Bangalore, India, 2007. IEEE Computer Society. 112, 124

[CAE] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html. 25, 156

[CCGD12] Pascal Cotret, Jeremie Crenne, Guy Gogniat, and Jean-Philippe Diguet. Bus-based MP-SoC Security Through Communication Protection: A Latency-efficient Alternative. In *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, FCCM '12, pages 200–207, Washington, DC, USA, 2012. IEEE Computer Society. 106

[CDKM04] Francis Crowe, Alan Daly, Tim Kerins, and William P. Marnane. Single-chip FPGA Implementation of a Cryptographic Co-Processor. In *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology*, pages 279–285, Brisbane, Australia, 2004. 62

[CEGK98] Guy Côté, Berna Erol, Michael Gallant, and Faouzi Kossentini. H.263+: Video Coding at Low Bit Rates. *IEEE Transactions on. Circuits and Systems for Video Technology*, pages 849–866, 1998. 67

[CG03] Pawel Chodowiec and Kris Gaj. Very compact FPGA implementation of the AES algorithm. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 319–333, Cologne, Germany, September 8–10, 2003. Springer, Heidelberg, Germany. 54

[Cha24] Jean-François Champollion. *Précis du Système Hiéroglyphique des Anciens Égyptiens*. Imprimerie royale, 1824. 13

[CHdAdC15] Jean-Michel Cioranesco, Craig Hampel, Guilherme Ozari de Almeida, and Rodrigo Portella do Canto. Cryptographically Secure On-Chip Firewalling. In *Network and System Security - 9th International Conference*, NSS'15, pages 428–438, New York, NY, USA, 2015. 23

[Chi06] Robert Chien. Cyclic Decoding Procedures for Bose- Chaudhuri-Hocquenghem Codes. *IEEE Transactions on Information Theory*, pages 357–363, 2006. 67, 75, 77

[CHVV03] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Password interception in a SSL/TLS channel. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. 133

[CKSV06] Ricardo Chaves, Georgi Kuzmanov, Leonel Sousa, and Stamatis Vassiliadis. Improving SHA-2 hardware implementations. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 298–310, Yokohama, Japan, October 10–13, 2006. Springer, Heidelberg, Germany. 63, 64, 65

[CMN+14] Simon Cogliani, Diana-Stefania Maimuţ, David Naccache, Rodrigo Portella do Canto, Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. OMD: A compression function mode of operation for authenticated encryption. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014: 21st Annual International Workshop on Selected Areas in Cryptography*, volume 8781 of *Lecture Notes in Computer Science*, pages 112–128, Montreal, QC, Canada, August 14–15, 2014. Springer, Heidelberg, Germany. 25

[CP98] Crispin Cowan and Calton Pu. Death, Taxes, and Imperfect Software: Surviving the Inevitable. In *Proceedings of the 1998 Workshop on New Security Paradigms*, NSPW '98, pages 54–70, New York, NY, USA, 1998. 41

[CS07] Debrup Chakraborty and Palash Sarkar. A general construction of tweakable block ciphers and different modes of operations. Cryptology ePrint Archive, Report 2007/029, 2007. http://eprint.iacr.org/2007/029. 10, 144, 146

[DES77] Data encryption standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, January 1977. 41

[DEV+07] Jean-Philippe Diguet, Samuel Evain, Romain Vaslin, Guy Gogniat, and Emmanuel Juin. NOC-centric Security of Reconfigurable SoC. In *Proceedings of the First International Symposium on Networks-on-Chip*, NOCS '07, pages 223–232, Washington, DC, USA, 2007. IEEE Computer Society. 103

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 13, 14

[DMO04a] Luigi Dadda, Marco Macchetti, and Jeff Owen. An ASIC Design for a High Speed Implementation of the Hash Function SHA-256 (384, 512). In *Proceedings of the 14th ACM Great Lakes Symposium on VLSI*, pages 421–425, Boston, MA, USA, 2004. 62

[DMO04b] Luigi Dadda, Marco Macchetti, and Jeff Owen. The Design of a High Speed ASIC Unit for the Hash Function SHA-256 (384, 512). In *Design, Automation and Test in Europe Conference and Exposition*, DATE '04, pages 70–75, Paris, France, 2004. 62

[DPR00] Andreas Dandalis, Viktor K. Prasanna, and José D. P. Rolim. A comparative study of performance of AES final candidates using FPGAs. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 125–140, Worcester, Massachusetts, USA, August 17–18, 2000. Springer, Heidelberg, Germany. 53, 55, 63

[DR99] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael. *NIST AES Proposal*, 1999. 44

[EKY11] Barics Ege, Elif Bilge Kavun, and Tolga Yalcin. Memory Encryption for Smart Cards. In *Proceedings of the 10th International Conference on Smart Card Research and Advanced Applications*, CARDIS'11, pages 199–216, Berlin, Heidelberg, 2011. Springer-Verlag. 105

[EYCP00] Adam Elbirt, Wai Yip, Brendon Chetwynd, and Christof Paar. An FPGA Implementation and Performance Evaluation of the AES Block-Cipher Candidate Algorithm Finalists. In *AES Candidate Conference*, pages 13–27, 2000. 55

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany. 54

[Fei73] Horst Feistel. *Cryptography and Computer Privacy*. Scientific American, 1973. 14

[Fei74] Horst Feistel. Block Cipher Cryptographic System, 1974. US Patent 3,798,359. 14

[FFLW11] Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. McOE: A family of almost foolproof on-line authenticated encryption schemes. Cryptology ePrint Archive, Report 2011/644, 2011. http://eprint.iacr.org/2011/644. 133

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988. 127

[Fou98] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998. 51

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. 127, 128, 130

[GB05] Tim Good and Mohammed Benaissa. AES on FPGA from the fastest to the smallest. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 427–440, Edinburgh, UK, August 29 – September 1, 2005. Springer, Heidelberg, Germany. 55, 56, 65

[GBC+96] Ulrich Golze, Peter Blinzer, Elmar Cochlovius, Michael Schafers, and Klaus-Peter Wachsmann. *VLSI Chip Design with the Hardware Description Language VERILOG: An Introduction Based on a Large RISC Processor Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1996. 8, 28, 29

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442, Washington, D.C., USA, August 10–13, 2008. Springer, Heidelberg, Germany. 118

[GC01] Kris Gaj and Pawel Chodowiec. Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 84–99, San Francisco, CA, USA, April 8–12, 2001. Springer, Heidelberg, Germany. 55

[GCC88] Marc Girault, Robert Cohen, and Mireille Campana. A generalized birthday attack. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 129–156, Davos, Switzerland, May 25–27, 1988. Springer, Heidelberg, Germany. 59

[GHT05] Catherine H. Gebotys, Simon Ho, and C. C. Tiu. EM analysis of Rijndael and ECC on a wireless Java-based PDA. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 250–264, Edinburgh, UK, August 29 – September 1, 2005. Springer, Heidelberg, Germany. 112

[GK83] Daniel Gajski and Robert Kuhn. New VLSI Tools. *Computer*, pages 11–14, 1983. 8, 28

[GLG+02] Tim Grembowski, Roar Lien, Kris Gaj, Nghi Nguyen, Peter Bellows, Jaroslav Flidr, Tom Lehman, and Brian Schott. Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. In Agnes Hui Chan and Virgil D. Gligor, editors, *ISC 2002: 5th International Conference on Information Security*, volume 2433 of *Lecture Notes in Computer Science*, pages 75–89, Sao Paulo, Brazil, September 30 – October 2, 2002. Springer, Heidelberg, Germany. 62

[GM90] Paul Gray and Robert Meyer. *Analysis and Design of Analog Integrated Circuits*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1990. 39

[GMN+15] Rémi Géraud, Diana-Stefania Maimuţ, David Naccache, Rodrigo Portella do Canto, and Emil Simion. Applying cryptographic acceleration techniques to error correction. Cryptology ePrint Archive, Report 2015/886, 2015. http://eprint.iacr.org/2015/886. 23

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261, Paris, France, May 14–16, 2001. Springer, Heidelberg, Germany. 18, 88

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. 127

[GP07] Gert-Martin Greuel and Gerhard Pfister. *A Singular Introduction to Commutative Algebra*. Springer, 2nd edition, 2007. 77

[GPZ60] Daniel Gorenstein, William Wesley Peterson, and Neal Zierler. Two-Error Correcting Bose-Chaudhuri Codes are Quasi-Perfect. *Information and Control*, pages 291–294, 1960. 67, 75

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128, Davos, Switzerland, May 25–27, 1988. Springer, Heidelberg, Germany. 127

[GTC05] Catherine Gebotys, Agnes Tiu, and Xiaojing Chen. A Countermeasure for EM Attack of a Wireless PDA. In *International Symposium on Information Technology: Coding and Computing*, ITCC '05, pages 544–549, Las Vegas, NV, USA, 2005. 112

[Gut03] Peter Gutmann. *Cryptographic Security Architecture Design and Verification*. SpringerVerlag, 2003. 41

[HC99] Ivan Hamer and Paul Chow. DES cracking on the transmogrifier 2a. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 13–24, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Heidelberg, Germany. 50

[HCPdCOdA15] Craig Hampel, Jean-Michel Cioranesco, Rodrigo Portella do Canto, and Guilherme Ozari de Almeida. Implementing Access Control by System-on-Chip, July 30 2015. WO Patent App. PCT/US2015/013,095. 23

[HEL05] HELION. HELION: Fast SHA-2 (256) Hash Core for Xilinx FPGA. *http://www.heliontech.com/hash.htm*, 2005. 64, 65

[Hoc59] Alexis Hocquenghem. Codes Correcteurs d'Erreurs. *Chiffres*, pages 147–158, 1959. 77

[HS05] Norden Huang and Samuel Shen. *The Hilbert-Huang Transform and its Applications*. World Scientific Publishing Company, 2005. 113

[HSL+98] Norden Huang, Zheng Shen, Steven Long, Manli Wu, Hsing Shih, Quanan Zheng, Nai-Chyuan Yen, Chi Chao Tung, and Henry Liu. The Empirical Mode Decomposition and the Hilbert Spectrum for Nonlinear and Non-Stationary Time Series Analysis. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, pages 903–995, 1998. 113, 115

[HV04] Alireza Hodjat and Ingrid Verbauwhede. A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA. In *12th IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM '04, pages 308–309, Napa, CA, USA, 2004. 56, 65

[IKM00] Tetsuya Ichikawa, Tomomi Kasuya, and Mitsuru Matsui. Hardware Evaluation of the AES Finalists. In *AES Candidate Conference*, pages 279–285, 2000. 55

[Int13] Intel® SHA Extensions, 2013. 133

[IOM12] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. 134

[Jr.96] Burton Kaliski Jr. IEEE P1363: A Standard for RSA, Diffie-Hellman, and Elliptic-Curve Cryptography. In *Security Protocols, International Workshop*, pages 117–118, Cambridge, UK, 1996. 18

[JTS03] Kimmo Järvinen, Matti Tommiska, and Jorma Skyttä. A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor. In *Proceedings of the 2003 ACM/SIGDA - 11th International Symposium on Field Programmable Gate Arrays*, FPGA '03, pages 207–215, Monterey, CA, USA, 2003. 56, 65

[Ker83] Auguste Kerckhoffs. La Cryptographie Militaire. *Journal des Sciences Militaires*, pages 5–83, 1883. 16

[Key75] Robert Keyes. Physical Limits in Digital Electronics. In *IEEE Proceedings*, pages 740–767, 1975. 112

[KGS+11] Armin Krieg, Johannes Grinschgl, Christian Steger, Reinhold Weiss, and Josef Haid. A Side Channel Attack Countermeasure Using System-on-chip Power Profile Scrambling. In *Proceedings of the 2011 IEEE 17th International On-Line Testing Symposium*, IOLTS '11, pages 222–227, Washington, DC, USA, 2011. IEEE Computer Society. 112, 124

[Kil07] Steve Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Wiley-IEEE Press, 2007. 42

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany. 17, 18, 89, 93, 118

[KLMR04] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. Security As a New Dimension in Embedded System Design. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 753–760, New York, NY, USA, 2004. ACM. 104

[KM10] Daniel Kaslovsky and François Meyer. Noise Corruption of Empirical Mode Decomposition and its Effect on Instantaneous Frequency. *Advances in Adaptive Data Analysis*, pages 373–396, 2010. 113

[KNdAdC13] Roman Korkikian, David Naccache, Guilherme Ozari de Almeida, and Rodrigo Portella do Canto. Practical Instantaneous Frequency Analysis Experiments. In *E-Business and Telecommunications - International Joint Conference*, ICETE '13, pages 17–34, Reykjavik, Iceland, 2013. 24

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany. 17, 18, 87, 93, 112

[Koc08] Cetin Kaya Koc. *Cryptographic Engineering*. Springer Publishing Company, Incorporated, 1st edition, 2008. 44, 90, 91

[KR11] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327, Lyngby, Denmark, February 13–16, 2011. Springer, Heidelberg, Germany. 10, 144, 146, 160

[Kuh02] Markus G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *2002 IEEE Symposium on Security and Privacy*, pages 3–18, Berkeley, California, USA, May 12–15, 2002. IEEE Computer Society Press. 18

[KV01] Henry Kuo and Ingrid Verbauwhede. Architectural optimization for a 1.82Gbits/sec VLSI implementation of the AES Rijndael algorithm. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 51–64, Paris, France, May 14–16, 2001. Springer, Heidelberg, Germany. 55

[KY01] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299, New York, NY, USA, April 10–12, 2001. Springer, Heidelberg, Germany. 133

[LG14] Michael LeMay and Carl A. Gunter. Network-on-Chip Firewall: Countering Defective and Malicious System-on-Chip Hardware. *Computing Research Repository*, abs/1404.3465, 2014. 103, 158

[LGG04] Roar Lien, Tim Grembowski, and Kris Gaj. A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512. In Tatsuaki Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 324–338, San Francisco, CA, USA, February 23–27, 2004. Springer, Heidelberg, Germany. 62

[LU02] Joe Loughry and David Umphress. Information Leakage From Optical Emanations. *ACM Transactions on Information and System Security*, pages 262–289, 2002. 18

[Luo10] Qiasi Luo. Enhance Multi-bit Spectral Analysis on Hiding in Temporal Dimension. In *Smart Card Research and Advanced Application*, Lecture Notes in Computer Science, pages 13–23. Springer, 2010. 112

[LV08] Christian Lavault and Mario Valencia-Pabon. A Distributed Approximation Algorithm for the Minimum Degree Minimum Weight Spanning Trees. *Journal of Parallel and Distributed Computing*, pages 200–208, 2008. 128

[MC80] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980. 112

[MCMM06] Robert McEvoy, Francis Crowe, Colin Murphy, and William Marnane. Optimization of the SHA-2 Family of Hash Functions on FPGAs. In *IEEE Computer Society Annual Symposium on VLSI*, ISVLSI '06, pages 317–322, Karlsruhe, Germany, 2006. 64, 65

[MD05] Marco Macchetti and Luigi Dadda. Quasi-Pipelined Hash Circuits. In *17th IEEE Symposium on Computer Arithmetic*, ARITH-17 '05, pages 222–229, Cape Cod, MA, USA, 2005. 62

[Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, April 1978. 13

[MG11] Edgar Mateos and Catherine Gebotys. Side Channel Analysis using Giant Magneto-Resistive (GMR) Sensors. In *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design*, COSADE '11, pages 42–49, Darmstadt, Germany, 2011. 112

[MGW] Arjan Mooij, Nicolae Goga, and Wieger Wesselink. A Distributed Spanning Tree Algorithm for Topology-Aware Networks. DASD '04. 128, 129

[MM01] Máire McLoone and John V. McCanny. High performance single-chip FPGA Rijndael algorithm implementations. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 65–76, Paris, France, May 14–16, 2001. Springer, Heidelberg, Germany. 55

[MM02] Máire McLoone and John McCanny. Efficient Single-Chip Implementation of SHA-384 and SHA-512. In *Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology*, FPT '02, pages 311–314, Hong Kong, China, 2002. 62

[MM03] Maire McLoone and John McCanny. *System-On-Chip Architectures and Implementations for Private-Key Data Encryption*. Plenum Publishing Co., 2003. 13, 41, 42

[Mon85] Peter Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, pages 519–521, 1985. 87

[Moo00] Gordon Moore. Readings in computer architecture. chapter Cramming More Components Onto Integrated Circuits, pages 56–59. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000. 27

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007. 93

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North Holland, 1977. 77

[MvV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997. 8, 15

[Nat04] National Institute of Standards and Technology (NIST). *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*, 2004. 105

[Nat06] National Institute of Standards and Technology (NIST). *Security Requirements for Cryptographic Modules*, 2006. 41

[Nat08] National Institute of Standards and Technology (NIST). *The Keyed-Hash Message Authentication Code*, 2008. 106, 158

[Nat12] National Institute of Standards and Technology (NIST). SHA-3 Competition, 2012. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`. 156

[OMA14] OMAP543x® Technical Reference Manual, 2014. 105

[OSBHR10] Paul Duplys Oliver Schimmel, Eberhard Böhl, Jan Hayek, and Wolfgang Rosenstiel. Correlation Power Analysis in Frequency Domain. In *First International Workshop on Constructive Side-Channel Analysis and Secure Design*, COSADE '10, pages 1–3, Darmstadt, Germany, 2010. 112

[PdCK14] Rodrigo Portella do Canto and Roman Korkikian. Hardware Encryption and Decryption Apparatus Using a N Round AES Algorithm, April 16 2014. EP Patent App. EP20,120,306,245. 24

[PdCKN16] Rodrigo Portella do Canto, Roman Korkikian, and David Naccache. *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, chapter Buying AES Design Resistance with Speed and Energy, pages 134–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. 24

[PGQK09] Zhang Peng, Deng Gaoming, Zhao Qiang, and Chen Kaiyan. EM Frequency Domain Correlation Analysis on Cipher Chips. In *Information Science and Engineering*, ICISE '09, pages 1729–1732, Nanjing, China, 2009. 112

[PP09] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer, 1st edition, 2009. 8, 13, 17, 57

[PST+02] Adrian Perrig, Robert Szewczyk, Doug Tygar, Victor Wen, and David Culler. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, pages 521–534, 2002. 127

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In *Smart Card Programming and Security, International Conference on Research in Smart Cards*, E-SMART '01, pages 200–210, Cannes, France, 2001. 18

[Raz06] Behzad Razavi. *Fundamentals of Microelectronics*. Wiley, 2006. 27

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM CCS 01: 8th Conference on Computer and Communications Security*, pages 196–205, Philadelphia, PA, USA, November 5–8, 2001. ACM Press. 133, 138

[RMMM03] Kaushik. Roy, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. *Proceedings of the IEEE*, pages 305–327, 2003. 39

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 98–107, Washington D.C., USA, November 18–22, 2002. ACM Press. 133, 137, 144, 159

[Rog04a] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31, Jeju Island, Korea, December 5–9, 2004. Springer, Heidelberg, Germany. 10, 144, 146

[Rog04b] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359, New Delhi, India, February 5–7, 2004. Springer, Heidelberg, Germany. 133, 159

[Ros96] David Rosenblum. Formal Methods and Testing: Why the State-of-the Art is Not the State-of-the Practice. *SIGSOFT Software*, pages 64–66, 1996. 41

[RRSY98] Ron Rivest, Matt Robshaw, Ray Sidney, and Yiqun Lisa Yin. The RC6 Block Cipher. *NIST AES Proposal*, 1998. 44

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 133

[RSQL04] Gaël Rouvroy, François-Xavier Standaert, Jean-Jacques Quisquater, and Jean-Didier Legat. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications. In *International Conference on Information Technology: Coding and Computing*, ITCC '04, pages 583–587, Las Vegas, NV, USA, 2004. 54

[SCG+03] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, MICRO'03, pages 339–350, Washington, DC, USA, 2003. IEEE Computer Society. 104

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, New York, second edition, 1996. 63

[Sch00] Bruce Schneier. A Self-study Course in Block-cipher Cryptanalysis. *Cryptologia*, pages 18–33, 2000. 16

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949. 14

[SHA93] Secure hash standard. National Institute of Standards and Technology, NIST FIPS PUB 180, U.S. Department of Commerce, May 1993. 60

[SHA95] Secure hash standard. National Institute of Standards and Technology, NIST FIPS PUB 180-1, U.S. Department of Commerce, April 1995. 61, 133, 146, 148, 149, 181, 182, 184

[Sho52] William Shockley. A Unipolar "Field-Effect" Transistor. *Proceedings of The Institute of Radio Engineers*, 40:1365–1376, 1952. 36

[SK] Nicolas Sklavos and Odysseas Koufopavlou. On the Hardware Implementations of the SHA-2 (256, 384, 512) Hash Functions. In *Proceedings Of The IEEE International Symposium On Circuits And Systems*. 64, 65

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. The Twofish Encryption Algorithm: A 128-bit Block Cipher. *NIST AES Proposal*, 1998. 44

[SL07] Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 661–670, San Diego, California, USA, June 11–13, 2007. ACM Press. 128

[SM99] Timothy Shimeall and John McDermott. Software Security in an Internet World: An Executive Summary. *IEEE Software*, pages 58–61, 1999. 41

[SMMS03] Giacinto Paolo Saggese, Antonino Mazzeo, Nicola Mazzocca, and Antonio Strollo. An FPGA-Based Performance Analysis of the Unrolling, Tiling, and Pipelining of the AES Algorithm. In *Field Programmable Logic and Application - 13th International Conference*, FPL '03, pages 292–302, Lisbon, Portugal, 2003. 56, 65

[SPG+12] Johanna Sepulveda, Ricardo Pires, Guy Gogniat, Wang Jiang Chau, and Marius Strum. QoSS Hierarchical NoC-based Architecture for MPSoC Dynamic Protection. *International Journal of Reconfigurable Computing*, pages 3:3–3:3, 2012. 103, 105

[SRQL03] François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 334–350, Cologne, Germany, September 8–10, 2003. Springer, Heidelberg, Germany. 56, 65

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption – FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195, Luxembourg, Luxembourg, March 26–28, 2007. Springer, Heidelberg, Germany. 155

[Ste94] Raymond Steele. *Mobile Radio Communications*. IEEE Press, 1994. 67

[Sti95] Douglas Stinson. *Cryptography - Theory and Practice*. Discrete Mathematics and its Applications Series. CRC Press, 1995. 17

[TAL97] Richard Tolimieri, Myoung An, and Chao Lu. *Mathematics of Multidimensional Fourier Transform Algorithms*. Signal Processing and Digital Filtering. Springer, New York, Berlin, Heidelberg, 1997. 70

[THM15] Wade Trappe, Richard Howard, and Robert Moore. Low-Energy Security: Limits and Opportunities in the Internet of Things. *IEEE Security & Privacy*, pages 14–21, 2015. 81

[Til99] Henk Van Tilborg. *Fundamentals of Cryptology: A Professional Reference and Interactive Tutorial*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1999. 14, 41

[TKM88] Kai-Yap Toh, Ping-Keung Ko, and Robert Meyer. An Engineering Model for Short-Channel MOS Devices. *IEEE Journal of Solid-State Circuits*, pages 950–958, 1988. 38

[Tyg96] J. D. Tygar. Atomicity in electronic commerce. In James E. Burns and Yoram Moses, editors, *15th ACM Symposium Annual on Principles of Distributed Computing*, pages 8–26, Philadelphia, PA, USA, August 23–26, 1996. Association for Computing Machinery. 92

[TYLL02] Kurt Ting, Steve Yuen, Kin-Hong Lee, and Philip Heng Wai Leong. An FPGA-Based SHA-256 Processor. In *Field-Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream - 12th International Conference*, FPL '02, pages 577–585, Montpellier, France, 2002. 62

[UMS11] Siba Udgata, Alefiah Mubeen, and Samrat Sabat. Wireless Sensor Network Security Model Using Zero Knowledge Protocol. In *Proceedings of IEEE International Conference on Communications*, ICC '11, pages 1–5, Kyoto, Japan, 2011. 127

[Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS... In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. 133

[Vau05] Serge Vaudenay. *A Classical Introduction to Cryptography: Applications for Communications Security*. Springer-Verlag New York, Inc., 2005. 13, 50

[Ver] Gilbert Vernam. Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications,. *American Institute of Electrical Engineers*. 14

[WBRF00] Bryan Weeks, Mark Bean, Tom Rozylowicz, and Chris Ficke. Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms. In *AES Candidate Conference*, pages 286–304, 2000. 55

[WH10] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010. 8, 11, 27, 28, 30, 38

[WT85] Robert Walker and Donald Thomas. A Model of Design Representation and Synthesis. In *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, DAC '85, pages 453–459, Piscataway, NJ, USA, 1985. 28

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. 60

[Zie60] Neal Zierler. On Decoding Linear Error-Correcting Codes. *IRE Transactions on Information Theory*, pages 450–459, 1960. 67, 75

[ZNC04] Joseph Zambreno, David Nguyen, and Alok Choudhary. Exploring Area/Delay Tradeoffs in an AES FPGA Implementation. In *Field Programmable Logic and Application - 14th International Conference*, FPL '04, pages 575–585, Leuven, Belgium, 2004. 56, 65

[ZP04] Xinmiao Zhang and Keshab Parhi. High-Speed VLSI Architectures for the AES Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 957–967, 2004. 56, 65

APPENDIX A

CODE: BARRETT'S ALGORITHM FOR POLYNOMIALS

$p_1(x) = \sum_{i=0}^{7}(10+i)x^i$ and $p_2(x) = x^3 + x^2 + 110$

```
(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 $\prec x^{lim}$

```
(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 $\prec x^{lim}$

```
(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 and Y representing K, L, P and $\vec{y}$)

```
(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))))
```

APPENDIX B

COMPRESSION FUNCTIONS

B.1 Compression Functions of SHA-256 and SHA-512

We recall the compression functions of the standard SHA-256 and SHA-512 hash functions following NIST FIPS PUB 180-4 [SHA95]. We denote the underlying compression functions of these standard hash functions by sha-256 and sha-512, respectively.

Note. In the following, by *word* we mean a group of either 32 bits (4 bytes) or 64 bits (8 bytes), depending on the compression function algorithm. Namely, in sha-256 each word is a 32-bit string and in sha-512 each word is a 64-bit string.

$\mathbf{ROTR^n(x)}$: The *rotate right* (circular right shift) operation, where x is a w-bit word and n an integer with $0 \leq n < w$, is defined by $\text{ROTR}^n(x) = (x \gg n) \vee (x \ll w - n)$

$\mathbf{SHR^n(x)}$: The *right shift* operation, where x is a w-bit word and n an integer with $0 \leq n < w$, is defined by $\text{SHR}^n(x) = (x \gg n)$.

Choice Function. Let m be 32 in the case of sha-256 and 64 in the case of sha-512. The *choice function* takes as input two m-bit words y and z, and one m-bit word x selector input, and returns an m-bit word. Every value of a single bit of x is used to select one of the bit of the m pairs (from (y_0, y_1) to (z_{m-2}, z_{m-1})). It is similar to an m-to-$m/2$ multiplexer (i.e., m 2-to-1 multiplexers). One can use indifferently whether inclusive OR ($\vee$) or exclusive OR ($\oplus$). The function is defined as follows:

$$\text{Ch} : \left| \begin{array}{rcl} \{0,1\}^m \times \{0,1\}^m \times \{0,1\}^m & \longrightarrow & \{0,1\}^m \\ x, y, z & \longmapsto & (x \wedge y) \oplus (\neg x \wedge z) \end{array} \right.$$

Majority Function. Let m be 32 in the case of sha-256 and 64 in the case of sha-512. In Boolean logic, the *majority function* (also called the median operator) is a function from n inputs to one output. The value of the operation is false when $n/2$ or more arguments are false, true otherwise. In sha-256/sha-512 design, *majority function* takes as input three m-bit words x, y, z, and returns an m-bit word. Such as *choice function*, one can use indifferently whether inclusive OR ($\vee$) or exclusive OR ($\oplus$). The function is defined as follows:

$$\text{Maj} : \left| \begin{array}{rcl} \{0,1\}^m \times \{0,1\}^m \times \{0,1\}^m & \longrightarrow & \{0,1\}^m \\ x, y, z & \longmapsto & (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \end{array} \right.$$

B.1.1 The Compression Function of SHA-256

Sigma Functions. The $\Sigma_0^{\{256\}}$ and $\Sigma_1^{\{256\}}$ functions are respectively represented by a multiplication by the $X^2 + X^{13} + X^{22}$ and $X^6 + X^{11} + X^{25}$ polynomials of $\mathbb{F}_2[X]/X^{32} + 1$, if one represents any 32-bit word $W = (W[0]W[1]...W[31])$ as a $\mathbb{F}_2[X]/X^{32} + 1$ polynomial $W[0] + W[1].X + W[2].X^2 + ... + W[31].X^{31}$. The functions as are follows:

$$\Sigma_0^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x) \end{array} \right.$$

$$\Sigma_1^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x) \end{array} \right.$$

The $\sigma_0^{\{256\}}$ and $\sigma_1^{\{256\}}$ functions are respectively represented by a multiplication by the $X^7 + X^{18}$ and $X^{17} + X^{19}$ polynomials of $\mathbb{F}_2[X]/X^{32} + 1$, if one represents any 32-bit word $W = (W[0]W[1]...W[31])$ as a $\mathbb{F}_2[X]/X^{32} + 1$ polynomial $W[0] + W[1].X + W[2].X^2 + ... + W[31].X^{31}$. These multiplications are applied on the quotient of the Euclidean division of the polynomial representation of a 32-bit word W, and the polynomial $P = X$. The functions as are follows:

$$\sigma_0^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) \end{array} \right.$$

$$\sigma_0^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x) \end{array} \right.$$

The Process. The sha-256 function process is defined as below:

$$\mathrm{sha}-256 \left| \begin{array}{rcl} \{0,1\}^{256} \times \{0,1\}^{512} & \longrightarrow & \{0,1\}^{256} \\ H, M & \longmapsto & C \end{array} \right.$$

Let H be the 256-bit *hash input* (chaining input) and M be the 512-bit *message input*. These two inputs are represented respectively by an array of eight 32-bit words $H_0, \cdots, H_7$ and an array of sixteen 32-bit words $M_0, \cdots, M_{15}$. The 256-bit output value C is also represented as an array of eight 32-bit words $C_0, \cdots, C_7$.

During the process of compression, a sequence of 64 constant 32-bit words, $K_0^{\{256\}}, ..., K_{63}^{\{256\}}$ are used. These 32-bit words represent the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers. We refer the reader to [SHA95] for a table containing these constants.

Furthermore, addition $(+)$ is performed modulo 2^{32}.

The compression function processes is detailed in Algorithm 9.

B.1.2 The Compression Function of SHA-512

Sigma Functions. The $\Sigma_0^{\{512\}}$ and $\Sigma_1^{\{512\}}$ functions are respectively represented by a multiplication by the $X^{28} + X^{34} + X^{39}$ and $X^{14} + X^{18} + X^{41}$ polynomials of $\mathbb{F}_2[X]/X^{64} + 1$, if one represents any 64-bit word $W = (W[0]W[1]...W[63])$ as a $\mathbb{F}_2[X]/X^{64} + 1$ polynomial $W[0] + W[1] \cdot X + W[2] \cdot X^2 + ... + W[63] \cdot X^{63}$. The functions are as follows:

$$\Sigma_0^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{ROTR}^{28}(x) \oplus \mathrm{ROTR}^{34}(x) \oplus \mathrm{ROTR}^{39}(x) \end{array} \right.$$

$$\Sigma_1^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{ROTR}^{14}(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{ROTR}^{41}(x) \end{array} \right.$$

The $\sigma_0^{\{512\}}$ and $\sigma_1^{\{512\}}$ functions are respectively represented by a multiplication by the $X^1 + X^8$ and $X^{19} + X^{61}$ polynomials of $\mathbb{F}_2[X]/X^{64} + 1$, if one represents any 64-bit word $W = (W[0]W[1]...W[63])$ as a $\mathbb{F}_2[X]/X^{64} + 1$ polynomial $W[0] + W[1] \cdot X + W[2] \cdot X^2 + ... + W[63] \cdot X^{63}$. These multiplications are applied on the quotient

Algorithm 9 Compression function of SHA-256

for $t \leftarrow 0, 15$ **do**
    $W_t \leftarrow M_t$
end for

for $t \leftarrow 16, 63$ **do**
    $W_t \leftarrow \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}$
end for

$(a, b, c, d, e, f, g, h) \leftarrow (H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$

for $t \leftarrow 0, 63$ **do**
    $T_1 \leftarrow h + \Sigma_1^{\{256\}}(e) + \mathrm{Ch}(e, f, g) + K_t^{\{256\}} + W_t$
    $T_2 \leftarrow \Sigma_0^{\{256\}}(a) + \mathrm{Maj}(a, b, c)$
    $(h, g, f, e, d, c, b, a) \leftarrow (g, f, e, d + T_1, c, b, a, T_1 + T_2)$
end for

$(C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7) \leftarrow (a + H_0, b + H_1, c + H_2, d + H_3, e + H_4, f + H_5, g + H_6, h + H_7)$

of the Euclidean division of the polynomial representation of a 64-bit word W, and the polynomial $P = X$. The functions as are follows:

$$\sigma_0^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x) \end{array} \right.$$

$$\sigma_0^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{SHR}^{19}(x) \oplus \mathrm{SHR}^{61}(x) \oplus \mathrm{SHR}^6(x) \end{array} \right.$$

The Process. The sha-512 compression function is defined as below:

$$\mathrm{sha}-512 \left| \begin{array}{rcl} \{0,1\}^{512} \times \{0,1\}^{1024} & \longrightarrow & \{0,1\}^{512} \\ H, M & \longmapsto & C \end{array} \right.$$

Algorithm 10 Compression function of SHA-512

for $t \leftarrow 0, 15$ **do**
    $W_t \leftarrow M_t$
end for

for $t \leftarrow 16, 79$ **do**
    $W_t \leftarrow \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16}$
end for

$(a, b, c, d, e, f, g, h) \leftarrow (H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$

$(C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7) \leftarrow (a + H_0, b + H_1, c + H_2, d + H_3, e + H_4, f + H_5, g + H_6, h + H_7)$

Let M be the 1024-bit *message input* and H the 512-bit *hash input* (chaining input). These two inputs are represented respectively by an array of sixteen 64-bit words $M_0, \cdots, M_{15}$, and an array of eight 64-bit words $H_0, \cdots, H_7$. The 512-bit output value C is also represented as an array of eight 64-bit words $C_0, \cdots, C_7$.

Let H be the 512-bit *hash input* (chaining input) and M be the 1024-bit *message input*. These two inputs are represented respectively by an array of 8 64-bit words $H_0, \cdots, H_7$ and an array of 16 64-bit words $M_0, \cdots, M_{15}$. The 512-bit output value C is also represented as an array of 8 64-bit words $C_0, \cdots, C_7$.

During the process of compression, a sequence of 80 constant 64-bit words $K_0^{\{512\}}, ..., K_{79}^{\{512\}}$ is used. These 64-bit words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. We refer the

reader to [SHA95] for a table containing these constants.

Addition ($+$) is performed modulo 2^{64}.

The compression function is described in Algorithm 10.

Résumé

Cette thèse aborde la conception et les contre-mesures permettant *d'améliorer* le calcul cryptographique matériel léger. Parce que la cryptographie (et la cryptanalyse) sont de nos jours de plus en plus omniprésentes dans notre vie quotidienne, il est crucial que les nouveaux systèmes développés soient suffisamment robustes pour faire face à la quantité croissante de données de traitement sans compromettre la sécurité globale. Ce travail aborde de nombreux sujets liés aux implémentations cryptographiques légères. Les principales contributions de cette thèse sont :

Un nouveau système d'accélération matérielle cryptographique appliqué aux codes BCH. Les codes BCH sont largement utilisés dans les systèmes numériques, les dispositifs de mémoire et les réseaux informatiques. Parce que les codes BCH requièrent le calcul répété de réductions polynomiales modulo un polynôme constant, il est possible d'appliquer l'algorithme de réduction modulaire de Barrett pour obtenir une meilleure performance. Ce travail se concentre sur la $\ll$ polynomialization $\gg$ de l'algorithme de réduction modulaire de Barrett.

Réduction de la consommation des systèmes embarqués et SoCs. Nous proposons un système de gestion d'énergie intelligent capable d'activer et de désactiver des modules SoC ou des systèmes embarqués selon un modèle proposé. Ce modèle suppose que le système a deux états possibles, A et B, dans lequel le système est disponible pour répondre aux nouvelles demandes ou est en mode veille, respectivement. L'importance d'avoir un tel gestionnaire intelligent d'énergie est qu'il permet au système de s'activer ou de se désactiver lorsque la probabilité de recevoir une demande est suffisamment faible pour prendre un tel risque. Tout en réduisant les sanctions pour les absences de réponse, notre modèle assure un meilleur contrôle de la dissipation de puissance, ce qui se traduit par une meilleure consommation d'énergie.

Contre-mesures légères des attaques par canal auxiliaire applicables à l'algorithme de chiffrement reconfigurable AES. Nous proposons une protection légère contre les attaques par canal auxiliaire pour une architecture AES. Deux menaces physiques ont été analysés : attaques en consommation et attaque par fautes. L'architecture proposée tire profit de la structure des algorithmes AES pour obtenir une protection à faible coût contre ces attaques. Cela autorise une reconfiguration à chaud sans impacter durement la performance.

CSAC : Un pare-feu sécurisé sur la puce cryptographique. Les attaques telles que l'hijacking, l'extraction de données secrètes et le déni de service sont bien comprises et sont habituellement traitées dans les systèmes sur puces (SoC). Les SoCs comptent sur les pare-feux pour protéger le flux de données et maintenir les règles d'accès de la mémoire. Ce travail propose de protéger le chemin de reprogrammation du pare-feu et de ses règles d'accès mémoire, évitant un attaquant pouvant élever les privilèges ou modifier les autorisations d'accès à la mémoire.

Attaques par analyse fréquentielle. Nous introduisons l'Analyse de Corrélation de Fréquence Instantanée (CIFA). Ce travail repose sur l'utilisation de la fréquence instantanée au lieu de l'amplitude de puissance et du spectre de puissance dans l'analyse par canal auxiliaire. Par opposition à la fréquence constante utilisée dans la transformation de Fourier, la fréquence instantanée reflète des différences de phase locales et permet de détecter des variations de fréquence. Ces variations dépendent des données binaires traitées et sont donc utiles pour l'analyse cryptographique. La relation provient du fait que, après une baisse de puissance élevée, un temps plus long est nécessaire pour rétablir le courant à sa valeur nominale.

Un nouveau protocole à divulgation nulle de connaissance appliquée aux réseaux de capteurs sans fil. Nous introduisons un protocole d'authentification basée sur le paradigme de divulgation nulle de connaissance qui établit l'intégrité du réseau, et tire parti de la nature distribuée des nœuds de calcul pour soulager l'effort de calcul individuel. Cela permet à la station de base d'identifier les nœuds qui doivent être remplacés. Dans ce travail, nous avons décrit un protocole d'authentification Fiat-Shamir distribuée qui permet l'authentification du réseau en utilisant très peu de tours de communication, allégeant ainsi la charge des dispositifs à ressources limitées telles que des capteurs sans fil et d'autres nœuds. Au lieu d'effectuer l'authentification individuelle pour vérifier l'intégrité du réseau, notre protocole donne une preuve d'intégrité pour l'ensemble du réseau.

OMD : Un nouveau schéma de chiffrement authentifié. Nous proposons un nouveau chiffrement authentifié, soumis à examen dans la compétition CAESAR. Notre système, appelé Offset Merkle-Damgård (OMD), est un mode d'opération en compression à clé avec nonce cryptographique. Pour instancier le mode OMD, nous recommandons deux fonctions spécifiques de compression pour être saisies et utilisées dans l'OMD : les fonctions de compression des fonctions de hachage standard SHA-256 et SHA-512. Nous croyons qu'un régime AE dont la sécurité est prouvée par une réduction modulaire et facile à vérifier, ne s'appuyant que sur des hypothèses standard et largement vérifiées concernant ses primitives sous-jacentes, fournit de meilleures garanties sur sa sécurité par rapport à un schéma qui exige des propriétés fortes e idéalisées, ou ne possède pas de preuve de sécurité formelle.

Mots-clés: cryptographie symétrique, cryptographie légère, authentification, chiffrement, contre-mesure matérielle, cryptosystème matériel, attaque par canal auxiliaire, attaque par fautes.

Abstract

This thesis addresses lightweight hardware design and countermeasures to *improve* cryptographic computation. Because cryptography (and cryptanalysis) is nowadays becoming more and more ubiquitous in our daily lives, it is crucial that newly developed systems are robust enough to deal with the increasing amount of processing data without compromising the overall security. This work addresses many different topics related to lightweight cryptographic implementations. The main contributions of this thesis are:

A new cryptographic hardware acceleration scheme applied to BCH codes. BCH codes are widely used in digital systems, memory devices and computer networks. Because BCH codes required repeated polynomial reductions modulo a constant polynomial, it is possible to apply the Barrett's modular reduction algorithm to achieve better performance. This work focuses on the "polynomialization" of Barretts modular reduction algorithm.

Hardware power minimization applied to SoCs and embedded devices. A smart energy management system capable of turning on and off an SoC or embedded system according to a proposed model is discussed. This model assumes that the system has two possible states, A and B, in which the system is available to respond to new requests or is in idle mode, respectively. The importance of having such smart energy manager is to endow that the system can go idle when the incoming request probability is low enough to take such risk. While reducing unresponsiveness penalties, our model ensures a better power dissipation usage, which translates to a better battery lifetime.

Timing and DPA lightweight countermeasures applied to the reconfigurable AES block cipher. A lightweight side-channel protection for a proposed AES implementation is presented. Two physical threats were analyzed: power and fault attacks. The proposed architecture leveraged the AES algorithms structure to create low-cost protections against these attacks. This allowed very flexible runtime configurability without significantly affecting performance.

CSAC: A cryptographically secure on-chip firewall. Attacks such as hijacking, secret data extraction and denial of service are well understood and are usually addressed in system-on-chips. SoCs rely on firewalls to protect the data flow and maintain the memory region access rules. This work proposes to protect the firewall's reprogramming path and its memory region access rules, avoiding an attacker being able to escalate privileges or changing the memory access permissions.

Frequency analysis attack experiments. We investigate Correlation Instantaneous Frequency Analysis (CIFA), a technique that makes use of instantaneous frequency instead of power amplitude and power spectrum in side-channel analysis. By opposition to the constant frequency used in Fourier Transform, instantaneous frequency reflects local phase differences and allows to detect frequency variations. These variations depend on the processed binary data and are hence cryptanalytically useful. The relationship stems from the fact that after higher power drops more time is required to restore power back to its nominal value.

A new zero-knowledge protocol applied to wireless sensor networks. We introduce an authentication protocol based on the zero-knowledge paradigm 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. In this work we described 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.

OMD: A new authenticated encryption scheme. We propose a new authenticated cipher for consideration in the CAESAR competition. Our scheme, called Offset Merkle-Damgaård (OMD), is a keyed compression function mode of operation for nonce-based AEAD. To instantiate the OMD mode, we recommend two specific compression functions to be keyed and used in OMD, namely, the compression functions of the standard SHA-256 and SHA-512 hash functions. We believe that an AE scheme whose security is proved by a modular and easy to verify security reduction, only relying on some widely-verified standard assumption(s) on its underlying primitive(s), can get more confidence on its security compared to a scheme that demands strong and idealistic properties from its underlying primitive(s) or is not supported by a formal security proof.

Keywords: symmetric-key cryptography, lightweight cryptography, authentication, encryption, hardware design, hardware cryptosystem, hardware countermeasure, DPA, fault attacks.

Résumé

Cette thèse aborde la conception et les contremesures permettant d'améliorer le calcul cryptographique matériel léger. Parce que la cryptographie (et la cryptanalyse) sont de nos jours de plus en plus omniprésentes dans notre vie quotidienne, il est crucial que les nouveaux systèmes développés soient suffisamment robustes pour faire face à la quantité croissante de données de traitement sans compromettre la sécurité globale. Ce travail aborde de nombreux sujets liés aux implementations cryptographiques légères. Les principals contributions de cette thèse sont :

- Un nouveau système d'accélération matérielle cryptographique appliqué aux codes BCH ;
- Réduction de la consommation des systèmes embarqués et SoCs ;
- Contre-mesures légères des attaques par canal auxiliaire applicables à l'algorithme de chiffrement reconfigurable AES ;
- CSAC : Un pare-feu sécurisé sur la puce cryptographique ;
- Attaques par analyse fréquentielle ;
- Un nouveau protocole à divulgation nulle de connaissance appliquée aux réseaux de capteurs sans fil ;
- OMD : Un nouveau schéma de chiffrement authentifié.

Abstract

This thesis addresses lightweight hardware design and countermeasures to improve cryptographic computation. Because cryptography (and cryptanalysis) is nowadays becoming more and more ubiquitous in our daily lives, it is crucial that newly developed systems are robust enough to deal with the increasing amount of processing data without compromising the overall security. This work addresses many different topics related to lightweight cryptographic implementations. The main contributions of this thesis are:

- A new cryptographic hardware acceleration scheme applied to BCH codes;
- Hardware power minimization applied to SoCs and embedded devices;
- Timing and DPA lightweight countermeasures applied to the reconfigurable AES block cipher;
- CSAC: A cryptographically secure on-chip firewall;
- Frequency analysis attack experiments;
- A new zero-knowledge zero-knowledge protocol applied to wireless sensor networks;
- OMD: A new authenticated encryption scheme.

Mots Clés

Cryptographie symétrique, cryptographie légère, authentification, chiffrement, contre-mesure matérielle, cryptosystème matériel, attaque par canal auxiliaire, attaque par fautes.

Keywords

Symmetric-key cryptography, lightweight cryptography, authentication, encryption, hardware design, hardware cryptosystem, hardware countermeasure, DPA, fault attacks.