UNIVERSITE DE LIMOGES - ECOLE DOCTORALE
Faculté des Sciences et Techniques
Département Mathématiques Informatique

Thèse № 96325

Thèse pour obtenir le grade de Docteur

présentée par

Georges GAGNEROT

Étude des attaques et des contre-mesures associées sur composants embarqués.

Study of attacks on embedded devices and associated counter-measures.

Thèse dirigée par Christophe CLAVIER

Soutenue le 29 Novembre 2013
devant la commission composée de:

Rapporteurs	**Examinateurs**	**Invités**
Pierre-Alain Fouque	Jean-Louis Lanet	Benoit Feix
Louis Goubin		

Résumé

Les systèmes cryptographiques embarqués sont de plus en plus employé de nos jours allant de la carte bancaire, à la SIM qui nous identifie sur le reseau téléphonique, jusqu'à nos smartphones qui contiennent de nombreuses données sensibles voir secrètes. Ces systèmes doivent résister à un grand nombre d'attaques autant physiques que logiques.

Les travaux qui suivent décrivent dans un premier temps la cryptographie ainsi que les différents algorithmes classiquement utilisés. Un état de l'art sur les techniques d'attaques par canaux cachés est ensuite présenté, ces attaques sont dévastatrices car elle ne demandent pas forcément de détenir le dispositif, elles peuvent être effectuées à distance grace à du matériel performant. Elles consistent à étudier les émissions produites par un composant pendant qu'il traite un secret afin de retrouver ce dernier par des calculs statistiques ou de simples observations comme le temps de traitement. Nous présentons ensuite les attaques par fautes qui sont un autre type d'attaque qui menacent les composants sécurisé, il s'agit cette fois d'induire un comportement différent en introduisant des modifications extérieures au composant comme l'envoie de photons ou des décharges electriques afin de le mettre dans un état non standard et d'utiliser cela pour récuperer les secrets utilisés.

Puis dans une seconde partie nous proposons de nouvelles attaques et contremesures associées ainsi que de nouvelles implémentations d'algorithme pour securiser les calculs tout en les accélérant. Nous présentons également un simulateur d'émissions par canaux cachés ainsi qu'un simulateur de faute pour permettre d'évaluer plus rapidement et à moindre coût la sécurité des systèmes sécurisés.

Mots clés: sécurité embarquée, simulateur, analyse par canaux auxilliaires, algorithme d'exponentiation securisé.

Abstract

Embedded cryptographic systems are more and more used nowadays, starting from our credit card, to our SIM that identifies us on the GSM networks, or to our smartphones for example that store personal or secret data. Those systems have to be resistant against different kind of attacks, physical and logical.

Our work that we present thereafter first introduce cryptography with the different classic algorithms used. We proceed to present the state of the art on side-channel attacks, those attacks are very dangerous since they do not always need the physical possession of the target device, and can be done remotely with the good material. They are based on the study of the emissions produced by a component while it is using the secret stored in its internal memory to retrieve the secret through statistic computations or simple observations like the time it takes to treat data. We then propose a survey on faults attacks that also threaten security product. The idea is to produce a non standard behaviour by changing its environment, like sending photons or power discharge and then trying to recover the secrets.

In a second part, we propose new attacks, associated counter-measures and new algorithm to secure computation while making them also quicker. We present after a simulator of side channel emissions and faults attacks that allows us to assess the security of embedded systems quicker and cheaper than standard methods.

Keywords: embedded security, simulator, side-channel analysis, secure exponentiation algorithm.

Remerciements

Durant ces années passées en thèse, un nombre important de personnes m'ont aidé moi ou mes recherches, et la rédaction de ce mémoire est une formidable opportunité pour les remercier et leur prouver ma gratitude.

Je remercie en premier lieu mes encadrants qui m'ont permis de mener cette thèse à terme. Mon directeur de thèse Christophe Clavier qui n'a pas hésité à me prendre sous son aile et à me faire participer à ses réflexions toujours très pertinentes sur la sécurité. Qui m'a fait découvrir de nouvelles branches des mathématiques lors de débats endiablés, et m'a prouvé à maintes reprises son soutien sans failles. Benoit Feix qui m'a donné la chance de découvrir la sécurité embarquée, qui tout au long de ces 4 années passées à ses côtés n'a eu de cesse de m'étonner par sa modestie, ses connaissances, son expérience ainsi que ses qualités humaines. Sans eux cette thèse n'aurait sans doute jamais eu lieu.

Je remercie également les membres du jury Pierre-Alain Fouque, Louis Goubin ainsi que Jean-Louis Lanet qui ont accepté d'évaluer mon travail. Un grand merci pour vos relectures ainsi que vos remarques pertinentes.

Merci à mes amis Mylène Roussellet et Vincent Verneuil, qu'il fut agréable de travailler avec vous et Benoit dans le bureau le plus convoité d'Inside pour son ambiance chaleureuse et relaxée tout autant que pour le plaisir des papilles. Mes chefs Bruno Benteo et Marie Noelle Lemaire pour m'avoir permis d'évoluer au sein de vos équipes et de profiter de vos expériences si précieuses.

Enfin un grand merci à mes amis, ma famille, ma grand-mère tout particulièrement qui s'est donnée tant de mal pour moi, mon frère pour nos discussions underground, mes enfants et ma femme Nathalie dont le soutien et l'affection qu'ils me témoignent tous les jours me permettent d'avancer même lorsque les difficultés s'accumulent.

Quelques remerciements moins formels suivent, les intéressés sauront se reconnaitre.

1. Un appétit insatiable (pour la science). Roule plus vite que le train!
2. Je savais pouvoir compter sur toi pour la relecture de ce travail!
3. Moooouuuuuuuhhhh! Chef officiel des weekly en vieux ou en corse. J'attends toujours ses pancakes!
4. "China" y est plus belle.
5. Un grand merci, surtout pour vos actions.
6. La reine du gateau au chocolat!
7.
8. Harrap's de nos articles.
9. CodeBreaker lui doit beaucoup.
10. Lucky luke du résumé, spécialiste du tiramisu.
11. Attaques puissantes et rapides.
12. Merci pour cette année passée a vos côtés!
13. Maintient nos esprits vifs et nos coudes serrés.
14. Autoconstructeur, apiculteur, berger corse ou covoitureur; merci pour ton amitié et tes conseils.
15. Aime les fous rires, la prune mais pas l'autre.

A ma femme Nathalie et mes enfants <3

Contents

Introduction 1

I Background 2

1 Cryptography 3
1.1 Introduction . . . 3
1.2 Symmetric Ciphers . . . 3
1.2.1 Stream Cipher . . . 4
1.2.2 Block Ciphers . . . 5
1.3 Standard Systems . . . 10
1.3.1 Data Encryption Standard . . . 10
1.3.2 AES . . . 13
1.3.3 Attacks on Symmetric Ciphers . . . 17
1.4 Asymmetric Ciphers . . . 18

2 Side Channel Analysis 19
2.1 Introduction . . . 19
2.2 Attacks . . . 20
2.2.1 Simple Power Analysis . . . 20
2.2.2 Differential Power Analysis . . . 23
2.2.3 Distinguishers . . . 24
2.2.4 High Order Attacks . . . 26

3 Fault Analysis 27
3.1 Introduction . . . 27
3.2 Cosmic Rays . . . 27
3.3 Heat / Infrared Radiation . . . 27
3.4 Power Spike (glitch) . . . 28
3.5 Clock glitches . . . 28
3.6 Laser . . . 28
3.7 Electromagnetic Pulses . . . 29
3.8 Focused Ion Beams (FIB) . . . 29
3.9 Sample Attacks . . . 29
3.9.1 PIN Verification . . . 29
3.9.2 RSA CRT . . . 29
3.9.3 Dump of ROM . . . 30
3.9.4 Hack of the Playstation 3 . . . 30
3.10 Countermeasures . . . 31

II Contributions **32**

4 Our work **33**

5 Horizontal Correlation Analysis **35**
5.1 Introduction 35
5.2 Public-Key Embedded Implementations 36
5.3 Side-Channel Analysis 37
5.3.1 Background 37
5.3.2 Vertical and Horizontal Attacks Classification 39
5.4 Horizontal Correlation Analysis 41
5.4.1 Recovering the Secret Exponent with One Known Message Encryption 41
5.4.2 Practical Results 43
5.4.3 Comparing our Technique with the Big Mac Attack 46
5.4.4 Horizontal Analysis on Blinded Exponentiation 47
5.5 Countermeasures 48
5.5.1 Hardware Countermeasures 48
5.5.2 Blinding 49
5.5.3 New Countermeasures 49
5.6 Concerns for Common Cryptosystems 51
5.7 Square and Multiply Power Curves Examples 52
5.8 Examples of l and l^2 values 53

6 Improved Collision-Correlation Power Analysis on AES **54**
6.1 Introduction 54
6.2 Targeted Implementations 55
6.2.1 Blinded Lookup Table 55
6.2.2 Blinded Inversion Calculation 55
6.2.3 Measurements and Validation of Implementations 56
6.3 Description of our Attacks 57
6.3.1 The Collision-Correlation Method 57
6.3.2 Attack on the Blinded Lookup Table Implementation 59
6.3.3 Attack on the Blinded Inversion Implementation 62
6.4 Comparison with Second Order Analysis 64
6.5 Countermeasures 65
6.6 Masked Inverse 65
6.7 Practical Results on Real Curves 66

7 ROSETTA **67**
7.1 Introduction 67
7.1.1 Roadmap 68
7.2 Background 68
7.2.1 RSA Implementation 68
7.2.2 Attacks Background 71
7.2.3 Big Mac Extension using Collision Correlation 72
7.3 ROSETTA: Recovery Of Secret Exponent by Triangular Trace Analysis 73
7.3.1 Attack Principle 73
7.3.2 Euclidean Distance Distinguisher 74
7.3.3 Collision-Correlation Distinguisher 75
7.4 Comparison of the Different Attacks 75
7.5 Countermeasures 78

8 Passive and Active Combined Analysis **80**
8.1 Introduction . . . 80
8.2 Side Channel and Fault Analysis Background . . . 81
8.2.1 Side Channel Analysis . . . 81
8.2.2 Fault Analysis . . . 82
8.3 Targeted AES Implementation . . . 83
8.4 Passive and Active Combined Attack on Masked AES . . . 84
8.4.1 Fault Model . . . 84
8.4.2 Attack on the First Key Addition . . . 85
8.5 Countermeasures . . . 87
8.5.1 Inverse computation . . . 87
8.5.2 Duplicated Rounds . . . 88
8.5.3 Data error . . . 88
8.5.4 Checksums . . . 88
8.6 Passive and Active Combined Attack on Masked AES with Safe Errors 89
8.6.1 Countermeasures . . . 89
8.7 Targeted AES Implementation . . . 90

9 Square Always **91**
9.1 Introduction . . . 91
9.2 Background on Exponentiation on Embedded Devices . . . 92
9.2.1 Square-and-Multiply Algorithms . . . 92
9.2.2 Side-Channel Analysis on Exponentiation . . . 93
9.2.3 Distinguishing Squarings from Multiplications . . . 94
9.3 Square Always Countermeasure . . . 95
9.3.1 Principle . . . 95
9.3.2 Atomic Algorithms . . . 96
9.3.3 Performance Analysis . . . 98
9.3.4 Security Considerations . . . 99
9.4 Parallelization . . . 99
9.4.1 Parallelized Algorithms . . . 100
9.4.2 Cost of Parallelized Algorithms . . . 101
9.5 Practical Results . . . 102
9.6 Cost of Algorithm 9.9 . . . 103

10 Simulation **107**
10.1 Introduction . . . 107
10.2 Power Simulation . . . 108
10.2.1 Core simulation . . . 108
10.2.2 Enhancements . . . 109
10.2.3 Usage . . . 110
10.2.4 Showcase . . . 110
10.3 Fault Simulation . . . 116
10.3.1 Results . . . 118
10.3.2 Showcase . . . 119

Conclusion **123**

List of Figures

1.1 Cipher overview . . . 4
1.2 Symmetric Cipher . . . 6
1.3 Feistel Network . . . 9
1.4 DES Overall Structure . . . 11
1.5 Feistel Function Figure . . . 12
1.6 DES Key Schedule . . . 13
1.7 AES SubBytes . . . 15
1.8 AES ShiftRows . . . 15
1.9 AES MixColumns . . . 16
1.10 AES AddRoundKey . . . 17
1.11 Asymmetric Cipher . . . 18

2.1 Good Pin . . . 21
2.2 Superposition of Good and Wrong PIN verification . . . 21
2.3 Contactless SPA . . . 22
2.4 Contactless SPA filtered . . . 22
2.5 DPA sample . . . 24
2.6 CPA . . . 25

5.1 Vertical Side Channel Analysis . . . 40
5.2 Horizontal side-channel analysis . . . 41
5.3 Beginning of a long integer multiplication power curve, lines delimitate each $C_{i,j}^k$ . . . 43
5.4 Vertical CPA on value y_j. . . . 44
5.5 Vertical CPA on value $x_i \times y_j$. . . . 44
5.6 Horizontal CPA on value $a_i \times m_j$. . . . 45
5.7 Horizontal CPA on value m_j. . . . 46
5.8 Power curve of a leaking square and multiply algorithm . . . 52
5.9 Power curve of an atomic square and multiply algorithm . . . 52

6.1 General description of the collision-correlation attack . . . 57
6.2 Collision between the computation of two S-Boxes on bytes 4 and 9 on the blinded lookup table implementation . . . 59
6.3 Correlation curves obtained for a message giving one collision (black curve) . . . 60
6.4 Correlation curves obtained for a message giving no collision . . . 60
6.5 Correlation peak on real curves when a collision occurs (black curve) 61
6.6 No correlation peak occurs on real curves when intermediate data differ . . . 61
6.7 Collision between the input and the output on byte 3 of the blinded inversion I' (values 0 and 1 lead to a collision) . . . 62

6.8 Collision-correlation curves in the pseudo-inversion of the first byte in $\mathrm{GF}(2^8)$ . . . 63
6.9 Success rates of different simulated second-order attacks . . . 64

7.1 Atomic multiply-always side-channel leakage . . . 70
7.2 Horizontal side-channel analysis on exponentiation . . . 72
7.3 Success rate of the different attacks with no noise. . . . 77
7.4 Success rate of the different attacks with a strong noise, $\sigma = 7$. . . . 77

8.1 Secure HODPA Implementation . . . 90

10.1 Simulated Pin on an ARM architecture . . . 111
10.2 Simulated Pin on a x86_64 architecture . . . 111
10.3 Simulated AES on an ARM architecture . . . 112
10.4 Simulated AES on a x86_64 architecture . . . 112
10.5 Simulated DES on an ARM architecture . . . 113
10.6 Simulated DES on a x86_64 architecture . . . 113
10.7 Simulated RSA on an ARM architecture . . . 114
10.8 Simulated RSA on a x86_64 architecture . . . 114
10.9 DPA simulation sample . . . 115
10.10 CPA simulation sample . . . 116
10.11 CPA simulation sample with good guess highlighted . . . 116
10.12 Pin Fault Simulation . . . 121
10.13 AES Fault Simulation . . . 122

List of Tables

1.1 AES S-Box 8

5.1 Examples of n, t, and l values with the number of available segments l^2 53

7.1 Success rate with a null noise, $\sigma = 0$ 76
7.2 Success rate with a moderate noise, $\sigma = 2$ 77
7.3 Success rate with a strong noise, $\sigma = 7$ 77

8.1 Performance (cyles) and memory costs (bytes) for AES_{DPA} and $\text{AES}_{\text{HODPA}}$ implementations 84
8.2 Performance and memory costs (bytes) for two Fault Analysis resistant implementations 88

9.1 Comparison of the expected cost of SPA protected exponentiation algorithms (including the multiply always which is not immune to the attack from [7]) 98
9.2 Comparison of the expected cost of parallelized exponentiation algorithms 103
9.3 On chip comparison of the Montgomery ladder and square always algorithms 103

List of Algorithms

1.1 Simple Key Whitening 7
1.2 Feistel Encryption Algorithm 8
1.3 AES algorithm 14
1.4 Double Encryption Attack 18
2.1 Differential Power Analysis 24
5.1 Long Integer Multiplication 36
5.2 Square and Multiply Exponentiation 37
5.3 LIM with lines randomization and blinding 50
5.4 LIM with lines and columns randomization 51
7.1 Atomic Multiply-Always Exponentiation 69
7.2 Schoolbook Long-Integer Multiplication 70
8.1 The attack algorithm on key byte K_n 87
9.1 Left-to-Right Square-and-Multiply Exponentiation 92
9.2 Right-to-Left Square-and-Multiply Exponentiation 92
9.3 Montgomery Ladder Exponentiation 94
9.4 Left-to-Right Multiply Always Exponentiation 94
9.5 Left-to-Right Square Always Exponentiation with (Equation 9.1) 97
9.6 Right-to-Left Square Always Exponentiation with (Equation 9.2) 98
9.7 Right-to-Left Parallel Square Always Exponentiation with (Equation 9.2) 100
9.8 Right-to-Left Atomic Parallel Square Always Exp. with (Equation 9.2) . 101
9.9 Right-to-Left Generalized Parallel Square Always Exp. with (Equation 9.2) 102
10.1 CPU simulation 108
10.2 Fault Algorithm 118
10.3 Find ROM Dump 119

Glossary

AES Advanced Encryption Standard. 6, 10, 13, 14, 16, 27, 33, 34, 80–85, 87, 88, 110, 115, 121, 123, 124

CFA Correlation Fault Analysis. 80, 82–84, 89, 124

CPA Correlation Power Analysis. 19, 24, 35, 38, 39, 41, 46–48, 51, 81, 83, 115, 123, 124

CRT Chinese Remainder Theorem. 47, 51, 123

CSCA Correlation Side-Channel Analysis. 80

DEMA Differential ElectroMagnetic Analysis. 35

DES Data Encryption Standard. 5, 6, 8, 10, 12, 13, 19, 23, 27, 81, 82, 109, 110

DFA Differential Fault Analysis. 27, 82, 88

DPA Differential Power Analysis. 19, 23, 24, 26, 35, 38, 39, 41, 51, 65, 81, 83, 115

DSA Digital Signature Algorithm. 36, 51, 123

DSCA Differential Side-Channel Analysis. 35, 38

ECC Elliptic Curve Cryptography. 18, 36, 51

ECDH Elliptic Curve Diffie–Hellman. 36

ECDSA Elliptic Curve Digital Signature Algorithm. 36

GSM Global System for Mobile. 1

HODPA High Order Differential Power Analysis. 26, 81, 83, 84, 86, 88, 89, 124

MIA Mutual Information Analysis. 19, 25, 26, 35

NBS National Bureau of Standars. 10

NIST National Institute of Standards and Technology. 5, 10, 13

NSA National Security Agency. 10, 13

OTP One-Time Pad. 4

PACA Passive and Active Combined Attack. 80, 124

PK Public Key. 3, 18

RSA Rivest Shamir Adleman. 18, 19, 21, 27, 35, 36, 41, 47, 51, 80, 82, 109, 110, 123

SCA Side-Channel Analysis. 1, 19, 20, 35, 47, 80, 107, 124

SCARE Side Channel Analysis for Reverse Engineering. 19

SEMA Simple ElectroMagnetic Analysis. 35

SK Secret Key. 3, 18

SPA Simple Power Analysis. 19, 20, 22, 23, 35, 37, 41

SPN Substitution Permutation Networks. 6, 7, 13

SSCA Simple Side-Channel Analysis. 35, 37, 38, 49, 52, 81

Introduction

Nowadays embedded devices are widespread in consumer, industrial, commercial and military applications. More and more devices are appearing each day with more processing power while standing in the pocket of the consumer. Smartphones are more powerful than computers from few years ago. Those devices used by an always increasing population, and soon everyone will be using them, storing in the cloud personal informations they would not want to be exposed, with growing memory and storage one will be able to store his whole life digitalised on those platform. Those devices usually contain secret, like our bank key for credit card, our client id for Global System for Mobile (GSM) networks or some login and password for mail connection for example in smart-phones. Exchanging such data between two parties without allowing anyone to spy on the conversation is a thousands years old problem, addressed by cryptography, that arose in ancient Egypt with scribes writing non-standard hieroglyphs to pass secret messages. This science evolved up to the current cryptography with the addition of other needs like proving his identity or signing contents and finding different ways to exchange the key of the cipher to prevent its discovery.

Recent embedded devices are composed of hardware blocks, made from transistors linked together with multiple level of lines, called intellectual property blocks (IP). The security of those IP has been the subject of intensive research after the rise of so called Side-Channel Analysis (SCA) attacks that were introduced by Kocher et al. in 1998 [61] and started this new field of research in the applied cryptography area. Those kinds of attacks are non intrusive as they only monitor the environment of the target to deduce internal workings and potentially retrieve the hidden secrets. Those systems are also vulnerable to so called Fault Attacks where an attacker voluntary modify the processing of a component in order to retrieve information about the target. The combination of those two different threats leads to very powerful attacks.

The first part of my PhD will be an introduction to the state of the art for those SCA and faults Attacks. Then in a second part, we will present our contributions to the embedded security. Those new attacks were presented in various recognized embedded security workshops. In the last contribution chapter we will propose a new cheap and quick way to assess the security of a product against faults and SCA attacks through software simulation.

Then we will make a summary of our contributions and conclude.

Part I

Background

Chapter 1

Cryptography

1.1 Introduction

Cryptography can be defined as the science of protecting sensible information. Cryptography is intimately connected to a second term, namely cryptanalysis, which aims at breaking cryptographic means and reading the secret information. Often, the term cryptology is used to involve both of these aspects. Basically, cryptography aims at providing four principal security aspects: privacy or confidentiality, authenticity, integrity, and non-repudiation.

- Confidentiality: It authorizes the access for only allowed parties (or users).
- Authenticity: It allows different parties to identify each other (source/destination).
- Integrity: It guarantees that the message is properly transmitted from the source to the destination.
- Non repudiation: It allows controlling message acknowledgment. It provides proof of the integrity and origin of data.

Depending on the system to secure and the nature of the secret information, usually one or all of the mentioned security aspects, that Cryptography can provide, are required.

1.2 Symmetric Ciphers

Symmetric-key algorithms are a class of algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext.

Secret Key (SK) cryptography are defined as the class of ciphers that use an unique key for both encryption and decryption. This key has to be shared between the source and the receiver. For this reason, SK cryptography is also called Symmetric encryption, whereas Public Key (PK) Cryptography is called asymmetric. Obviously, the main issue with symmetric encryption relies on the distribution of the secret key. Secret-key cryptography schemes are classified into stream ciphers and block ciphers.

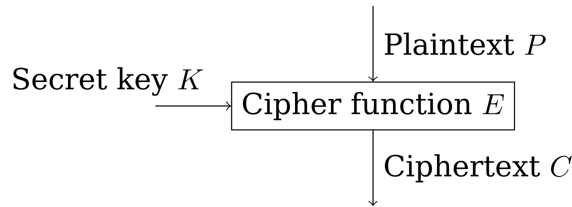


Figure 1.1: Cipher overview

1.2.1 Stream Cipher

A stream cipher is a symmetric key cipher where plaintext is combined with a (pseudo) random stream named keystream. Each plaintext digit is encrypted with the corresponding digit of the keystream, providing a digit of the cyphertext stream. Such ciphers are also named state cipher, as the encryption of each digit is dependent on the current state. In practice, a digit is typically a bit and the combining operation an exclusive-or (xor). The pseudorandom keystream is typically generated from a random seed value using digital shift registers. The seed value serves as the cryptographic key for decrypting the ciphertext stream.

Stream ciphers represent a different approach than block ciphers. Block ciphers operate on large blocks of digits with a fixed transformation. In some modes of operation, a block cipher primitive can be used in such a way that it acts effectively as a stream cipher. Stream ciphers usually execute at a speed greater than block ciphers and are simpler in term of hardware complexity. However, stream ciphers can be susceptible to serious security problems if used incorrectly, the same starting state (seed) must never be used twice.

The One-Time Pad (OTP), also known as Vernam Cipher, is proven to be an unbreakable cipher. Stream ciphers can be viewed as trying to approximate its action. A one-time pad uses truly random digits for its keystream, the keystream is then combined with the plaintext to form the resulting ciphertext. Claude E. Shannon proved in 1949 that is was secure. A drawback though is that the keystream has to be of at least the same length than the plaintext! Since the receiver has to know both to recover the message, it usually has to be delivered by two different paths, since if both would use the same path an eventual interception would break the system. The system is consequently very hard to implement in practice and as a result it is not widely used, except for the most critical applications and some governmental transmissions.

Classical Stream Cipher use smaller key size like 8 or 16 digits long (64 or 128 bits). The key is derived in different ways and can be combined too with the plaintext to generate a pseudo random keystream that is used then like the secure OTP. However, the proof of security associated with the previous algorithm does not hold anymore and such a Stream Cipher can be insecure.

Linear Feedback Shift Register-based Stream Ciphers

Binary stream ciphers are usually made using LFSRs (Liner Feedback Shift Registers) since they are really efficient hardware blocks that can be synthetized and

optimized very efficiently. They can also be mathematically analyzed very well. LFSRs as such are not enough to provide good security, they usually have to be coupled with other mechanism in order to increase their security.

Security

A basic necessary condition for a stream cipher to be secure is to have a large period for its keystream. The internal state of the keystream must be impossible to recover too. Usually cryptographers requires for any key and any ciphertext the following properties.

- The absence of any bias of the keystream that could help attackers to distinguish the keystream from a random distribution.
- No known relationships between the key or related nonces and the keystream.

For some stream ciphers though weak keys do exist that do not exhibit such properties.

Cryptographers do not need to actually break the cipher, but only to exhibit weaknesses, in order to find an attack. To be successfully accepted an attack complexity has only to be less than the complexity the exhaustive key search.

Securely using a synchronous stream cipher requires that one never reuses the same keystream twice; that generally means a different nonce or key must be supplied to each invocation of the cipher. Application designers must also recognize that most stream ciphers do not provide authenticity, only privacy: encrypted messages may still have been modified during the transmission. Short periods for stream ciphers have been a practical concern. For example, 64-bit block ciphers can be used to generate a keystream in output feedback (OFB) mode but in that case by using the birthday theorem we get a probability of $1/2$ to get 2 computations with the same output for 2^{32} tries. The period of such a keysteam would then be 2^{32} on average.

1.2.2 Block Ciphers

A block cipher uses a fixed algorithm operating on a group of bits of fixed size. The size of such a group of bit - called block - is dependent on the cipher. The transformation is fixed and dependent of the key used. The block ciphers are important cryptographic primitives in the design of secure protocols nowadays and are widely used for encryption. The design of block ciphers is based on the concept of an iterated product ciphers, such ciphers were analyzed by Claude Shannon in his Communication Theory of Secrecy Systems to improve security by combining simple operations such as substitutions and permutations [94].

The publication of the Data Encryption Standard (DES) by the National Institute of Standards and Technology (NIST) in 1977 was a fundamental move in the understanding of modern block cipher design. That algorithm has been thoughtfully studied and a palette of attack techniques that a block cipher has to be secure against emerged from that in addition from the robustness to brute force attacks.

Multiple iterations (rounds) are usually required by iterated product ciphers to do the encryption, each round using a roundkey derived from the master key. Feis-

tel networks (section 1.2.2), named after their creator Horst Feistel, used in the DES cipher are a widespread implementation of such ciphers. The Advanced Encryption Standard (AES) algorithm on another hand uses Substitution Permutation Networks (SPN) (section 1.2.2).

We can cite some known block ciphers: DES, AES, Twofish, IDEA, RC5, Serpent, Blowfish.

Definition

A block cipher consists essentially of two paired algorithms

1. An encryption algorithm E
2. A decryption algorithm $D = E^{-1}$

Both algorithms accept two kind of inputs

1. A block input of size n bits.
2. A key of size k bits, K named the key.

The result of E and D are two blocks of n bits respectively C the ciphertext and P the plaintext.

A block cipher is defined by its encryption function

$$E_K(P) := E(K, P) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n \tag{1.1}$$

For each K, the function $E_K(P)$ is required to be an invertible mapping on $\{0,1\}^n$. The inverse for E is defined as a function

$$E_K^{-1}(C) := D_K(C) = D(K, C) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n \tag{1.2}$$

taking a key K and a ciphertext C to return a plaintext value P, such that

$$\forall P \forall K : D_K(E_K(P)) = P.$$

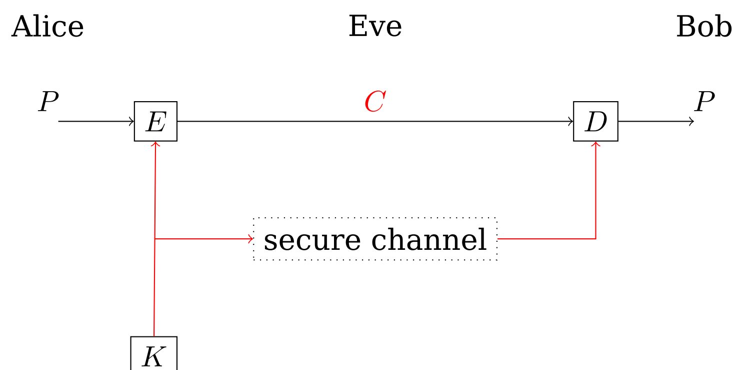


Figure 1.2: Symmetric Cipher

Let us take the AES block cipher as example, its encryption algorithm takes a 128-bit block of plaintext as input and outputs a 128-bit block of ciphertext. The transformation is dependent on the secret key that controls some internal state

of the cipher. The decryption is similar, it takes the same block size as input and output. Assuming the same secret key is used the decryption of the ciphertext returns the plaintext. For any 2^k key K, E_K is a permutation over the set of input blocks. Each key selects one permutation from the possible set of $(2^n)!$.

Iterated block ciphers The vast majority of block cipher algorithm belongs to the iterated block ciphers class, meaning that their transformation is done via repeated application of an invertible transformation known as the round function. Each iteration is commonly referred to as a round. Usually different round keys K_i are used for the round function f on different rounds. Those round keys are derived from the original key K through a commonly called key expansion algorithm. Let r be the number of round, P_i the entry of the ith round, with P_0 the plaintext and P_r the ciphertext.

$$P_i = f_{K_i}(P_{i-1}) \tag{1.3}$$

Frequently, key whitening is used in addition to this. At the beginning and the end, the data is modified with key material for instance it can be $\oplus$, addition or Subtraction operation. It would give by example:

Alg. 1.1 Simple Key Whitening

```
1: P_0 = P ⊕ K_0
2: for i = 1 to r do
3:     P_i = f_{K_i}(P_{i-1}) ;
4: end for
5: C = P_r ⊕ K_{r+1}
6: return (C)
```

Substitution-Permutation Networks Another important type of Iterated block ciphers is SPN. It works by taking a block of plaintext and a key as inputs, and applying rounds of alternating substitution and permutation stages to produce the ciphertext. The substitution stage should be non-linear and mixes the key bits with the plaintext to create Shannon's confusion. The permutation state is used to create diffusion and to dissipate redundancies. A Substitution box named S-Box substitutes a small block of input data replacing it with another block of output bits. The S-Box must be bijective to ensure invertibility and decryption. To be qualified secure, changing one bit in the input data should change about half of the output bits. This principle is known as the avalanche effects. For example you can have a look at Table 1.1.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 1.1: AES S-Box

A permutation shuffles all S-Box output bits to produce the input of the next round. A good permutation should ensure that the output bits of any S-Box are distributed to as many S-Box inputs of the next round.

The round key is usually obtained by combining some group operation like $\oplus$ or shifts on itself. Decryption is done simply by reversing the process using the inverse S-Box $^{-1}$ and the inverse permutation.

Feistel ciphers The plaintext entering such a cipher is split into two halves of the same size. The round function is then applied on one half using a subkey then XORed with the other half. The function then swaps both halves.

Let F be the round function and let $K_0, K_1, \ldots, K_n$ be the round keys for rounds $1, \ldots, n$ respectively. Then the basic operation is performed as follow:

Alg. 1.2 Feistel Encryption Algorithm

1: Split the plaintext into two block of same length L_0 and R_0
2: **for** $i = 1$ **to** n **do**
3: $\quad L_{i+1} = R_i$
4: $\quad R_{i+1} = L_i \oplus \mathrm{F}(R_i, K_i)$
5: **end for**
6: **return** (R_{n+1}, L_{n+1})

The decryption of a ciphertext (R_{n+1}, L_{n+1}) is accomplished by computing the same algorithm in reverse order.

One advantage of the Feistel model compared to a substitution-permutation network is that the round function F does not have to be invertible. The best known usage of a Feistel cipher is the one used in the DES detailed in Figure 1.5.

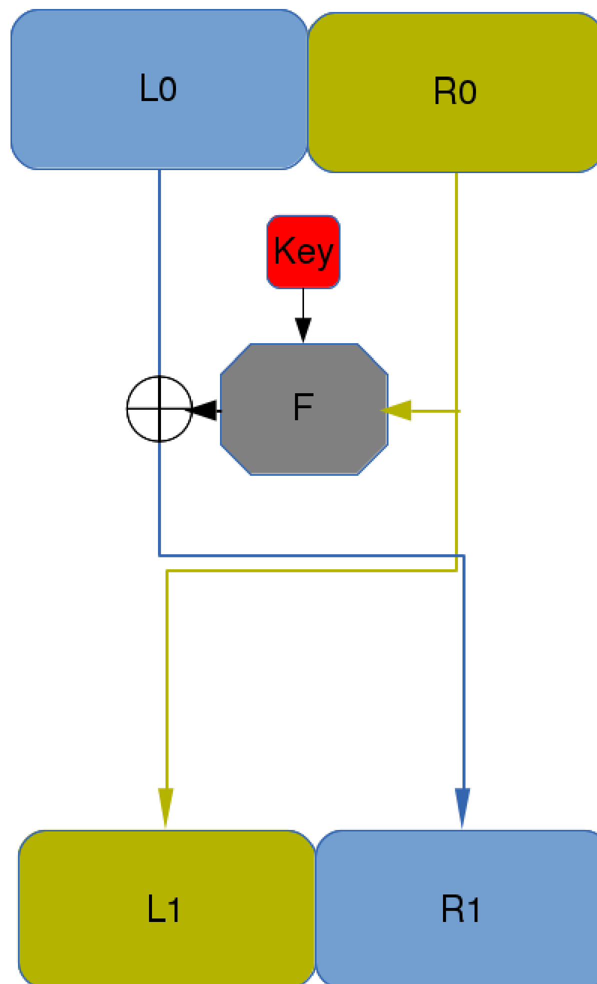


Figure 1.3: Feistel Network

Mode of Operations

A block cipher is only suitable to securely transform either by encryption or decryption **one** block. A mode of operation describes how to repeatedly apply a cipher single-block operation to securely transform amounts of data larger than one block.

- Electronic Code Book (ECB) mode The same block of ciphertext is always generated for a given block of plaintext and a given key. ECB mode is often used for small sizes of input block, such as encrypting and protecting secret keys. The disadvantage of this method is that it does not hide data patterns well since the same block is always encrypted to the same ciphertext. In some senses, it does nOT provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.
- Cipher Block Chaining (CBC) mode This mode makes the being processed ciphertext block dependent on plaintext blocks previously processed. Actually, the plaintext block currently processed is $\oplus$ with the previous ciphertext block before being encrypted. Besides, an initialization vector must be used for the initial block, in order to guarantee the uniqueness of the message. Decrypting with the incorrect IV causes the first block of plaintext to be cor-

rupted but subsequent plaintext blocks will be correct. This is because a plaintext block can be recovered from two adjacent blocks of ciphertext. Decryption can be parallelized. Note that a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext, but the rest of the blocks remains intact.

- Cipher Feedback (CFB) mode Data are encrypted in new blocks smaller than the initial block size. It can recover from an error received on stream decryption after a certain number of blocks. That property makes it a so called self-synchronizing ciphers.
  CFB has two advantages over CBC mode:
  - the block cipher is only ever used in the encrypting direction
  - the message does not need to be padded to a multiple of the cipher block size
- Output Feedback (OFB) mode It operates in a manner to guarantee the uniqueness of generated ciphertext blocks.

1.3 Standard Systems

1.3.1 Data Encryption Standard

The DES used to be the Number one cryptographic algorithm for electronic data. It had a high influence in the advancement of the study of modern cryptography in the academic world. Thanks to it new attacks were found, and cryptographers have a better understanding on what property are mandatory to a block cipher in order to be secure! It appeared in the early 1970s at IBM and is based on an design made by Horst Feistel [38]. The algorithm was submitted to the National Bureau of Standars (NBS) after an invitation to propose the algorithm that would be used for the protection of sensitive, unclassified government data. After consultation of the National Security Agency (NSA), the NBS selected a slightly modified version of the algorithm which was then published as an official FIPS (Federal Information Processing Standard) for the United States or America (USA) in 1977.

The publication of an encryption standard algorithm recognized by the NSA resulted in its quick adoption and widespread scrutiny. Since some design elements were classified, controversies arose quickly. The relatively short key length [1] of the block cipher design nourished suspicions about an eventual backdoor put in by the NSA in the algorithm in order to be able to crack it. The intense academic studies of the algorithm led over time to a much better understanding of block cipher and their cryptanalysis. Nowadays DES is considered to be insecure, for example in 1999 a collaboration broke a DES key in a little more than 22 hours. Some analytical results [68] demonstrate theoritical weaknesses in the cipher but none were implemented more efficiently than the brute force attack (first implemented by the Electronic Frontier Fondation (EFF) in 1998). The cipher has been superseded by the AES and DES was withdraw by the NIST.

[1] See the interesting transcript about DES modifications made by the NSA.

Overall Structure

The algorithm overall structure is shown on Figure 1.4.

- There are 16 rounds of processing
- Use a Feistel Network (section 1.2.2) with two halves of 32-bit each.
- There is also initial and final permutation, termed IP and FP $IP = FP^{-1}$. Those fonctions have no cryptographic significance and were introduced mostly to facilitate the task of 8-bit based hardware in 1970 [2].

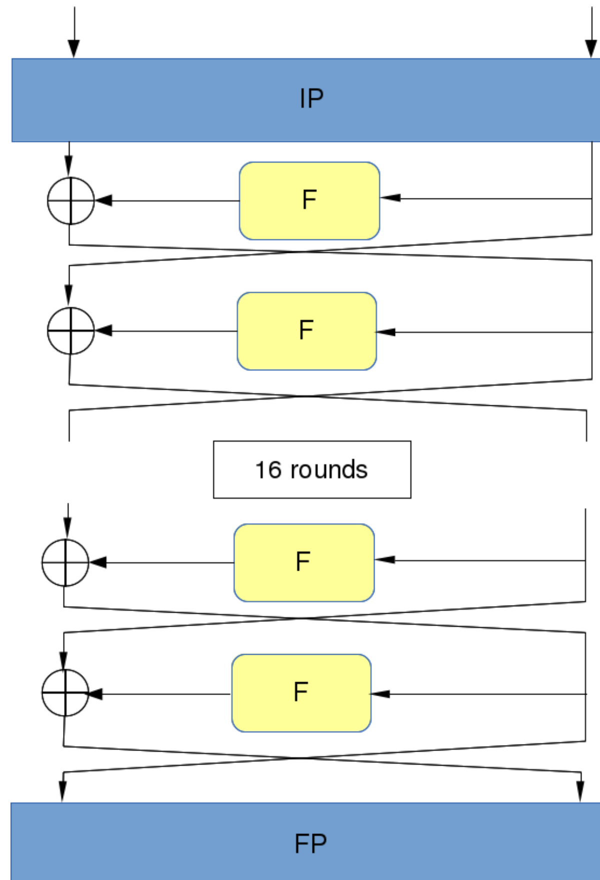


Figure 1.4: DES Overall Structure

Feistel Function

The F function, depicted on Figure 1.5, operates on half a block (32 bits) at a time and consists of four stages:

[2] Benefits of the permutation tables in DES

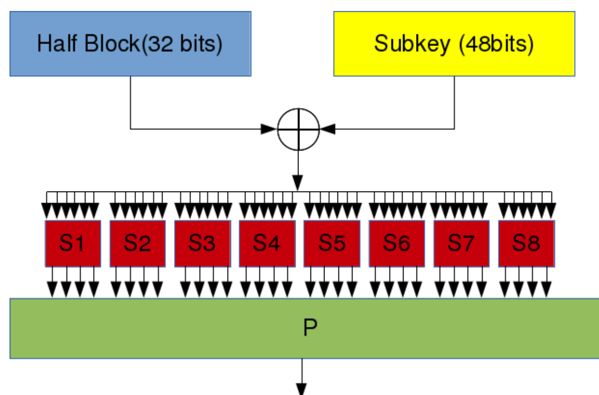


Figure 1.5: Feistel Function Figure

Expansion the 32-bit half-block are expanded to 48 bits using an expansion permutation, denoted E in the diagram, by duplicating half of the bits. The output consists of eight 6-bit parts, each containing a copy of 4 corresponding input bits, and a copy of the immediately adjacent bit from each of the input pieces to either side.

Key mixing the result is combined with a round subkey through an XOR operation. Sixteen 48-bit subkeys are derived from the master key using the key schedule (section 1.3.1).

Substitution the block is divided into eight 6-bit parts before entering the S-Boxes. Each of the eight S-Boxes replaces its six input bits with four output bits according to a lookup table. The S-Boxes provide the core of the security of DES. Without them, the cipher would be linear, and trivially breakable. The S-Boxes are non-linear.

Permutation the 32-bit output from the S-Boxes are mixed according to a fixed permutation. The permutation is designed in such a way that each S-Box's output bits are spread over 4 S-Box in the next round. The S-Boxes, permutation and E expansion provide the so-called "confusion and diffusion" respectively, a concept that Claude Shannon identified as necessary for a secure cipher.

DES Key Schedule

Figure 1.6 describes the algorithm which generates the subkeys known as the Key Schedule. At first 56 bits of the key are chosen by the Permuted Choice 1 (PC1) then the left bits are discarded. The bits are then split in two 28-bit halves treated separately. Both halves are rotated left by one or two bits (depending on the round) and 48 bits subkey are chosen by Permuted Choice 2 (PC2), 24 bits from the left part, and 24 from the right one. The rotations "$<<<$" make that different set of bits are used for each subkey. Each bit of the master key is used approximatively on 14 of the 16 subkeys. For decryption, the key schedule does not change, only the order of the subkeys that are used in reverse order compared to encryption.

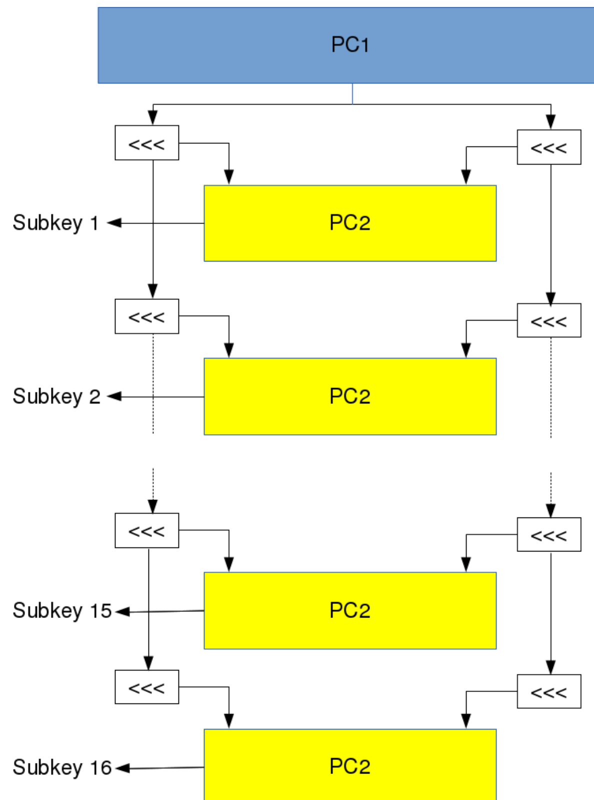


Figure 1.6: DES Key Schedule

1.3.2 AES

In 2001 the NIST published the specification of the new encryption cipher for electronic data named the AES. It was named Rijndael after both its inventors, Joan Daemen and Vincent Rijmen who submitted it during the five years NIST selection process where fifteen competing designs were proposed and evaluated. AES after being adopted by the USA government is used worldwide nowadays, superseding the DES presented in subsection 1.3.1. The cipher uses symmetric keys. It became effective as a federal government standard in 2002. AES is provided in the ISO/IEC 18033-3 standard. It is available in many different libraries, and is the first publicly accessible cipher approved by the NSA for top secret information when used in an NSA approved cryptographic module. The AES standard is a variant of Rijndael with a block size fixed to 128 bits.

Overall Structure

AES is based on a design principle known as a SPN (section 1.2.2) not on Feistel Networks and fast in both software and hardware. Its key size can be 128, 192, or 256 bits. In the Rijndael specification block and key sizes may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits [3]. AES operates on a 4x4 matrix of bytes, named its state. Most AES computations are done in a finite field. The number of rounds of the AES cipher is specified by the size of the secret key, defined as follows:

[3] Rijndael Specifications

- 10 cycles of repetition for 128-bit keys.
- 12 cycles of repetition for 192-bit keys.
- 14 cycles of repetition for 256-bit keys.

Each round consists of several steps containing four stages including one that depends on the encryption key. Reverse rounds are alse provided to transform back the ciphertext back to the plaintext assuming the same encryption key is used.

The AES algorithm is composed of 5 states:

- Key Expansion: round keys are derived from the cipher key using Rijndael's key schedule.
- SubBytes: a non-linear substitution step. Each byte is replaced with another thanks to a table.
- ShiftRows: a transposition step. Each row of the state is shifted circularly a certain number of times.
- MixColumns: a mixing operation. It combines the four bytes in each column.
- AddRoundKey: round keys are derived from the cipher key using Rijndael's key schedule.

The algorithm is the following one:

Alg. 1.3 AES algorithm

```
Key Expansion
AddRoundKey
for all Rounds do
    SubBytes
    ShiftRows
    if (round != lastRound) then
        MixColumns
    end if
    AddRoundKeys
end for
```

AES is well suited for hardware design, only the S-Box part being non logical operation.

SubBytes

In this step each byte of the state matrix is replaced with another one using a S-Box table shown on Table 1.1. This operation gives the non-linearity needed by the cipher to be secure. The S-Box used is derived from the multiplicative pseudo-inverse over $GF(2^8)$, since its known to have good non-linearity properties and combined with an affine transformation (invertible) to prevent attacks based on algebraic properties. The S-Box is chosen to avoid any fixed or opposite fixed points.

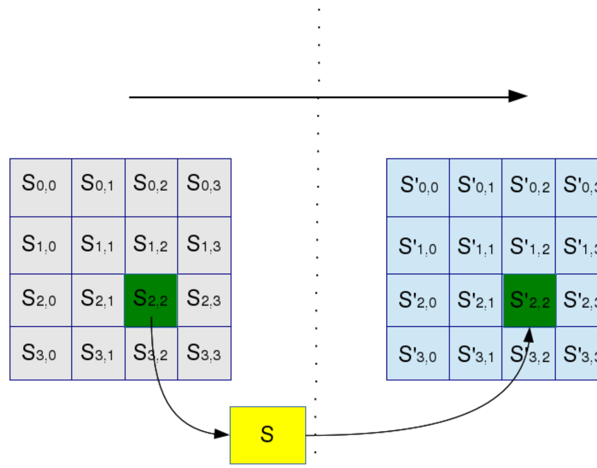


Figure 1.7: AES SubBytes

ShiftRows

This step operates on the rows of the state. The bytes are cyclically shifted for each row by a certain offset. The first row is left unchanged. The row n are shifted by $n-1$ to the left circularly. This steps tries to provide columns linearly independent.

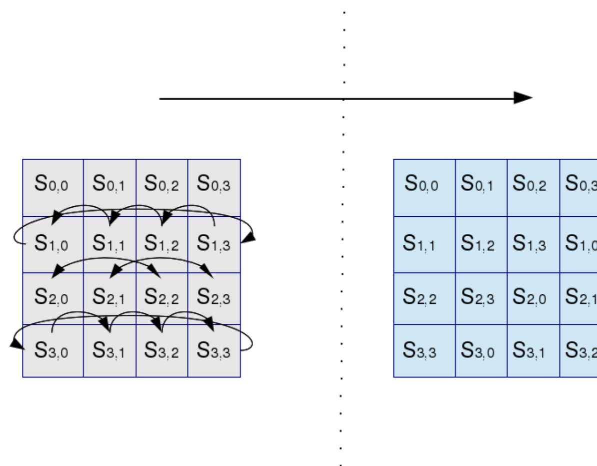


Figure 1.8: AES ShiftRows

MixColumns

In this step each bytes of a column are combined using an invertible linear transformation. Each input byte affects the four output bytes of the MixColumn function. It provides with ShiftRows the diffusion of the cipher. During this operation, each column is multiplied by the known matrix that for the 128-bit key is:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Let define:

$$\texttt{xtime}(\mathsf{a}) = \begin{cases} (a_6, \ldots, a_0, 0), \\ \quad \text{if } a_7 = 0 \\ \\ (a_6, \ldots, a_0, 0) \oplus (0, 0, 0, 1, 1, 0, 1, 1), \\ \quad \text{if } a_7 = 1 \end{cases}$$

With the AES parameters, the multiplication of x resume to the following

$$\begin{aligned} 01 \cdot \mathbf{a} &= \mathbf{a} \\ 02 \cdot \mathbf{a} &= \mathbf{xtime}(\mathbf{a}) \\ 03 \cdot \mathbf{a} &= 02 \cdot \mathbf{a} \oplus \mathbf{a}. \end{aligned}$$

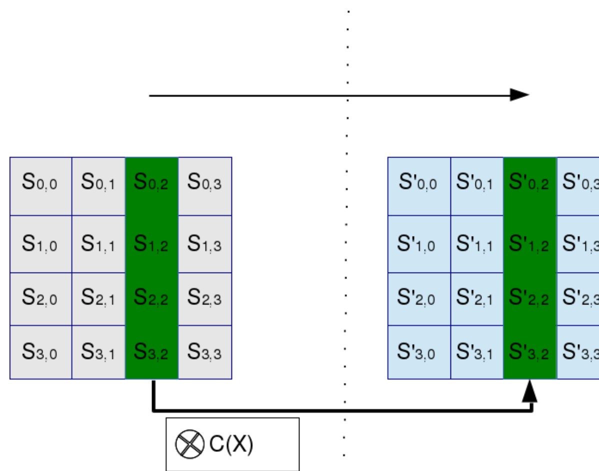


Figure 1.9: AES MixColumns

AddRoundKey

In the AddRoundKey step the subkey is changed by combining it with the state. For each round, a subkey is generated from the master key using the key schedule, each key is the same size as the state. The subkey is obtained by using the $\oplus$ operation on each byte of the state with the corresponding byte of the subkey.

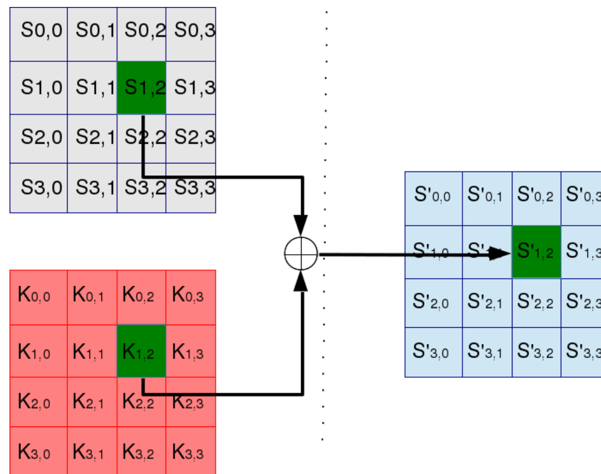


Figure 1.10: AES AddRoundKey

Software Implementation

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by combining the SubBytes and ShiftRows steps with the MixColumns step by transforming them into a sequence of table lookups. This requires four 256-entry 32-bit tables, and utilizes a total of four kilobytes (4096 bytes) of memory - one kilobyte for each table. A round can then be done with 16 table lookups and 12 32-bit exclusive-or operations, followed by four 32-bit exclusive-or operations in the AddRoundKey step. If the resulting four-kilobyte table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bit (i.e. 1 kilobyte) table by the use of circular rotates. Using a byte-oriented approach, it is possible to combine the SubBytes, ShiftRows, and MixColumns steps into a single round operation.

1.3.3 Attacks on Symmetric Ciphers

In modern cryptography the attacker is supposed to know everything about the algorithm, only the keys are assumed to be secret. This is called the Kerckhoff's princple. An attacker has to discover the key, a part of it, or be able to encrypt or decrypt blocks of data.

Exhaustive keysearch : given a few plaintext/ciphertexts search through all the possible keys until the correct one is found.

Ciphertex-only attack : the attacker has at his disposition only ciphertexts.

Known-plaintext attack : the attacker has at his disposition a certain number of plaintext and corresponding ciphertext.

Chosen-plaintext attack : the attacker can choose the plaintexts that are encrypted.

Double encryption

A common mistake is to think that a double encryption double the security of the cipher. A **meet in the middle attack** is possible is that case.

Alg. 1.4 Double Encryption Attack

Input: P_1, C_1,P_2,C_2

1: $A = L = \varnothing$
2: **for all** $K_f \in K$ **do**
3: $\quad SubCipherE = ENC(K_f, P_1)$
4: $\quad A[SubCipherE] \leftarrow K_f$
5: **end for**
6: **for all** $K_{b_1} \in K$ **do**
7: $\quad SubCipherD = DEC(K_b, C_1)$
8: $\quad K_t = A[SubCipherD]$
9: $\quad$ **if** $ENC(K_t, P_2) = DEC(K_b, C_2)$ **then**
10: $\quad\quad L \leftarrow L \cup (K_t, K_b)$
11: $\quad$ **end if**
12: **end for**
13: If $\|L\| \neq 1$ do an exhaustive search on L
14: **return** (L)

1.4 Asymmetric Ciphers

PK cryptography, refers to a cryptographic algorithm which requires two separate keys. One public that can be provided to everyone and one that is secret and must be kept secure by the owner of the key. Although different, both parts are linked mathematically. The public part is used to encrypt plaintext or verify signature. The private part is used to generate the signature or retrieve the plaintext from the encrypted ciphertext. It is also named asymmetric cryptography since different keys are used to perform opposite functions (like encryption and decryption).

The most common algorithm in PK are Rivest Shamir Adleman (RSA) and Elliptic Curve Cryptography (ECC). We will not get as deep as we did for the SK since our work only depends on some base function from an exponentiation. We will show in chapter 9 how to improve the exponentiation algorithm used on Embedded Devices in order to increase the security and speed of such operations if the required Hardware is available.

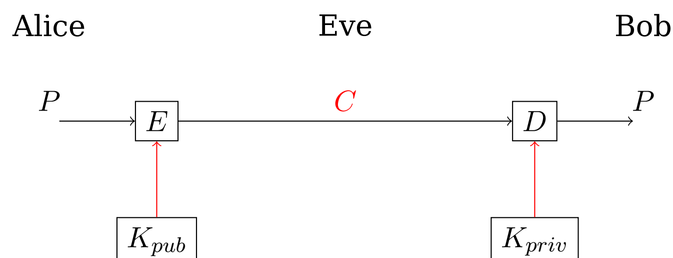


Figure 1.11: Asymmetric Cipher

Chapter 2

Side Channel Analysis

2.1 Introduction

SCA can be defined as any attack exploiting physical information leaked from a cryptographic device depending on the data processed.

During a cryptographic process, the device is likely to leak sensitive information, that can be timing information, power consumption, electromagnetic radiations, sound leaks, etc. SCA are passive attacks, in that the device under attack is not aware of its leaks being recorded.

In cryptography, a side channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms (compared to cryptanalysis). In the common literature information such as timing, power, electromagnetic or sound for sample were used in order to exploit and break a cryptosystem. Some SCA requires knowledge about algorithm and internal operations made by the system in order to break it while others like Differential Power Analysis (DPA) or Side Channel Analysis for Reverse Engineering (SCARE) attacks are effective as black-box approach and can even help an attacker understand what is computed by a secret device.

Analyzing properties (electromagnetic radiations) leaked from systems started many years years ago. The so-called TEMPEST project led by the US provided solutions to exploit and counteract the electromagnetic emanations (EM) of different electronic devices. It was used successfully in order to remotely view the screen of old CRT screens for example. Later Wim Van Eck studied also later those attacks in [101], then Paul C. Kocher et al. proposed in [60, 61] two variants of SCA: the Simple Power Analysis (SPA) presented in subsection 2.2.1 and the DPA presented in subsection 2.2.2. Those new attacks were a real breakthrough in the embedded security field and could break almost all the devices on field in the late 90's, like DES and RSA implementations.

After the initial publications of SPA and DPA, the passive embedded security field evolved quickly and some new generic attacks more powerful were found like the Correlation Power Analysis (CPA) [20], then others attacks like Mutual Information Analysis (MIA) or Templates Analysis appeared that can be efficient in some specific cases.

2.2 Attacks

2.2.1 Simple Power Analysis

SPA is defined by the Bundesamt für Sicherheit in der Informationstechnik (BSI) in the Common Criteria Protection Profile BSI-PP-0042 as "a direct analysis of patterns of instruction execution, obtained through monitoring variations in electrical power consumption of a cryptographic algorithm". One can try to break a chip secrets by finding in the patterns of a side-channel measurement enough information on the key to be able to recover it.

Timing attacks is a kind of SCA where the attacker focus on the time spent by the device to recover some secret information. It was introduced by Kocher on asymmetric algorithm.

SPA involves measuring variations in power consumption of a device as it performs an operation, in order to discover information about secret key material or data. This is achieved by mapping certain operation types to consumption patterns. For example, a series of $\oplus$ operations exhibits a different trace on an oscilloscope to a series of multiplication operations.

A good example in order to understand SPA is to take for sample a PIN verification shown in Figure 2.1. We can clearly see the 4 bytes verification on a good PIN that are done by the card when the PIN matches the one stored in its secret memory. Now let us take the same card and compute a verification on a PIN that has only its first 2 digits correct. The resulting SPA trace would be the one shown in Figure 2.2. On this last diagram we voluntarily zoomed on the interesting part containing the comparison of the PIN digits. While the green curve representing the computation done with the complete PIN has clearly 4 verifications, the signal of the one plotted in red changes after the second comparison when the 3rd digit is checked. This information could then be used by an attacker to retrieve the PIN stored on the card with a complexity reduced from 10^4 in the case where he would have to check all the different PIN possible of 4 characters long, to $10 * 4$ since he could test all the digits for the first one and look at the trace to see which one has a longer comparison time than the other one. This one would be the good guess. Then repeat for the 3 other digits.

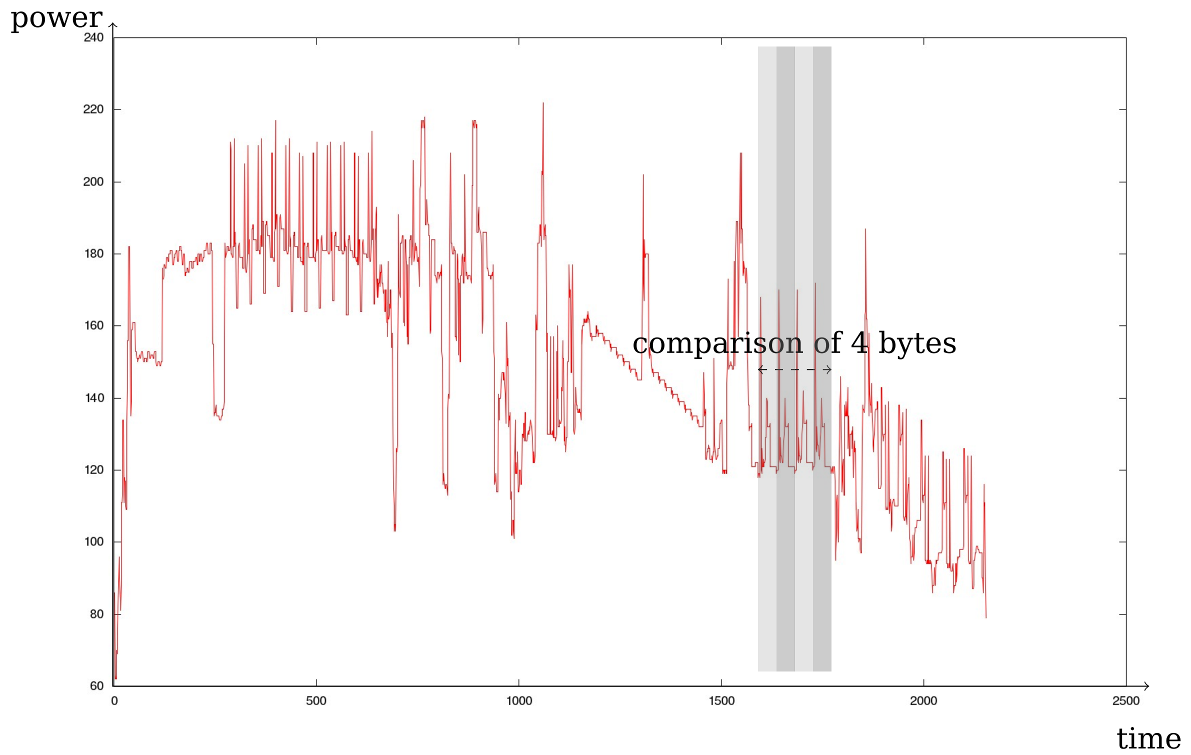


Figure 2.1: Good Pin

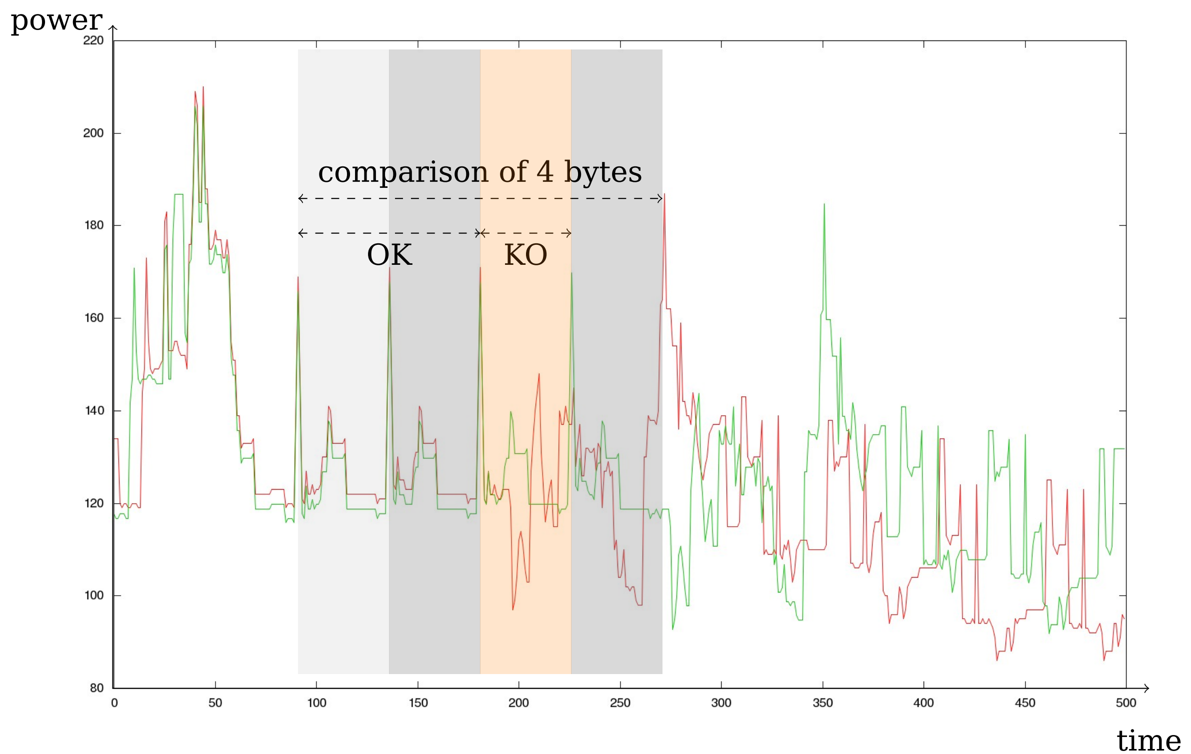


Figure 2.2: Superposition of Good and Wrong PIN verification

Another good example of this is RSA, which has to perform large multiplications, and therefore leaks information about the internal state of a large integer

multiplication via the pattern of operations it performs. If the algorithm is not protected, like the one presented in 9.1, one can deduce by the size of the pattern for sample whether the secret exponent bit is 0 or 1.

One can also use the fact that unprotected signals are represented physically by high or low voltages, therefore a 1 uses more power than a 0 for the period that bit is used by the processor.

While smart cards are usually very specific and perform only a single thread at a time, more complex systems are multithreaded and can execute multiple interlaced operations at the same time. The SPA on a modern desktop computer or a smart-phone would be more difficult to implement from what is done on a smart card, but other kinds of powerful attacks like cache attacks [13] are present on those complex systems.

Acquiring the signal is necessary but most of the time it then has to be filtered in order to remove any excessive noise or to display more information. We have the same signal acquired from a contactless card, the first one (Figure 2.3) is not filtered, while the second one (Figure 2.4) is. The reader can easily understand that it is easier to retrieve what the card is doing when the noise is removed in the second plot, and usually the attacks will also work better with noise reduction.

For more example of SPA, the reader should have a look at the section subsection 10.2.4.

Figure 2.3: Contactless SPA

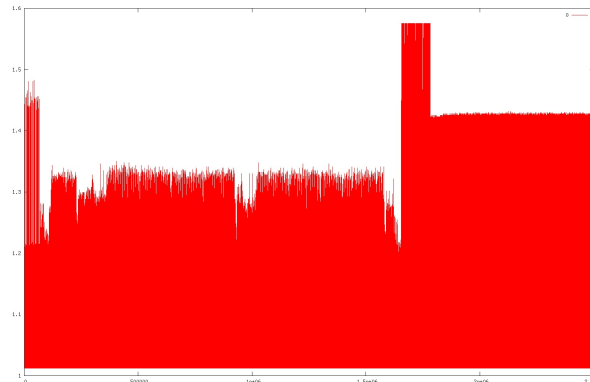
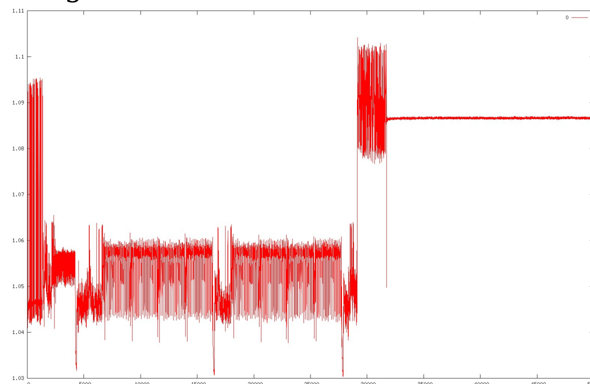


Figure 2.4: Contactless SPA filtered



2.2.2 Differential Power Analysis

SPA by itself is already a really efficient attack, but another breakthrough published on the same paper [61] made it looks not so powerful. The noise introduced by the component and the measurement bench and the fact that the difference in consumption between two different operations is very subtle makes it very hard on a single trace to get enough information about the internal of an algorithm when no timing information appears. By using multiple traces though, assuming that the same operation in all those traces appears at the same point in time, one can use statistical means in order to remove the noise in the traces and to find statistical bias in the measurement yielding to the secret key used.

A selection function $D(i, G)$ depending on a part of a secret and an input is applied on measurements in order to split them in two parts, then statistical means are used to find if the variables $D(i, G)$ and the power traces T_i are independent or not. If they are not independent then the correct part k of the key has been retrieved.

In the original DPA the selection function denoted $D(i, G_{b,K_s})$ is defined as computing the value of bit number b ($0 <= b < 32$) of the DES intermediate L at the beginning of the 16th round for ciphertext C_i, where the 6 key bits entering the S-Box corresponding to bit b are represented by K_s ($0 <= K_s < 2^6$). If K_s is incorrect, then evaluating $D(i, G_{b,K_s})$ will yield the correct value for bit b with a probability $p \approx \frac{1}{2}$ for each ciphertext.

The attacker would first observe n encryptions and capture power traces $T_{1..n}[1..k]$ containing k samples each and the ciphertext $C_{1..n}$ without knowledge of the plaintext.

Once the preparation phase is done, the attacker computes

$$\Delta_D[j] = \frac{\sum_{i=1}^{n} D(i, G_{b,K_s})T_i[j]}{\sum_{i=1}^{n} D(i, G_{b,K_s})} - \frac{\sum_{i=1}^{n} 1 - D(i, G_{b,K_s})T_i[j]}{\sum_{i=1}^{n} 1 - D(i, G_{b,K_s})} \tag{2.1}$$

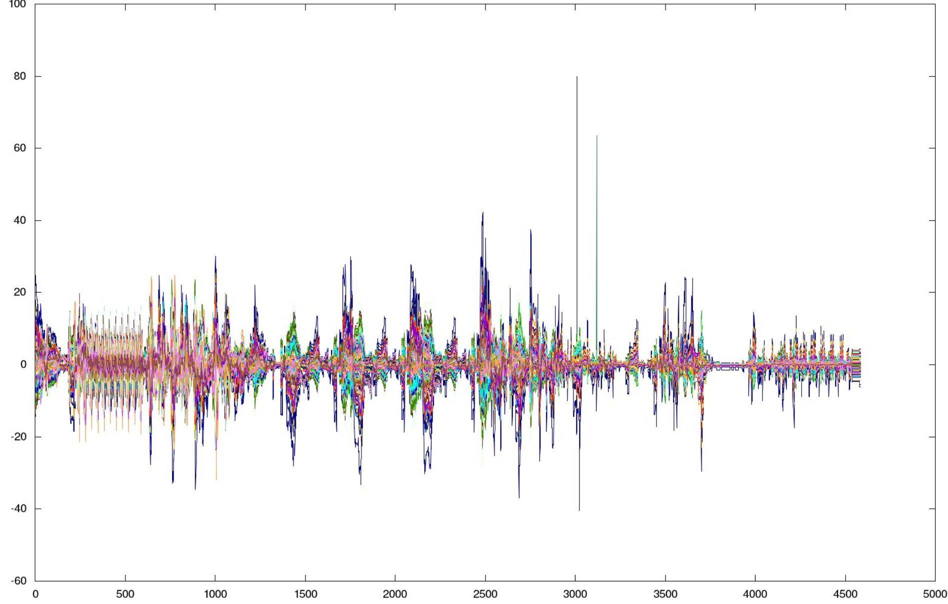
If K_s is incorrect, the bit computed using D will differ from the actual target bit for about half of the ciphertexts C_i and as a consequence the selection function $D(i, G_{b,K_s})$ is uncorrelated to the power traces saved from the computations made by the device. If a random function is used to divide a set into two subsets, the difference in the averages of the subsets should approach zero as the subset sizes approach infinity.

$$\lim_{n \to \infty} \Delta_D[j] \approx 0, \forall j = 1, .., k \tag{2.2}$$

Thus if K_s is incorrect because trace components uncorrelated to D will diminish with $\frac{1}{\sqrt{n}}$ making the differential trace become flat. If K_s is correct the computed value will equal the actual value of the target bit b with a probability of 1. The selection function is thus correlated to the power traces and as a result $\Delta_D[j]$ approaches the effect of the target bit on the power consumption as $n \to \infty$ while other data values, noise, ... that are not correlated to D approach zero. Since power consumption is correlated to data bit values, only a few spot on the traces will be correlated to the handling of the specified bit and as a result the plot of Δ_D will be almost flat everywhere with spikes in regions where D is correlated to our selection function.

One can then find the good values for K_s visually.

Figure 2.5: DPA sample



2.2.3 Distinguishers

We call distinguisher the function applied to the selection function in order to find if it is or not correlated to the side-channel measurements [97]. Let $D_{i,G}$ be the selection function that for a computation i and a guess G associates a model of consumption that can be either 0 or 1 like in the original DPA an integer between 0 and 8 for a classical CPA ....

Alg. 2.1 Differential Power Analysis

Input: $D(i, G)$, T_i, Δ
  for all guesses G **do**
    build $\Delta_{D_G}[j], \forall j = 1, .., n$
  end for
  return $\mathrm{argmax}_G\,(|\Delta_{D_G}[j]|, \forall j = 1, .., n)$

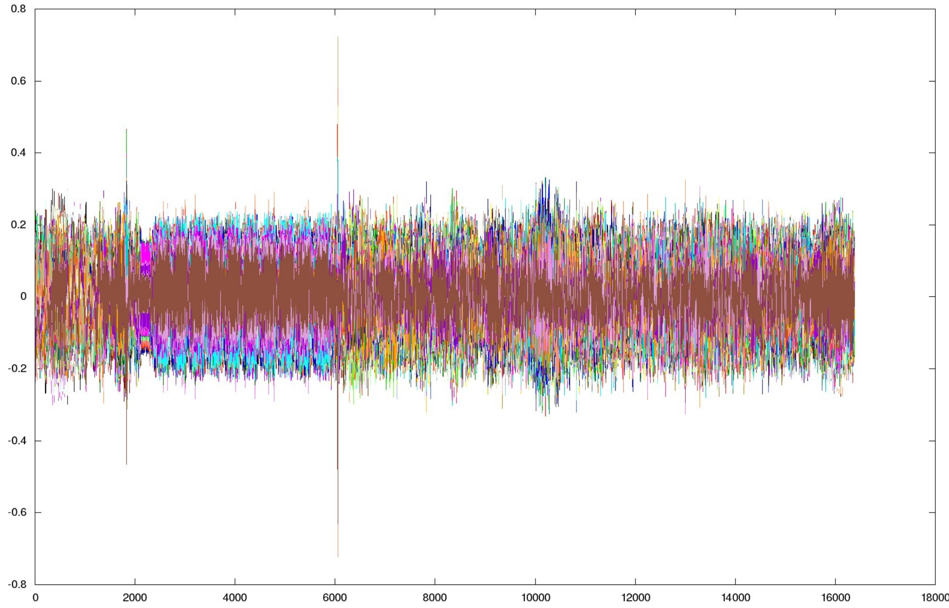
Original DPA , also called Difference of Means (DoM), involves a partition in two sets as a single bit consumption activity is considered. This distinguisher is simple and can be seen as:

$$DoM : \Delta_{D_G}[j] = \frac{\sum (D(i,G)T_i[j])}{\sum D(i,G)} - \frac{\sum (1 - D(i,G)T_i[j])}{\sum (1 - D(i,G))} \tag{2.3}$$

CPA Using the normalized coefficient of correlation instead of the covariance. It was introduced with the Pearson correlation coefficient with the form:

$$CPA : \Delta_{D_G}[j] := \frac{N \sum T_i[j] D(i,G) - \sum T_i[j] \sum D(i,G)}{\sqrt{N \sum T_i[j]^2 - (\sum T_i[j])^2} \sqrt{N \sum D(i,G)^2 - (\sum D(i,G))^2}} \tag{2.4}$$

Figure 2.6: CPA



Mutual Information Analysis (MIA) Issued from the probability and information theory, the MIA of two random variables measure the mutual dependence between them. The measurements and the model of consumption can be considered as two discrete variable X and Y and their mutual information then express as:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right) \tag{2.5}$$

With $p(x,y)$ the joint probability distribution function of X and Y, and $p(x)$ and $p(y)$ the marginal probability distribution functions.

Intuitively, mutual information measures the information that X and Y share. It can measures how much knowing one of these variables reduces the uncertainty about the other one. If X and Y are independent, knowing X or knowing Y does not give any information about the other and then their mutual information is zero. On the other hand if X and Y are identical [1] then knowing X yields the value of Y and as a result the mutual information is the same as the uncertainty contained in X (or Y) alone, namely the entropy of X (or Y).

Mutual information (MI) measures the dependence in the joint distribution of X and Y relative to the joint distribution of X and Y under the assumption of independence. Mutual information measures the dependence in the sense: $I(X;Y) = 0$ if and only if X and Y are independent random variables. This is easy to see in one direction: if X and Y are independent, then $p(x,y) = p(x)p(y)$, and therefore:

$$\log \left(\frac{p(x,y)}{p(x)\,p(y)} \right) = \log 1 = 0 \tag{2.6}$$

[1] Or one of them is a deterministic function of the other.

Moreover, mutual information is nonnative (i.e. $I(X;Y) \geq 0$) and symmetric (i.e. $I(X;Y) = I(Y;X)$).

Gierlichs et al. in [46] first applied the MIA to embedded security field. Let T and D be the variable associated with the curves and the selection function, t and d the realizations of T and D respectively, $H(T)$ is an estimation of the entropy of T, $H(T|D)$ is the conditional entropy of T knowing L,$p(t,d)$ is the joint probability density function of T and D, $p(t)$ is the marginal probability density function of T. By applying the previous equation formula we get:

$$MI(T;D) = \sum_{t \in T} \sum_{d \in D} p(t,d) \log \left(\frac{p(t,d)}{p(t)\,p(d)} \right) \tag{2.7}$$

$$MI(T;D) = H(T) - H(T|D) \tag{2.8}$$

The higher the value of $MI(T;D)$ is, the higher the dependency between T and D is. In practice, it is hard to get an accurate estimation for the probability density functions. Many methods have been proposed to estimate entropy histograms, kernel density functions, Gaussian parametric estimators etc. [83]. With the distribution of (T,D) assumed Gaussian, the Gaussian parametric estimation can be used as a first approximation of the distribution and then

$$H(T) = -\Sigma_i p(t_i) \log(p(t_i)) = \log(\sigma_t(2\pi e)) \tag{2.9}$$

Moreover, under the Gaussian assumption, it can be verified that mutual information is intimately connected to the Pearson coefficient ρ and can be expressed as [88, 64] follows:

$$MI(T;D) = -\frac{1}{2} \log(1 - \rho^2_{T,D}) \tag{2.10}$$

2.2.4 High Order Attacks

Introduced in [61] and revisited in [71] a High Order Differential Power Analysis (HODPA) attack is defined as a DPA attack that combines one or more samples within a single power trace. A nth-*order* DPA attack makes use of n different samples in the power consumption signal that correspond to n different intermediate values calculated during the execution of an algorithm. It usually requires more computing power or more information on the targeted device than classical DPA.

Chapter 3

Fault Analysis

3.1 Introduction

During the 1970's some researchers noticed that sensitive areas of electronic chips would be changed by radioactive particles. Starting from this observation and led by the aerospace industry research on electronic devices resistance to various environments progressed. It produced various patents and mechanisms both for fault creation and propagation and for protection against them. It was observed also that many cryptographic algorithms implemented by electronic could be cracked by using such faults, those kind of attacks were named accordingly "fault attacks". Being able to prevent such faults while at the same time keeping good performances is a non trivial problem. Such fault analysis were applied to almost every cryptosystems in cryptography, on block ciphers like DES or AES and at the same time on asymmetric cryptography like RSA CRT.

The first fault (active) attacks, named Differential Fault Analysis (DFA), were published by Boneh, DeMillo and Lipton [19] and Biham and Shamir [15]. The first technique, also known as the *Bellcore* attack, threatens implementations of the RSA cryptosystem [86] while the second one targets the DES algorithm [39].

3.2 Cosmic Rays

The cosmic rays probably were the first cause of fault observed involuntary on aviation or space travel research. Such faults cause issues since the first days of space flights. They are subatomic particles with high-energy coming from the outer space. They can be compared to high-energy protons and neutrons produced by accelerators. Relying on such rays for attacks means that an attacker waits until by chance such an event happend. The authors in [50] expect that an adversary has to wait for several months until a bit flips caused by cosmic rays in DRAM cells.

3.3 Heat / Infrared Radiation

Electronic equipment works reliably only in certain range of temperature. If the outside temperature is under or above a certain threshold, faults start to occur.

Most PCs have a fan to prevent overheating. In [50] the authors experimented infrared radiation coming from a spotlight lamp together with a variable power supply. They successfully induced faults on a standard PC (single bit flip) for temperatures between 80 and 100 Celsius.

3.4 Power Spike (glitch)

A glitch is a fault with a short life in a system. It is usually used in order to describe a fault that is difficult to troubleshoot. The term is common in computing, electronic components as well as video games. It can be however applied to all types of systems.

An electronic glitch can be defined as an unexpected transition that occurs in the transistors. In other words, it is a short electrical pulse that changes the behavior of the system. For example flip-flops are triggered by a pulse that must not be shorter than a specified minimum duration; otherwise, the component may malfunction. A pulse shorter than the specified minimum is called a glitch. Two related concepts are the runt pulses, a pulse whose amplitude is smaller than the minimum level specified for correct operation, and a spike, a short pulse similar to a glitch but often caused by ringing or crosstalk. A glitch can occur in the presence of race condition in a poorly designed digital logic circuit.

Such glitches introduced in a microprocessor may allow an attacker to introduce faults in cryptosystems and consequently "break" them. Spikes allow to induce both memory faults as well as faults in the execution of a program (code change attacks), cf. [63].

3.5 Clock glitches

An attacker may replace the smart-card reader by laboratory equipment, he may then provide his own clock signal with incorporates short massive deviations which are beyond the required tolerance bounds. Such signals are called clock glitches and can be used to both induce memory faults or to change the code executed. Hence, the possible effects are the same as in power spike.

3.6 Laser

Power or clock glitches while already powerful are the big hammer tool for fault attacks. They are very global and can trigger lot of different behaviors on the chip at the same time. By using laser to send photons to specific part of an electronic component, attackers may target part of an hardware, like a specific IP or even a bit in memory on in a register depending on the product.

3.7 Electromagnetic Pulses

The electromagnetic (EM) medium may be used to conduct active attacks. Two kinds of near-field EM perturbations are usually considered: transient pulses [84, 90] and harmonic emissions [3, 12].

For example the effect of both electric and magnetic fields along the x-, y-, and z-axes were studied. Their test circuits were found sensitive to both magnetic and, to a greater extent, electric fields. Those results create new threats against the security of random generator using such IP as source of entropy.

We will not go more into detail on fault attacks since we did not create a new kind of fault attack, but combined them with side-channel to create a new kind of attack. We will present some consequences of fault attacks.

3.8 Focused Ion Beams (FIB)

They are frequently used in the reverse engineering of smart-cards. They are composed of a particle gun shooting ions (for example Gallium ions from a liquid metal cathode), and a microscope which focus the beam of ions. FIB are used to drill holes in different layers of a smart-card to access single elements of bus lines with measuring equipment. FIB can also be tuned finely to ionize silicon locally, that action may be interpreted as a signal by the circuit.

3.9 Sample Attacks

Examples of those attacks can be found in subsection 10.3.2

3.9.1 PIN Verification

In order to make a PIN verification, a target tests at some points whether an input match a secret stored by the card. If not protected correctly, one can easily bypass a PIN verification with a simple laser pulse by tricking the target into thinking that the input matched the expected value, while they were actually different.

3.9.2 RSA CRT

RSA CRT is a way of implementing RSA based on splitting the exponentiation in two halves. Using the Chinese Remainder Theorem, one computes a result modulo $n = pq$ by splitting it into two computations modulo p and q respectively.

So, in RSA we need to compute $c = m^d \pmod n$, but what we actually do is computing intermediate results modulo p and q and then combining them. So we compute $c_q = m^{d_q} \pmod q$ and $c_p = m^{d_p} \pmod p$, and we combine them to obtain the whole result like this:

$$c = (((c_q - c_p) * I_p) \pmod q) \cdot p + c_p where I_p = p^{-1} \cdot \pmod q$$

Now, imagine you are able to modify one of the two exponentiations, for instance you modify c_q and get c'_q instead. Now, if we subtract the two results, c and c', we get the following:

$$c - c' = (((c_q - c_p) * Ip) \pmod q) \cdot p + c_p - ((((c'_q - c_q) * Ip) \pmod q) \cdot p + c_p)$$

Obviously, many of these terms are the same on both sides, and thus we can simplify this result a lot:

$$c - c' = (((c_q - c'_q) * Ip) \pmod q) \cdot p$$

This number is a multiple of one of the primes. By computing the greatest common divisor between $c - c'$ and n, we obtain the common factor p. From there we can compute the other prime q, by just dividing n by p.

3.9.3 Dump of ROM

One can also too use faults attacks in order to dump ROM parts. For sample a routine used for emission could be coded as:

```
void transmit_array(unsigned char* array,size_t len)
{
        for(int i=0;i<len;i++){
                transmit_byte(array[i]);
        }
}
```

In that case if the variable `len` is faulted during the loop to `0xFFFF` (worst case on a 16 bits platform) we would dump 65kb of memory, possibly the whole RAM, ROM, EEPROM depending on the hardware architecture.

3.9.4 Hack of the Playstation 3

The Playstation 3 (PS3) is a really powerful console made by Sony and depends on a hypervisor to enforce security. This console allowed users to run an ordinary Linux if they wanted under the management of this hypervisor. The hypervisor prevents the Linux kernel from accessing different devices like the GPU by acting as a firewall layer between the OS and the hardware. Hacking the hypervisor would give the user full control over the console and the possibility to run whatever game, original or not that would be inserted. The hack of the console, first published by George Hotz, compromises the hypervisor after booting standard Linux through fault injection. He connected an Field-Programmable Gate Array (FPGA) to a single line on his PS3 memory bus. He then programmed the FPGA to send a pulse of 40 ns when instructed to do so via a switch. It represents about 100 memory clock cycle for the PS3. Even though the fault attack itself is very imprecise, he used software means in order to enhance the likelihood of success.

The internals of the exploit is to prevent memory deallocation by the hypervisor through glitch. The memory is deallocated by the processor first but since there is a cache the writing is done to the cache, once the cache tries to write back to the memory bus the attack takes place and prevents it. The hypervisor now thinks that the memory is not mapped anymore while the Linux kernel has still access to it. The next step consists in making requests to the hypervisor to create new

virtual code segment until one is found to be positioned in the memory controlled by the Linux Kernel. Then one can modify the data of the virtual code segment and allows it to have full access of the memory. The Linux kernel has then access to the whole hardware with the same rights as the hypervisor.

3.10 Countermeasures

A variety of hardware countermeasures have been developed as fault attacks became stronger and stronger. These countermeasures are usually specific to a certain kind of physical attack. Sensors and filters aim to detect attacks like light detectors, anomalous voltage detectors or anomalous frequency detectors. Redundancy is another countermeasure commonly used in dual-rail logic for sample, where each bit of memory is doubled at the cost of twice the hardware size, where each computation is done twice in parallel and checked to detect any discrepancy. If both results are identical then one can assume that no fault occurred. Randomized clocks are another countermeasure commonly used to provides some unstable frequency of the internal clock and makes the synchronization works harder. Memory encryption, passive or active shields, dummy random cycles are others hardware countermeasures used to protect smart-cards.

Hardware only countermeasures are very expensive though and usually specific to a special kind of attack, since new kind of faults frequently appear, detecting only currently known ones is required but not sufficient to provide resistance during the lifetime of a secure product. Therefore software countermeasure are used since they are easier to implement, cost less in hardware size of price and are usually easier to deploy since a simple patch can add new countermeasures. The current state of the art for software countermeasures are masking, checksums, randomization, redundancy, golden values (baits) and counters.

Part II

Contributions

Chapter 4

Our work

During the three years of my PhD we contributed to different fields of the embedded security. I will now present briefly the different results we got and the improvement over the state of the art.

Horizontal Correlation Analysis on Exponentiation In chapter 5 a new side-channel analysis on an RSA exponentiation. It computes correlation estimates, which are generally used by side-channel analysis to infer information on a secret key from numerous exponentiation traces. Our method allows a secret exponent to be recovered from a single trace under realistic assumptions. This was presented in ICICS 2010 [29].

Improved Collision-Correlation Power Analysis on AES The recent results presented by Moradi et al. on AES at CHES 2010 and Witteman et al. on square-and-multiply always RSA exponentiation at CT-RSA 2011 have shown that collision-correlation power analysis is able to recover the secret keys on embedded implementations. However, we noticed that the attack published last year by Moradi et al. is not efficient on correctly first-order protected implementations. We propose in chapter 6 improvements on collision-correlation attacks which require less power traces than classical second-order power analysis techniques. We present here two new methods and show in practice their real efficiency on two first-order protected AES implementations. We also mention that other symmetric embedded algorithms can be targeted by our new techniques.This was presented in CHES 2011 [31].

ROSETTA In most efficient exponentiation implementations, recovering the secret exponent is equivalent to disclosing the sequence of squaring and multiplication operations. Some known attacks on the RSA exponentiation apply this strategy, but cannot be used against classical blinding countermeasures. In chapter 7, we propose new attacks distinguishing squaring from multiplications using a single side-channel trace. It makes our attacks more robust against blinding countermeasures than previous methods even if both exponent and message are randomized, whatever the quality and length of random masks. We demonstrate the efficiency of our new techniques using simulations in different noise configurations. This was presented in INDOCRYPT 2012 [27].

Passive and Active Combined Analysis Tamper resistance of hardware products is currently a very popular subject for researchers in the security domain. Since the first Kocher side-channel (passive) attack, the Bellcore researchers

and Biham and Shamir fault (active) attacks, many other side-channel and fault attacks have been published. The design of efficient countermeasures still remains a difficult task for IP designers and manufacturers as they must also consider the attacks which combine active and passive threats. It has been shown previously that combined attacks can defeat RSA implementations if side-channel countermeasures and fault protections are developed separately instead of being designed together. In chapter 8 we demonstrates that combined attacks are also effective on symmetric cryptosystems and show how they may jeopardize an allegedly state of the art secure AES implementation. This was presented in FDTC 2010 [28].

Square Always In chapter 9 we present new exponentiation algorithms using squarings only while implementing the atomicity principle. The proposed algorithms are shown to be more resistant against attacks than previous atomic methods, we also achieve higher speed than known regular algorithms. We then propose parallelization of squarings to make our algorithms the most efficient. This was presented in INDOCRYPT 2011 [32].

Simulation In chapter 10 we start by having a look at setup needed to produce fault attacks, laser and other glitch bench are expensive and need quite a time of setup each time a new product has to be tested. We then review previous work done on hardware and software simulation and present the work we started to develop in 2010 by modelizing CPU at the instruction level for the simulation of power consumptions traces used in some of our papers like [30]. We then expose the rationale and inner working of the fault simulator we used to understand and fix software code that was embedded in product destined to be used worldwide. This part was not presented at any conference, but we think that the Linux compatibility and other implementation details are new in the security field.

Chapter 5

Horizontal Correlation Analysis

5.1 Introduction

Securing embedded products from *Side-Channel Analysis* (SCA) has become a difficult challenge for developers who are confronted with more and more analysis techniques as the physical attacks field is studied. Since the original *Simple* Side-Channel Analysis (Simple Side-Channel Analysis (SSCA)) – which include *Timing Attacks*, SPA, and Simple ElectroMagnetic Analysis (SEMA) – and *Differential* Side-Channel Analysis (Differential Side-Channel Analysis (DSCA)) – including DPA and Differential ElectroMagnetic Analysis (DEMA) – have been introduced by Kocher et al. [60, 61] many improvements and new SCA techniques have been published. Messerges et al. were the first to apply these techniques to public-key implementations [70]. Later on, original DSCA has been improved by more efficient techniques such as the one based on the *likelihood test* proposed by Bevan et al. [14], the *Correlation* Power Analysis (CPA) introduced by Brier et al. [20], and more recent techniques like the *Mutual Information Analysis* (MIA) [45, 82, 98]. A common principle of all these techniques is that they require many power consumption or electromagnetic radiation curves to recover the manipulated secre . Hardware protections and software blinding [34, 60] countermeasures are generally used and when correctly implemented they counteract these attacks.

Among all those studies the so-called *Big Mac attack* is a refined approach introduced by Walter [104, 105] from which our contribution is inspired. This technique aims at distinguishing squarings from multiplications and thus recovering the secret exponent of an RSA exponentiation with a single execution curve. This can be achieved by averaging and comparing the cycles of a device multiplier during long integer multiplications.

We present in this chapter another analysis which uses a single curve. We named this technique *horizontal* correlation analysis, which consists of computing classical statistical treatments such as the correlation factor on several segments extracted from a single execution curve of a known message RSA encryption. Since this analysis method requires only one execution of the exponentiation as the Big Mac attack, it is then not prevented by the usual exponent blinding countermeasure.

The chapter is organized as follows. The section 5.2 gives an overview of asymmetric algorithms and the way to compute long integer multiplication in embedded

implementations. The section 5.3 reminds to the reader previous studies on power analysis techniques discussed in this article. The horizontal correlation analysis is presented in section 5.4 with some practical results and a comparison between our technique and the Big Mac attack. Known and new countermeasures are discussed in section 5.5 and in section 5.6 we deal with horizontal side channel analysis in the most common cryptosystems.

5.2 Public-Key Embedded Implementations

RSA is well-known to be currently the most used public-key cryptosystem in smart devices. Other public-key schemes such as Digital Signature Algorithm (DSA) [41], Diffie-Hellman key exchange [36] protocols, and their equivalent in Elliptic Curve Cryptography (ECC) – namely Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Diffie–Hellman (ECDH) [41] – are also often involved in security products. Interestingly, all of them are based on the modular exponentiation or the scalar multiplication and in both cases the underlying operation is modular long integer multiplication. Heavy efficiency constraints thus lie on this operation, especially in the context of embedded devices. Many methods such as the Montgomery multiplication [77] and interleaved multiplication-reduction with Knuth, Barrett, Sedlack or Quisquater methods [35] can be applied to perform efficient modular multiplications. Most of them have in common that the long integer multiplication is internally done with a loop of one (or more) smaller multiplier(s) operating on t-bit words. An example is given in Alg. 5.1 which performs the schoolbook long integer multiplication using a t-bit internal multiplier giving a $2t$-bit result. The decomposition of an integer x in t-bit words is given by $x = (x_{l-1}x_{l-2}\dots x_0)_b$ with $b = 2^t$ and $l = \lceil log_b(x) \rceil$. Other long integer multiplication algorithms may also be used such as Comba [33] and Karatsuba [58] methods.

Alg. 5.1 Long Integer Multiplication

1: $x = (x_{l-1}x_{l-2}\dots x_0)_b, y = (y_{l-1}y_{l-2}\dots y_0)_b$ $\mathrm{LIM}(x, y) = x \times y$
2: **for** $i = 0$ to $2l - 1$ **do**
3: $\quad w_i = 0$
4: **end for**
5: **for** $i = 0$ to $l - 1$ **do**
6: $\quad c \leftarrow 0$
7: $\quad$ **for** $j = 0$ to $l - 1$ **do**
8: $\quad\quad (uv)_b \leftarrow (w_{i+j} + x_i \times y_j) + c$
9: $\quad\quad w_{i+j} \leftarrow v$ and $c \leftarrow u$
10: $\quad\quad w_{i+l} \leftarrow c$
11: $\quad$ **end for**
12: **end for**
13: **return** (w)

We consider that a modular multiplication $x \times y \bmod n$ is performed using a long integer multiplication followed by a Barrett reduction denoted by $\mathrm{BarrettRed}(\mathrm{LIM}(x, y), n)$.

Alg. 5.2 presents the classical *square and multiply* modular exponentiation algorithm using Barrett reduction. More details on Barrett reduction can be found in [10, 69] and other methods can be used to perform the exponentiation such as Montgomery ladder [75] and *sliding window* techniques [22].

Alg. 5.2 Square and Multiply Exponentiation

1: integers m and n such that $m < n$, v-bit exponent $d = (d_{v-1}d_{v-2}\dots d_0)_2$
   $\mathrm{Exp}(m, d, n) = m^d \bmod n$
2: $a \leftarrow 1$
3: Process Barrett reduction precomputations
4: **for** $i = v - 1$ to 0 **do**
5: $\quad a \leftarrow \mathrm{BarrettRed}(\mathrm{LIM}(a, a), n)$
6: $\quad$ **if** $d_i = 1$ **then**
7: $\quad\quad a \leftarrow \mathrm{BarrettRed}(\mathrm{LIM}(a, m), n)$
8: $\quad$ **end if**
9: **end for**
10: **return** (a)

We assume in the following of this chapter that Alg. 5.2 is implemented in an SPA resistant way, for instance using the *atomicity* principle [23].

While we have chosen to consider modular multiplication using Barrett reduction, and square and multiply exponentiation, the results we present in this chapter also apply to the other modular multiplication methods, long integer multiplication techniques and exponentiation algorithms mentioned above.

5.3 Side-Channel Analysis

We have chosen to introduce the terms of *vertical* and *horizontal* side-channel analysis to classify the different known attacks. The present section deals with known vertical and horizontal power analysis techniques. Our contribution, the horizontal correlation analysis on exponentiation is detailed in section 5.4.

5.3.1 Background

Side-channel attacks rely on the following physical property: a microprocessor is physically made of thousands of logical gates switching differently depending on the executed operations and on the manipulated data. Therefore the power consumption and the electromagnetic radiation, which depend on those gates switches, reflect and may leak information on the executed instructions and the manipulated data. Consequently, by monitoring the power consumption or radiation of a device performing cryptographic operations, an observer may recover information on the implementation of the program executed and on the secret data involved.

Simple Side-Channel Analysis

In the case of an exponentiation, original SSCA consists in observing that, if the squaring operation has a different pattern from the one of the multiplication, the secret exponent can be read from the curve. Classical countermeasures consist of

using so-called *regular* algorithms like the *square and multiply always* or Montgomery ladder algorithms [75, 55], *atomicity* principle which leads to regular power curves as presented in Appendix section 5.7 in Figure 5.9.

Differential Side-Channel Analysis

Deeper analysis such as DSCA [70] can be used to recover the private key of an SSCA protected implementation. These analysis make use of the relationship between the manipulated data and the power consumption/radiation. Since this leakage is very small, hundreds to thousands of curves and statistical treatment are generally required to learn a single bit of the exponent. Usual countermeasures consist of randomizing the modulus, the message, and/or the exponent.

Correlation Power Analysis

This technique is essentially an improvement of the Differential Power Analysis. Initially published by Brier et al. [20] to recover secrets on symmetric implementations, CPA is also successful in attacking asymmetric algorithms [8] with much fewer curves than classical DPA. The power consumption of the device is supposed to vary linearly with $\mathrm{HW}(D \oplus R)$, the Hamming distance between the data manipulated D and a *reference state* R. The consumption model W is then defined as $W = \mu \cdot \mathrm{HW}(D \oplus R) + \nu$, where ν captures both the experimental noise and the non modelized part of the power consumption. The linear correlation factor $\rho_{C,H} = \frac{\mathrm{cov}(C,H)}{\sigma_C \sigma_H}$ is then used to correlate each power curve C with $\mathrm{HW}(D \oplus R)$. The maximum correlation factor being obtained for the right guess of secret key bits, an attacker can try all possible secret bits values and select the one corresponding to the highest correlation value.

In [8], Amiel et al. apply the CPA to recover the secret exponent of public-key implementations. Their practical results show that the number of curves needed for an attack is much lower compared to DPA: less than one hundred of curves is sufficient. It is worth noticing that the correlation is the highest when computed on t bits, t being the bit length of the device multiplier.

The authors shows the details [8, Fig. 8] of the correlation factor obtained for every multiplicand t-bit word A_i during the squaring operation $A \times A$ using a hardware multiplier. Interestingly a correlation peak occurs for $\mathrm{HW}(A_i)$ each time a word A_i is involved in a multiplication $A_i \times A_j$.

We present in the next section our horizontal correlation analysis which takes advantage of this observation.

Collision Power Analysis

The *Doubling attack* from Fouque and Valette [42] is the first collision technique published on public-key implementations. It is originally presented on elliptic curve scalar multiplication but can be applied on exponentiation algorithms. It recovers the whole secret scalar (exponent) with only a couple of curves. Other collision attacks have been presented in [5, 53, 108]. They all require at least two

power execution curves, therefore the classical exponent randomization (blinding) countermeasure prevents those techniques.

Notations Let C^k denote the portion of an exponentiation curve C corresponding to the k-th long integer multiplication, and $C^k_{i,j}$ denote the curve segment corresponding to the internal multiplication $x_i \times y_j$ in C^k.

Big Mac Attack

Walter's attack needs, as our technique, a single exponentiation power curve to recover the secret exponent. For each long integer multiplication, the Big Mac attack detects if the operation processed is either $a \times a$ or $a \times m$. The operations $x_i \times y_j$ – and thus curves $C^k_{i,j}$ – can be easily identified on the power curve from their specific pattern which is repeated l^2 times in the long integer multiplication loop. A template power trace T^1_m is computed (either from the precomputations or from the first squaring operation) to characterize the message value m manipulation during the long integer multiplication. The Euclidean distance between T^1_m and each long integer multplication template power trace is then computed. If it exceeds a threshold the multiplication trace is supposed to be a squaring, and a multiplication by m otherwise. An example of such calculation is given in the following: for each t-bit word m_i of the message m, compute $T^1_{m_i} = \frac{1}{l}\sum_{j=0}^{l-1} C^1_{i,j}$ by averaging the l subcurves $C^1_{i,0} \dots C^1_{i,l-1}$. Then the template curve T^1_m is the concatenation of the average curves $T^1_{m_0} ... T^1_{m_{l-1}}$. In the exponentiation loop, at each k-th long integer multiplication, the curve T^k is computed in the same manner. The Euclidean distance between T^k and T^1_m is computed. If it exceeds a threshold the multiplication is supposed to be a squaring, and a multiplication by m otherwise. The attack is innovative and has been presented by Walter with theoretical and simulation results. The efficiency of the attack increases with the key length and decreases with the multiplier size.

Cross-Correlation

Cross-correlation technique has been used in [70] to try to recover the secret exponent with a single curve. However the cross correlation curve obtained by the authors did not allow distinguishing a multiplication from a squaring. More generally no successful practical result for cross correlation using a single exponentiation power curve has been yet published.

5.3.2 Vertical and Horizontal Attacks Classification

We refer to the techniques analyzing a same time sample in many execution curves – see Figure 5.1 – as *vertical* side-channel analysis. The classical DPA and CPA techniques thus fall into this category. We also include in the vertical analysis class the collision attacks mentioned above. Indeed even if many points on a same curve are used by those techniques, they require at least two power execution curves and manipulate them together. All those attacks are avoided with the exponent blinding countermeasure presented by Kocher [60, Section 10].

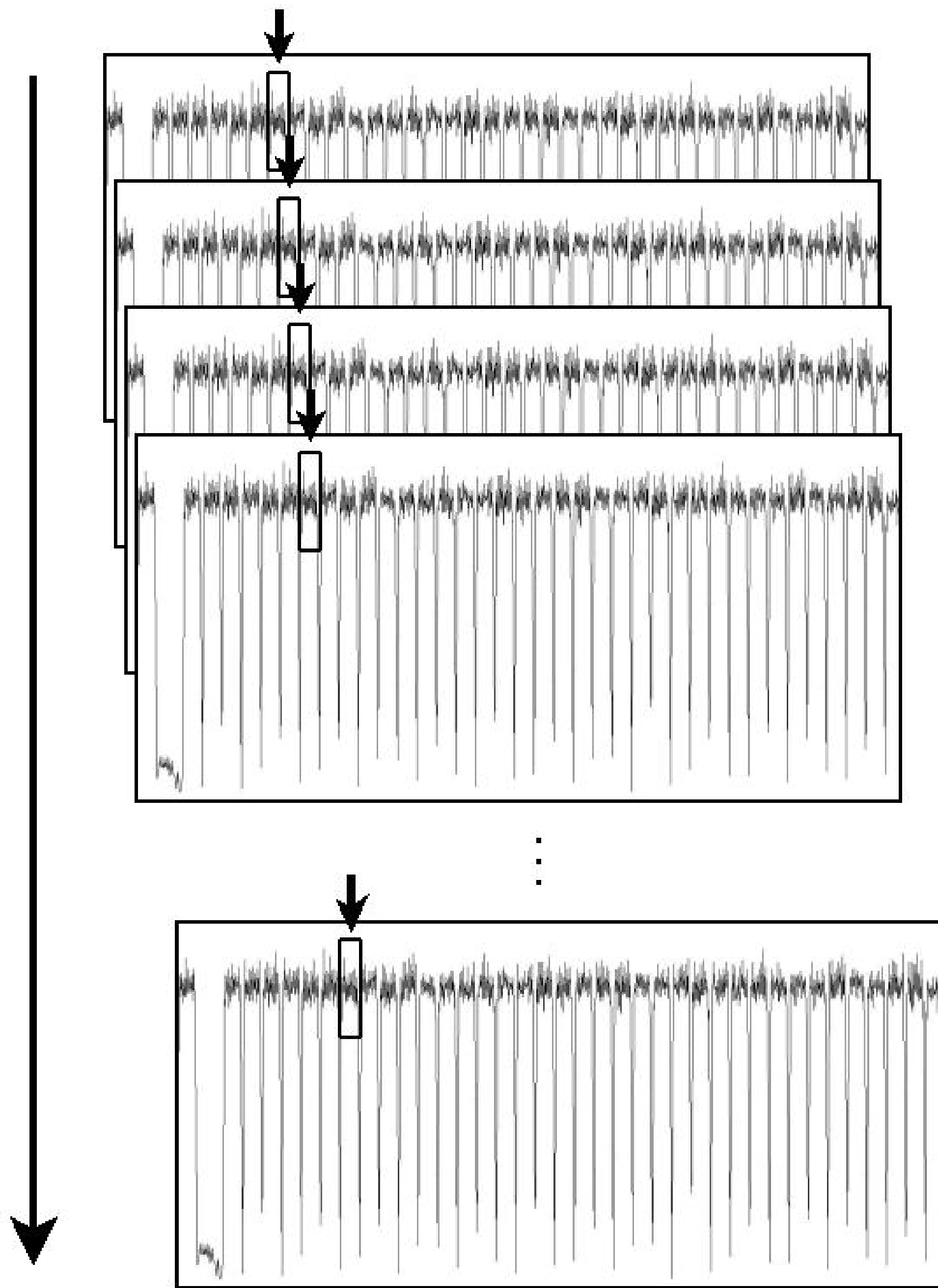


Figure 5.1: Vertical Side Channel Analysis

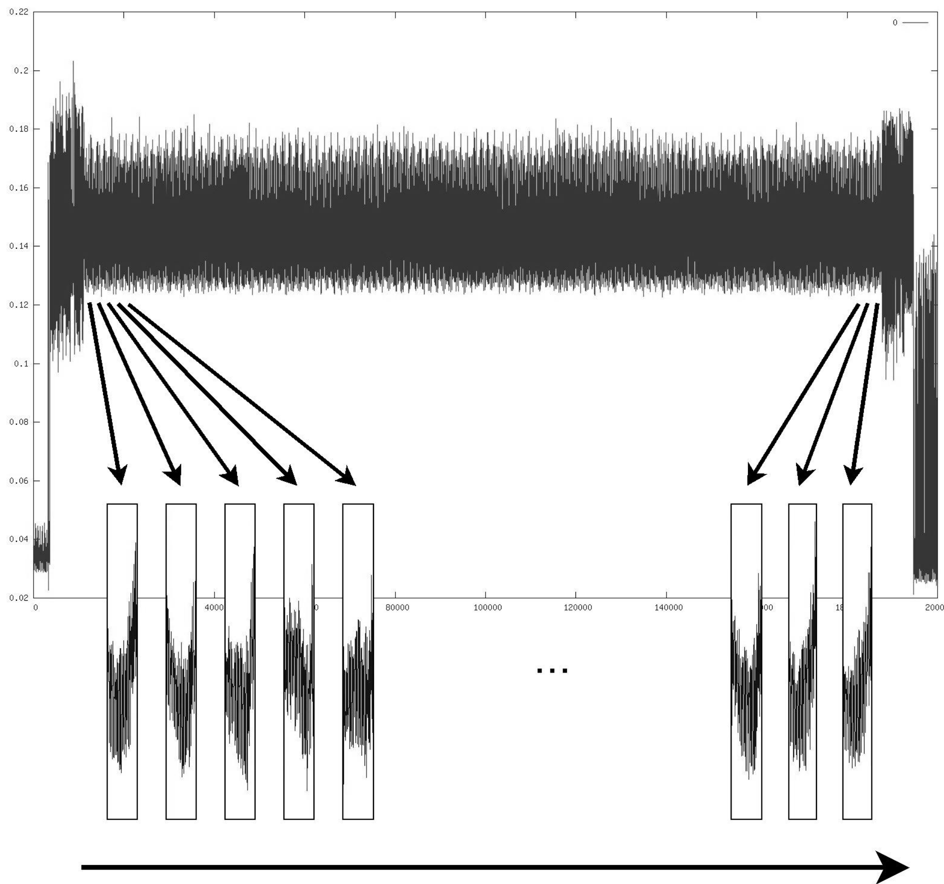


Figure 5.2: Horizontal side-channel analysis

We propose the *horizontal* side-channel analysis denomination for the attacks using a single curve. First known horizontal power analysis is the classical SPA. Single curve Cross-correlation and Big Mac attacks are also horizontal techniques.

Our attack, we present in the next section, computes the correlation factor on many curve segments extracted from a single consumption/radiation curve as depicted in Figure 5.2. It thus contrasts with vertical attacks which target a particular instant of the execution in several curves. The exponent blinding is not an efficient countermeasure against horizontal attacks.

5.4 Horizontal Correlation Analysis

We present hereafter our attack on an atomically protected RSA exponentiation using Barrett reduction.

5.4.1 Recovering the Secret Exponent with One Known Message Encryption

As in vertical DPA and CPA on modular exponentiation, the horizontal correlation analysis reveals the bits of the private exponent d one after another. Each exponent

bit is recovered by determining whether the processing of this bit involves a multiplication by m or not (cf. Alg. 5.2). The difference with classical vertical analysis lies in the way to build such hypothesis test. Computing the long integer multiplication $x \times y$ using Alg. 5.1 requires l^2 t-bit multiplier calls. The multiplication side-channel curve thus yields l^2 curve segments $C_{i,j}^k$ available to an attacker.

Assuming that the first s bits $d_{v-1}d_{v-2}\dots d_{v-s}$ of the exponent are already known, an attacker is able to compute the value a_s of the accumulator in Alg. 5.2 after processing the s-th bit. The processing of the first s bits corresponds to the first s' long integer multiplications with $s' = s + \mathrm{HW}(d_{v-1}d_{v-2}\dots d_{v-s})$ known by the attacker. The value of the unknown $(s+1)$-th exponent bit is then equal to 1 if and only if the $(s'+2)$-th long integer multiplication is ${a_s}^2 \times m$.

$C^{s'+1}$ $C^{s'+2}$

a_s — $d_{v-s-1} = 1$: $a_s \times a_s \longrightarrow {a_s}^2 \times m \quad \cdots$

a_s — $d_{v-s-1} = 0$: $a_s \times a_s \xrightarrow{d_{v-s-2} = 0,1} {a_s}^2 \times {a_s}^2 \quad \cdots$

At this point there are several ways of determining whether the multiplication by m is performed or not.

First, one may show that the series of consumptions in the set of l^2 curve segments is consistent with the series of operand values m_j presumably involved in each of these segments. To this purpose the attacker simply computes the correlation factor between the series of Hamming weights $\mathrm{HW}(m_j)$ and the series of curve segments $C_{i,j}^{s'+2}$ – i.e. taking $D = m_j$ and $R = 0$ in the correlation factor formula. In other words we use the curve segments as they would be in a vertical analysis if they were independent aligned curves. A correlation peak reveals that $d_{v-s-1} = 1$ since it occurs if and only if m is actually handled in this long multiplication.

Alternatively one may correlate the curves segments with the intermediate results of each t-bit multiplication $x_i \times y_j$, cf. Alg. 5.1, with $x = a_s$ and $y = m$, or in other words take $D = a_i \times m_j$. This method may also be appropriate since the words of the result are written in registers at the end of the operation. Moreover in that case l^2 different values are available for correlating the curve segments instead of l previously. This diversity of data may be necessary for the success of the attack when l is small. Note that other intermediate values may also lead to better results depending on the hardware leakages.

Another method consists of using the curve segments $C_{i,j}^{s'+3}$ of the next long integer multiplication and correlating them with the Hamming weight of the words of the result ${a_s}^2 \times m$. If the $(s'+2)$-th operation is a multiplication by m then the $(s'+3)$-th operation is a squaring ${a_{s+1}}^2$, manipulating the words of the integer ${a_s}^2 \times m$ in the t-bit multiplier. As pointed out by Walter in [105] for the Big Mac attack, the longer the integer manipulated is and the smaller the size t of the multiplier, the larger the number l^2 of curve segments will be. Thus longer keys are more at risk with respect to horizontal analysis. For instance in an RSA 2048-bit encryption, if the long integer multiplication is implemented using a 32-bit multiplier we obtain $(2048/32)^2 = 4096$ segments $C_{i,j}^k$ per curve C^k. In Appendix section 5.8 Table Table 5.1 proposes examples of values for l and l^2 for different sizes of the modulus n and different sizes t of the multiplier. *Remark* The series of Hamming weights

$\mathrm{HW}(m_j)$ is not only correlated with the series of curve segments in $C^{s'+2}$ (provided that $d_{v-s-1} = 1$), but also with the series of curve segments in each and any C^k

corresponding to a multiplication by m. Defining a *wide segment* $C_{i,j}^{*}$ as the set of segments $C_{i,j}^{k}$ for all k on the curve C and correlating the series of $\mathrm{HW}(m_j)$ with the series of wide segments $C_{i,j}^{*}$ (instead of the series of segments $C_{i,j}^{s'+2}$) will produce a wide segment correlation curve with a peak occurring for each k corresponding to a multiplication by the message. It is thus possible to determine in one shot the exact sequence of squarings and multiplications by m, revealing the whole private exponent with only one curve and only one correlation computation.

5.4.2 Practical Results

This section presents the successful experiments we conducted to demonstrate the efficiency of the horizontal correlation analysis technique. We used a 16-bit RISC microprocessor on which we implemented a software 16×16 bits long integer multiplication to simulate the behavior of a coprocessor. We aim at correlating a single long integer multiplication with one or both operands manipulated – i.e. y_j or $x_i \times y_j$.

The measurement bench is composed of a Lecroy Wavepro oscilloscope, and homemade softwares and electronic cards were used to acquire the power curves and process the attacks.

Firstly we performed a classical vertical correlation analysis to characterize our implementation and measurement bench, and to validate the correlation model; then we processed with the horizontal correlation analysis previously described.

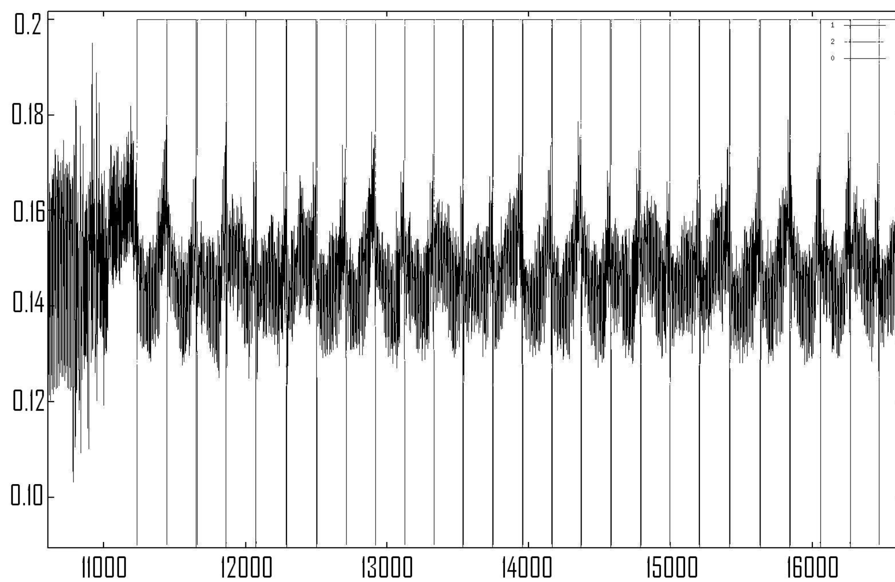


Figure 5.3: Beginning of a long integer multiplication power curve, lines delimitate each $C_{i,j}^{k}$

Vertical Correlation Analysis

This analysis succeeded in two cases during the operation $x \times y$. We obtained correlation peaks by correlating power curves with values x_i and y_j and also by

correlating the power curves with the result value of operation $x_i \times y_j$. Figure 5.4 and Figure 5.5 show the correlation traces we obtained for both cases with 500 power curves.

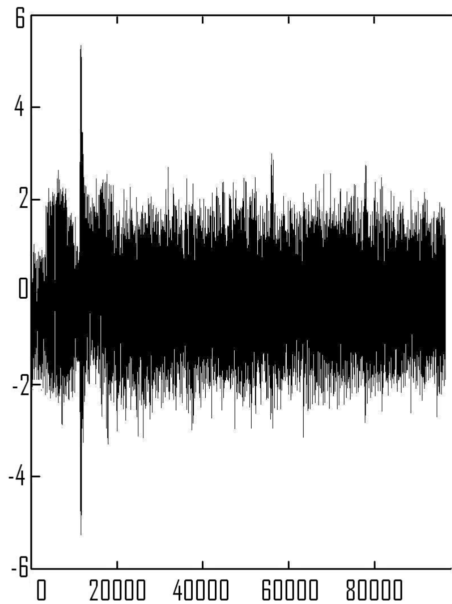


Figure 5.4: Vertical CPA on value y_j.

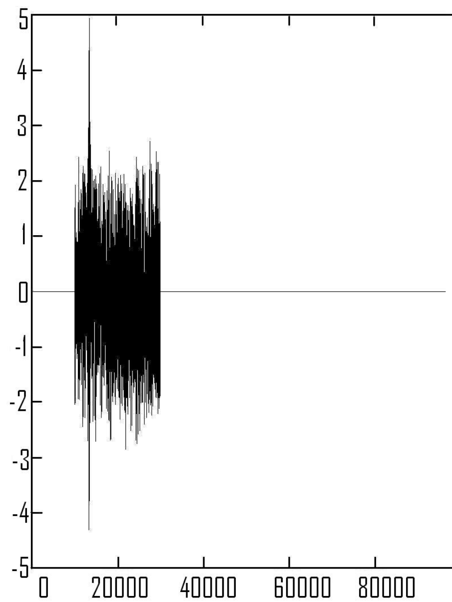


Figure 5.5: Vertical CPA on value $x_i \times y_j$.

This suggests that one can perform horizontal correlation as explained previously either using y_i values or using result values $x_i \times y_j$ for correlating with segment curves of the long integer multiplication.

Horizontal Correlation Analysis

We have chosen to test our technique within a 512-bit multiplication $\mathrm{LIM}(x, y)$. This allows us to obtain 1024 curve segments $C_{i,j}^k$ of 16-bit multiplications to mount the analysis, which should be enough for the success of our attack regarding the vertical analysis results. From the single power curve we acquired, we processed the signal in order to detect each set of cycles corresponding to each t-bit multiplication $x_i \times y_j$ and divide the single power curve in 1024 segments $C_{i,j}^k$ as depicted in Figure 5.3.

We performed horizontal correlation analysis as explained in section 5.4 for the two cases $D = a_i \times m_j$ and $D = m_j$ and recovered the operation executed as shown in Figure 5.6 and Figure 5.7. In each figure, the grey trace shows a greater correlation than the black one and thus corresponds to the correct guess on the operation.

Since our attack actually enabled us to distinguish one operation from another, it is then possible to identify a squaring $a \times a$ from a multiplication $a \times m$ in the Step 3 of Alg. 5.2. The secret exponent d used in an exponentiation can thus be recovered by using a single power trace, even when the exponentiation is protected by an atomic implementation.

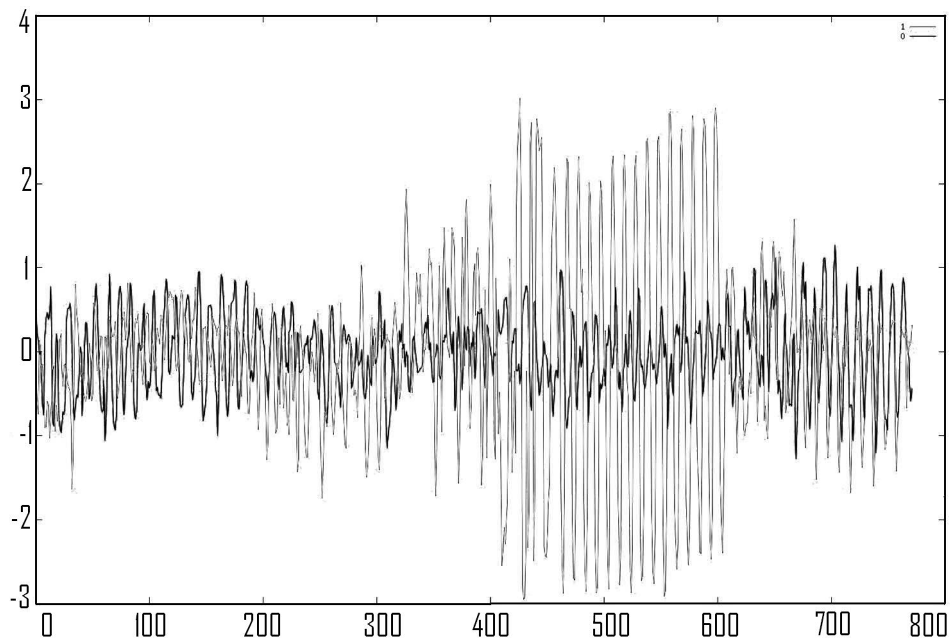


Figure 5.6: Horizontal CPA on value $a_i \times m_j$.

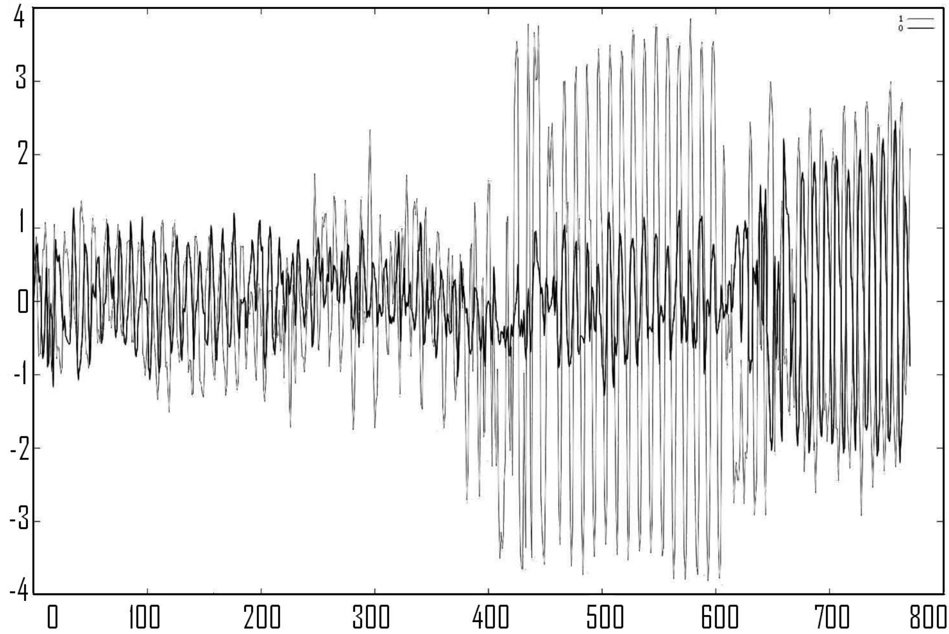


Figure 5.7: Horizontal CPA on value m_j.

We have presented here a technique to recover the secret exponent using a single curve when the input message is known and have proven this attack to be practically successful. Although the attack is tested on a software implementation, results obtained by Amiel et al. [8, Fig. 8] prove that correlation techniques are efficient on hardware coprocessors (with multiplier size larger than 16 bits), and enable to locate each little multiplication involved in a long integer multiplication. We thus consider that our attack can also threaten hardware coprocessors.

5.4.3 Comparing our Technique with the Big Mac Attack

We now compare our proposed horizontal CPA on exponentiation with the Big Mac attack which is the most powerful known horizontal analysis to recover a private exponent. A common property is that both techniques counteract the randomization of the exponent.

A first difference between both methods is that the Big Mac templates are generated by averaging the leakage dependency from a not targeted argument. It is thus implicitly accepted to lose the information brought by this auxiliary data. On the other hand, horizontal correlation exploits the knowledge of both multiplication operands a and m (under assumption on the exponent bit) to correlate it with all l^2 segments $C_{i,j}^k$. This full exploitation of the available information included in the l^2 curve segments tends us to expect a better efficiency of the correlation method particularly when processing noisy observations.

But the main difference is not there. What fundamentally separates the Big Mac and correlation methods is that the former deals with templates – which the attacker tries to identify – while the later rather consider intermediate results – whose manipulation validates a secret-dependent guess. With the Big Mac technique an attacker is able to answer the question *Is this operation of that particular kind?* (squaring, multiplication by m or a power thereof) while the correlation with

intermediate data not only brings the same information but also answers the more important question *Is the result of that operation involved in the sequel of the computation?* The main consequence is that horizontal CPA is effective even when the exponentiation implementation is *regular* with respect to the operation performed. This is notably the case of the *square and multiply always*[1] and the Montgomery ladder exponentiations which are not threaten by the Big Mac attack. In this respect we can say that our horizontal CPA combines both the advantage of classical CPA which is able to validate guesses based on the manipulation of intermediate results (but which is defeated by the randomization of the exponent) and that of horizontal techniques which are immune to exponent blinding.

On the other hand the limitation of the Big Mac attack – its ignorance of the intermediate results – is precisely the cause of its noticeable property to be applicable also when the base of the exponentiation is not known from the attacker. The Big Mac attack thus applies when the message is randomized and/or in the case of a Chinese Remainder Theorem (Chinese Remainder Theorem (CRT)) implementation of RSA. While the horizontal correlation technique does not intrinsically deals with message randomization, we give in the next section some hints that allow breaking those protected implementations when the random bit-length is not sufficiently large.

5.4.4 Horizontal Analysis on Blinded Exponentiation

To protect public-key implementations from SCA developers usually include blinding countermeasures in their cryptographic codes. The most popular ones on RSA exponentiation are:

- Additive randomization of the message and the modulus: $m^\star = m + r_1 \cdot n \bmod r_2 \cdot n = m + u \cdot n$ with r_1, r_2 being λ-bit random values different each time the computation is executed, and $u = r_1 \bmod r_2$.
- Multiplicative randomization of the message: $m^\star = r^e \cdot m \bmod n$ with r a random value and e the public exponent,
- Additive randomization of the exponent: $d^\star = d + r \cdot \phi(n)$ with r a random value.

All these countermeasures prevent from the classical vertical side-channel analysis but the efficiency of the implementations is penalized as the exponent and modulus are extended of the random used bit lengths.

Guessing the randomized message $m^\star$

In this paragraph we consider that the message has been randomized by an additive (or multiplicative) method, the secret exponent has also been randomized and the message is encrypted by an atomic multiply always exponentiation. We analyze the security of such implementation against horizontal CPA. The major difference with vertical side-channel analysis is that the exponent blinding has no effect since we analyze a single curve and recovering $d^\star$ is equivalent to recovering d.

[1] Referring to the description given in subsection 5.4.1 the method using the curve segments $C_{i,j}^{s'+3}$ validates that the value produced by the multiplication by m is involved or not in the next squaring operation. A similar technique also applies to the Montgomery ladder.

Assuming that the entropy of u is λ bits, there are 2^λ possible values for the message $m^\star$ knowing m and n. The first step of an attack is to deduce the value of the random u. This is achieved by performing one horizontal CPA for each possible value of u on the very first multiplication which computes $(m^\star)^2$. Since this multiplication is necessarily computed, the value of u should be retrieved as the one showing a correlation peak. Once u is recovered, the randomized message $m^\star$ is known and recovering the bits of the exponent d is similar to the non blinded case using $m^\star$ instead of m. Consequently, the entropy of u must be large enough (e.g. $\lambda \geq 32$) to make the number of guess unaffordable and prevent from horizontal correlation analysis.

Actual Entropy of Randomization

In the case of additive randomization of the message, $m^\star$ depends on two λ-bit random values r_1 and r_2. Obviously, the actual entropy of this randomization is not 2λ bits, and interestingly it is even strictly less than λ bits. The reason is that $m^\star = m + u \cdot n$ with $u = r_1 \bmod r_2$, and thus smaller u values are more probable than larger ones.

Assuming that r_1 and r_2 are uniformly drawn at random in the ranges $[0, \ldots, 2^\lambda - 1]$ and $[1, \ldots, 2^\lambda - 1]$ respectively, statistical experiments show that the actual entropy of u is about $\lambda - 0.75$ bits[2].

A consequence of this bias on the random u is that an attacker can exhaust only a subset of the smaller guesses about u. If the attack does not succeed, then he can try again on another exponentiation curve. For $\lambda = 8$ guessing only the 41 smaller u will succeed with probability $\frac{1}{2}$.

An extreme case, which optimizes the average number of correlation curve computations, is to guess only the value $u = 0$ [3]. This way, only 38 and 5352 correlation curve computations are needed in the mean when λ is equal to 8 and 16 respectively.

These observations demonstrate that the guessing attack described in the previous paragraph is more efficient than may be trivially expected. This confirms the need to use a large random bit length λ.

5.5 Countermeasures

Having detailed the principle and the threats of our horizontal side-channel analysis on exponentiation, we now study the real efficiency of the classical side channel countermeasures and propose new countermeasures.

5.5.1 Hardware Countermeasures

Classical countermeasures consisting in perturbing the signal analysis e.g. clock jitters, frequency clock dividers or dummy cycles, may considerably complicate

[2]The loss of 0.75 bits of entropy is nearly independent of λ for typical values ($\lambda \leq 64$).

[3]Or $u = 1$ if the implementation does not allow $u = 0$.

the analysis but should not be the only countermeasures since efficient signal processing could bypass them depending on their real efficiency.

Techniques consisting in balancing the power consumption of the chip with dual rail, precharge logics or other methods, if really efficient, could be a better solution. However they are expensive countermeasures from the chip surface point-of-view.

5.5.2 Blinding

All SSCA resistant algorithms that can be used to implement the exponentiation – either those protected with atomicity principle or regular ones as *square and multiply always* and Montgomery ladder – are threatened by the horizontal analysis. It is then necessary to randomize the data manipulated during the computation. As said previously the blinding of the exponent is not an efficient countermeasure here, it is thus highly recommended to implement a resistant and efficient blinding method on the data manipulated, for instance by using additive message randomization with random values larger or equal to 32 bits. As regard to the previous analysis on the actual entropy of u, an additional solution consists in eliminating the bias on u by setting r_2 to a constant value, for instance $2^\lambda - 1$.

5.5.3 New Countermeasures

We suggest protecting sensitive implementations from this analysis by introducing blinding into the t-bit multiplications, by randomizing their execution order or by mixing both solutions. These countermeasures are presented on modular multiplication using the Barrett reduction.

Blind Operands in LIM

A full blinding countermeasure on the words x_i and y_j consists in replacing in Alg. 5.1 the operation $(w_{i+j} + x_i \times y_j) + c$ by $(w_{i+j} + (x_i - r_1) \times (y_j - r_2)) + r_1 \times y_j + r_2 \times x_i - r_1 \times r_2 + c$ with r_1 and r_2 two t-bit random values. For efficiency purposes, the values $r_1 \times x_i$, $r_2 \times y_j$, $r_1 \times r_2$ should be computed once and stored. Moreover, these precomputations must also be protected from correlation analysis. For example, performing them in a random order yields $(2l+1)!$ different possibilities. In this case the LIM operation requires $l^2 + 2l + 1$ t-bit multiplications and needs $2(n+2t)$ bits of additional storage.

In the following we improve this countermeasure by mixing the data blinding with a randomization of the order of the internal loops of the long integer multiplication.

Randomize One Loop in LIM and Blind

This countermeasure consists in randomizing the way the words x_i are taken by the long integer multiplication algorithm. In other words it randomizes the order of the lines of the schoolbook multiplication. Then computing correlation between

x_i and $C^k_{i,j}$ does not yield the expected result anymore. On the other hand it remains necessary to blind the words of y. An example of implementation is given in Alg. 5.3.

The random permutation provides $l!$ different possibilities for the execution order of the first loop. For example, using a 32-bit multiplier, a 1024-bit long integer multiplication has about 2^{117} possible execution orders of the first loop and with 2048-bit operands it comes to about 2^{296} possibilities.

Alg. 5.3 LIM with lines randomization and blinding

1: $x = (x_{l-1}x_{l-2}\dots x_1x_0)_b, y = (y_{l-1}y_{l-2}\dots y_1y_0)_b$
2: LinesRandLIM(x,y) $= x \times y$
3: Draw a random permutation vector $\alpha = (\alpha_{l-1}\dots\alpha_0)$ in $[0, l-1]$
4: Draw a random value r in $[1, 2^t - 1]$
5: **for** $i = 0$ to $2l - 1$ **do**
6:     $w_i = 0$
7: **end for**
8: **for** $h = 0$ to $l - 1$ **do**
9:     $i \leftarrow \alpha_h$, $r_i \leftarrow r \times x_i$ and $c \leftarrow 0$
10:     **for** $j = 0$ to $l - 1$ **do**
11:         $(uv)_b \leftarrow (w_{i+j} + x_i \times (y_j - r) + c) + r_i$
12:         $w_{i+j} \leftarrow v$ and $c \leftarrow u$
13:         **while** $c \neq 0$ **do**
14:             $uv \leftarrow w_{i+j} + c$
15:             $w_{i+j} \leftarrow v$, $c \leftarrow u$ and $j \leftarrow j + 1$
16:         **end while**
17:     **end for**
18: **end for**
19: **return** w

Compared to the previous countermeasure, Alg. 5.3 requires only $l^2 + l$ t-bit multiplications and $2t$ bits of additional storage.

Remark One may argue that in the case of very small l values such a countermeasure might not be efficient. Remember here that if l is very small, the horizontal correlation analysis is not efficient either because of the small number of curve segments.

Randomize the Two Loops in LIM

We propose a variant of the previous countermeasure in which the execution order of the both internal loops of the long integer multiplication are randomized. This means randomizing both lines and columns of the schoolbook multiplication. The main advantage is that none of the operands x_i or y_j needs to be blinded anymore. The number of possibilities for the order of the l^2 internal multiplication is increased to $(l!)^2$. An example of implementation is given in Alg. 5.4.

Unlike the two previous countermeasures, Alg. 5.4 requires no extra t-bit multiplication compared to LIM. It is then an efficient and interesting countermeasure, while the remaining difficulty for designers consists in implementing it in hardware.

Alg. 5.4 LIM with lines and columns randomization

1: $x = (x_{l-1}x_{l-2}\dots x_1x_0)_b, y = (y_{l-1}y_{l-2}\dots y_1y_0)_b$
2: MatrixRandLIM(x,y) $= x \times y$
3: Draw two random permutation vectors α, β in $[0, l-1]$
4: **for** $i = 0$ to $2l - 1$ **do**
5: $\quad c_j = 0$
6: **end for**
7: **for** $h = 0$ to $l - 1$ **do**
8: $\quad i \leftarrow \alpha_h$
9: $\quad$ **for** $j = 0$ to $2l - 1$ **do**
10: $\quad\quad c_j = 0$
11: $\quad$ **end for**
12: $\quad$ **for** $k = 0$ to $l - 1$ **do**
13: $\quad\quad j \leftarrow \beta_k$
14: $\quad\quad (uv)_b \leftarrow w_{i+j} + x_i \times y_j$
15: $\quad\quad w_{i+j} \leftarrow v$ and $c_{i+j+1} \leftarrow u$
16: $\quad\quad u \leftarrow 0$
17: $\quad$ **end for**
18: $\quad$ **for** $s = i + 1$ to $2l - 1$ **do**
19: $\quad\quad (uv)_b \leftarrow w_s + c_s + u$
20: $\quad\quad w_s \leftarrow v$
21: $\quad$ **end for**
22: **end for**
23: **return** (w)

5.6 Concerns for Common Cryptosystems

We presented our analysis on straightforward implementations of the RSA signature and decryption algorithms which essentially consist of an exponentiation with the secret exponent. In the case of an RSA exponentiation using the CRT method our technique cannot be applied since the operations are performed modulo p and q which are unknown to the attacker. On the other hand DSA and Diffie-Hellman exponentiations were until now considered immune to DPA and CPA because the exponents are chosen at random for each execution. Indeed it naturally protects these cryptosystems from vertical analysis. However, as horizontal CPA requires a single execution power trace to recover the secret exponent, DSA and Diffie-Hellman exponentiations are prone to this attack and other countermeasures must be used in embedded implementations. It is worth noticing that ECC cryptosystems are theoretically also concerned by the horizontal side-channel analysis. However since key lengths are considerably shorter – for instance ECC 224 bits is considered having equivalent mathematical resistance than RSA 2048 – very few curves per scalar multiplication will be available for the attack. On the other hand, scalar multiplication involves point doublings and point additions instead of field multiplications and squarings. Each point operation requires about 10 modular multiplications and thus correlation computation could take advantage of all the corresponding curves. Nevertheless, a factor of about 10 should not balance the key length reduction which has a quadratic influence on the number of available curve segments.

5.7 Square and Multiply Power Curves Examples

The following figures illustrate the SPA as described in section 5.3.1. Figure 5.8 corresponds to the power curve execution of a classical square and multiply algorithm. We can observe that the multiplication operation has a different pattern from the squaring one. The multiplications can then be identified and the secret exponent be recovered.

On Figure 5.9, the implementation analyzed uses the atomicity principle. It allows the exponentiation to be SSCA resistant since we cannot distinguish anymore the squarings from the multiplications.

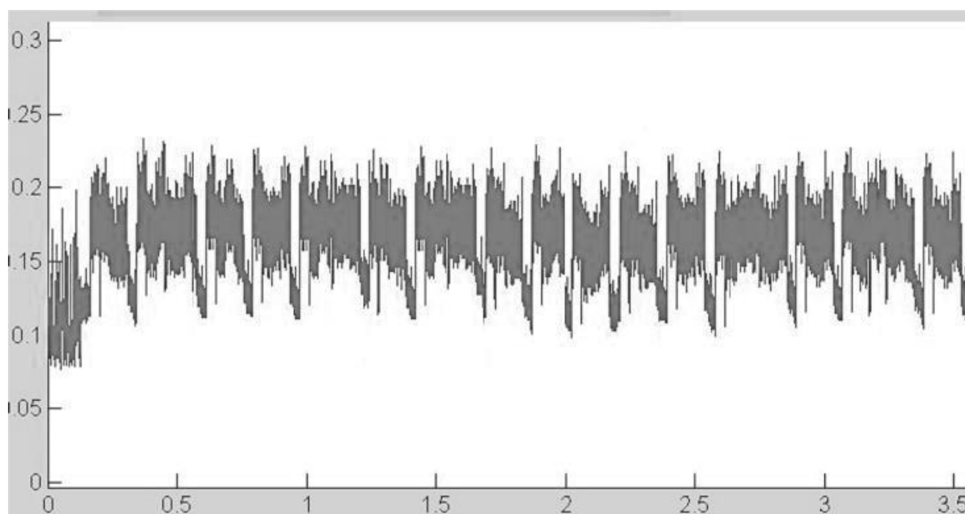


Figure 5.8: Power curve of a leaking square and multiply algorithm

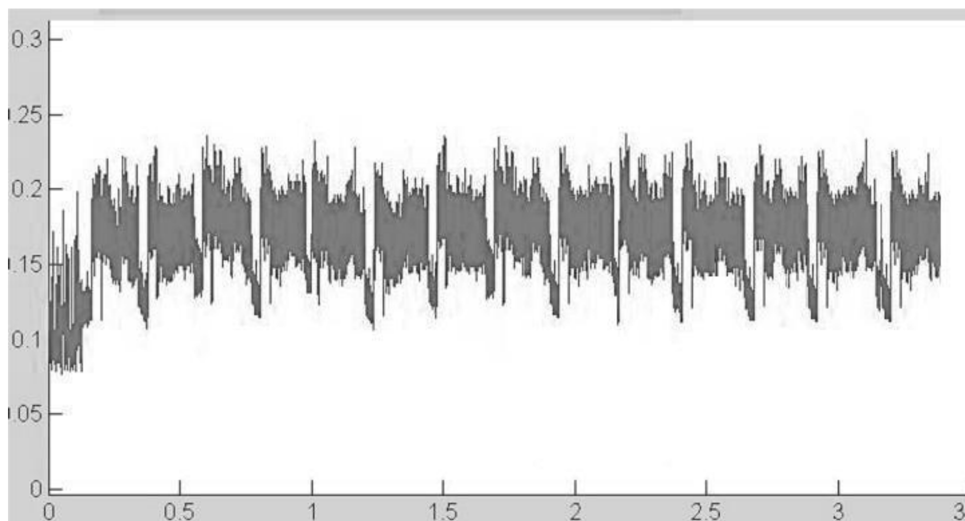


Figure 5.9: Power curve of an atomic square and multiply algorithm

5.8 Examples of l and l^2 values

In this paragraph, we illustrate with examples the property: the longer the keys, the more efficient the horizontal correlation analysis. The following table proposes examples of values for l and l^2 using different key lengths and different multiplier sizes.

length n in bits	multiplier size t	l	l^2
2048	32	64	4096
2048	64	32	1024
1536	32	48	2304
1536	64	24	576
1024	16	64	4096
1024	32	32	1024
1024	64	16	256
512	16	32	1024
512	32	16	256

Table 5.1: Examples of n, t, and l values with the number of available segments l^2

Considering that $n \geq 500$ should be enough to perform the horizontal correlation analysis, many implementations may be subject to this attack.

Chapter 6

Improved Collision-Correlation Power Analysis on AES

6.1 Introduction

Side-channel analysis was introduced by Kocher et al. in 1998 [61] and marks the outbreak of this new research field in the applied cryptography area. Meanwhile, many side-channel techniques have been published. For example Brier et al. proposed the Correlation Power Analysis (CPA) [20] which has shown to be very efficient as it significantly reduces the number of curves needed for recovering a secret key, and more recently the Mutual Information Analysis from Gierlichs et al. [45] has generated a lot of interest.

Since side-channel attacks potentially concern any kind of embedded implementations of symmetric or asymmetric algorithms, it is recommended to apply various masking countermeasures (among others) in sensitive products [2, 73]. Second-order or higher-order side-channel analysis can however defeat such countermeasures by combining leakages from different instants of the execution of an algorithm and canceling the effect of a mask [66, 71]. Such attacks are considered very difficult to implement and generally require an important number of power curves.

A specific approach for side-channel analysis is using information leakages to detect collisions between data manipulated in algorithms. Side-channel collision attacks against a block cipher were first proposed by Schramm et al. in 2003 [93]. Their attack uses differential analysis to exploit collisions in adjacent S-Boxes of the DES algorithm. In [92] an attack against the AES is proposed to detect collisions in the output of the first round `MixColumns`. Later, Bogdanov [17] improved this attack by looking for equal S-Boxes inputs in several AES executions. He then studied in [18] statistical techniques to detect collisions between power curves. Two recent papers have updated the state-of-the-art by introducing correlation based collision detection: Moradi et al. [78] proposed a collision attack to defeat an AES implementation using masked S-Boxes, while Witteman et al. [106] applied a cross-correlation analysis to an RSA implementation using message blinding.

In this chapter, we present two collision-correlation attacks on software AES implementations protected against first-order power analysis using masked S-Boxes and practical results on both simulated and real power curves. Our attacks are

much more efficient and generic compared to the one presented in [78]. Moreover we believe our techniques to be applicable to other embedded implementations of symmetric block ciphers.

The remainder of the chapter is organized as follows: The section 6.2 presents the two AES first-order protected implementations targeted by our study. Then in section 6.3 we present our attacks and practical results on simulated power curves and on a physical integrated circuit. In section 6.4 we compare our technique with second-order power analysis and section 6.5 deals with the possible countermeasures.

6.2 Targeted Implementations

The AES Algorithm.

For the sake of simplicity in this contribution we focus on the AES-128 which includes 10 rounds, each one decomposed into four functions: `AddRoundKey`, `SubBytes`, `ShiftRows` and `MixColumns`. It encrypts a 128-bit message $M = (m_0, \ldots, m_{15})$ using a 128-bit secret key $K = (k_0, \ldots, k_{15})$ and produces a 128-bit ciphertext $C = (c_0, \ldots, c_{15})$. Note however that the techniques presented in this contribution are easily applicable to AES-192 and AES-256.

The only non-linear function of the AES is `SubBytes` (also referred to as the S-Boxes S in the following) which is a substitution function defined by the pseudo-inversion I in $\mathrm{GF}(2^8)$ and an affine transformation. In this chapter, we consider the two following solutions that have been proposed to protect this function against first-order attacks.

6.2.1 Blinded Lookup Table

The first targeted implementation uses a *masked* substitution table as proposed by Kocher et al. [62] and Akkar et al. [2]. This masked table S' is defined by $S'(x_i \oplus u_i) = S(x_i) \oplus v_i$, with u_i (resp. v_i) the mask of the i-th input byte x_i (resp. output byte) of function `SubBytes`, $x_i, y_i, u_i, v_i \in \mathrm{GF}(2^8)$, $0 \leq i \leq 15$. This table is usually computed before the AES execution and stored in volatile memory.

We further consider that the same masks u and v are applied on all S-Boxes during one execution (or a round at least) of the algorithm, i.e. $u_i = u$ and $v_i = v$ for $0 \leq i \leq 15$. We believe that this hypothesis is realistic for embedded security products considering that an expensive recomputation of the 256-byte substitution table S' is necessary for each new pair (u, v) and that the storage of many masked tables is not conceivable in memory constrained devices.

6.2.2 Blinded Inversion Calculation

An alternative solution has been proposed by Oswald et al. [79] and improved on by Canright et al. [21]. It consists in computing the inversion in $\mathrm{GF}(2^8)$ using a

multiplicative mask. To do this efficiently it is proposed to decompose the computation using inversions in the subfield $\mathrm{GF}(2^4)$ (and possibly in $\mathrm{GF}(2^2)$). Such masking method is well suited for hardware implementations.

We recall some properties of the masked inversion. Let I' denote the masked pseudo-inversion such that $I'(x_i \oplus u_i) = I(x_i) \oplus u_i$. The element $x_i \oplus u_i$ in $\mathrm{GF}(2^8)$ is mapped to a couple $(x_{i,h} \oplus u_{i,h}, x_{i,l} \oplus u_{i,l})$ of $\mathrm{GF}(2^4)$ such that $x_i \oplus u_i \cong (x_{i,h} \oplus u_{i,h})X + (x_{i,l} \oplus u_{i,l})$. As detailed in [79] many calculations occur on these subfield elements to compute the masked inversion of $x_i \oplus u_i$. The exact details of these computations can be found in [79]. Note that in these formulas neither $x_{i,h}$ nor $x_{i,l}$ is directly inversed in $\mathrm{GF}(2^4)$ but the following value:

$$d_i \oplus u_{i,h} = x_{i,h}^2 \times 14 \oplus (x_{i,h} \times x_{i,l}) \oplus x_{i,l}^2 \oplus u_{i,h} \ .$$

Then the masked inversion in $\mathrm{GF}(2^4)$ of $d_i \oplus u_{i,h}$ gives $d_i^{-1} \oplus u_{i,h}$ and is used to compute $I'(x_i \oplus u_i)$.

The 16 input bytes of `SubBytes` are blinded using different masks u_i, but one can notice that input and output masks of the inversion stage are identical. Therefore another threat to take into consideration is the zero value power analysis. This technique has been introduced in [49] and [65], and recently implemented on the masked inversion in [78]. Finally, note that the technique presented in this chapter also applies to the improved version of Canright et al. [21] when input and output of the inversion are masked with the same value.

6.2.3 Measurements and Validation of Implementations

Curve Acquisition.

We have developed software implementations on a contact smart card using a 16-bit RISC CPU with low power consumption. Two different methods were used to validate our attacks.

First, we used *simulated curves*: a proprietary tool was used to simulate power curves based on the chip architecture and the code executed. This tool generates ideal power consumption curves without any noise which enables to validate in practice the resistance of an implementation to a set of side-channel attacks leaving aside the acquisition and signal processing problems.

Second, we used *real curves*: we made physical measurements on the chip itself using a MicroPross MP100 reader and a Lecroy WavePro numerical oscilloscope.

First-Order Resistance Validation.

Since our aim was to present techniques able to defeat first-order protected devices, we performed the classical first-order differential and correlation analysis on the two implementations presented above, before testing our collision attacks.

To do so, we applied DPA and CPA on the `AddRoundKey`, `SubBytes` and `MixColumns` functions at the first and the last rounds of our implementations. We also performed detailed SPA for each input byte value using many average curves to detect any noticeable (biased) power traces that would reveal a potential leakage.

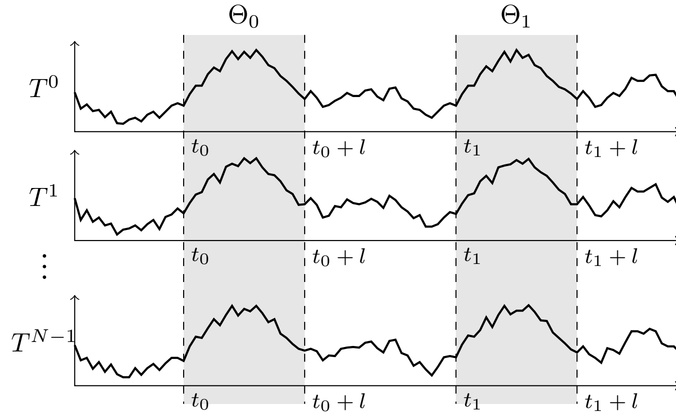


Figure 6.1: General description of the collision-correlation attack

In any case no leakage were observed. We also verified that both implementations were immune to zero value power analysis and to the attack presented by Moradi et al.

We have thus checked that to the best of our knowledge both considered AES implementations are resistant to known first-order attacks. Nevertheless we present in the next section two new collision-correlation techniques which jeopardize these implementations.

6.3 Description of our Attacks

In this section, we present the general principle of collision-correlation attacks and then detail how it can be applied on the two considered AES implementations.

6.3.1 The Collision-Correlation Method

The principle of the attacks presented in this chapter is to detect internal collisions between data processed in blinded S-Boxes on the first round of an AES execution. We demonstrate in the following that if i) we are able to detect that the same data is processed at instants t_0 and t_1, and ii) the S-Boxes are blinded such that either the same mask is applied to all message bytes or the mask is identical at the input and the output of each S-Box, then it is possible to infer information on the secret key with very few curves.

In the following, we will denote $(T^n)_{0 \leq n \leq N-1}$ a set of N power traces captured from a device processing N encryptions of the same message M. Then we consider two instructions[1] whose processing starts at times t_0 and t_1 and denote l the number of points acquired per instruction processing. As depicted in Figure 6.1 we finally consider $\Theta_0 = (T^n_{t_0})_n$ and $\Theta_1 = (T^n_{t_1})_n$ the two series of power consumptions segments at instants t_0 and t_1.

[1]In our attacks we only consider the correlation between two identical instructions, but it may even be possible to detect that two different instructions manipulate identical data, e.g. by spotting a data bus using EMA.

Note that in practice the N power curves should start at the same instant of the encryption and be perfectly aligned. Such conditions generally require signal processing to be performed first. Note also that as the sampling rate is usually such that $l > 1$ points are acquired per instruction, we can generalize the definition of Θ_0 and Θ_1 as being series of l-sample curve segments instead of series of single power consumption samples.

The final stage of the attack consists in applying a statistical treatment to (Θ_0, Θ_1) in order to identify if the same data was involved in $T_{t_0}^n$ and $T_{t_1}^n$ for $0 \leq n \leq N-1$. Let `Collision`(Θ_0, Θ_1) denote a decision function returning `true` or `false` depending on whether this property is presumed to be fulfilled or not. Such a decision function would usually compare the value of a synthetic criterion with a practically determined threshold. Possible examples of such a criterion include the mean[2] squared difference, the least squared difference with binary or ternary voting [18], and the maximum Pearson correlation factor. As we used this latter criterion in our study, we recall that an estimation of the Pearson correlation factor between series of curve segments Θ_0 and Θ_1 at time offsets t $(0 \leq t \leq l-1)$ is expressed as

$$\begin{aligned} \hat{\rho}_{\Theta_0,\Theta_1}(t) &= \frac{\mathrm{Cov}(\Theta_0(t), \Theta_1(t))}{\sigma_{\Theta_0(t)}\sigma_{\Theta_1(t)}} \\ &= \frac{N\sum(T_{t_0+t}^n T_{t_1+t}^n) - \sum T_{t_0+t}^n \sum T_{t_1+t}^n}{\sqrt{N\sum(T_{t_0+t}^n)^2 - (\sum T_{t_0+t}^n)^2}\sqrt{N\sum(T_{t_1+t}^n)^2 - (\sum T_{t_1+t}^n)^2}} \end{aligned}$$

where summations are taken over $0 \leq n \leq N-1$, and $\Theta_i(t) = (T_{t_i+t}^n)_n$ for $i \in \{0, 1\}$.

`Collision`(Θ_0, Θ_1) thus consists in comparing $\max_{0 \leq t \leq l-1}(\hat{\rho}_{\Theta_0,\Theta_1}(t))$ to a given threshold. In our experiments a preliminary characterization of the targeted device enabled us to find proper values for l and the threshold.

Note that in this collision-correlation technique we compute the correlation factor between a set of real power consumptions Θ_0 with another set of real power consumptions Θ_1, rather than with model dependent estimations. As Bogdanov already described in [18] about binary and ternary voting techniques, an interesting property of this method is that, unlike Hamming weight based CPA, our criterion does not rely on a particular leakage model. The consequences of this are that i) the attack is more generic and requires much less knowledge of the targeted device, and ii) the secret S-Boxes may be attacked as well as known ones.

As said above, correlating two instants (curve segments) on different traces has already been applied by Moradi et al. [78] on a particular AES implementation. However they collect many traces obtained by encrypting random messages and average them according to the value of an S-Box input byte. This results in 2^8 averaged curves for each byte position, from which they try to detect collisions between two bytes. They successfully carried out this attack on their implementation of the Canright et al. [21] first-order protected implementation. However as indicated by the authors their implementation presented a remaining first-order leakage based on zero-value attack. We applied Moradi's attack to the first-order protected implementations considered in this study without success. We thus consider that this attack is not applicable to most first-order protected implementations. Indeed averaging different traces implies the use of new random mask values which should spoil the influence of the unmasked data and make the collision of intermediate values undetectable. The technique we develop in this chapter improves on

[2] The mean being taken over the N traces as well as over the l samples.

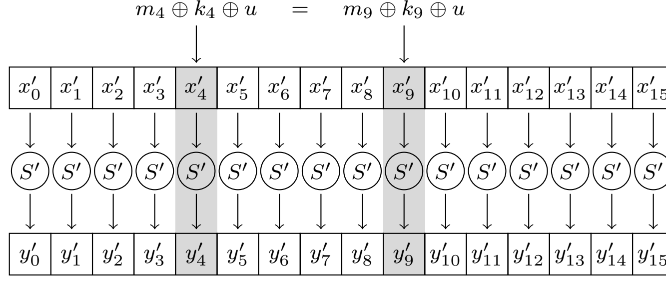


Figure 6.2: Collision between the computation of two S-Boxes on bytes 4 and 9 on the blinded lookup table implementation

Moradi's attack in order to detect data collisions by comparing two instants on a same trace and repeating it on many executions without the destructive averaging process. In the following we detail two applications of our attack on two different implementations.

Remark Collision based analyzes are also known as *cross-correlation attacks* in [106] and *multiple-differential collision attacks* in [18]. We prefer the term *collision-correlation attacks* since cross-correlation may be ambiguous depending on the context, and multiple-differential collision attacks seems us too generic for our method.

6.3.2 Attack on the Blinded Lookup Table Implementation

First, we present an application using principle presented above on the implementation described in subsection 6.2.1. This attack targets the execution of the first round SubBytes function. Each 16 masked input byte $x'_i = x_i \oplus u$ is substituted by a masked output byte $y'_i = y_i \oplus v$ where $y'_i = S'(x'_i)$. We try to detect when two SubBytes inputs (and outputs) are equal within the first AES round as depicted on Figure 6.2.

Detecting a collision in the first AES round between bytes i_1 and i_2 yields that $x_{i_1} \oplus u = x_{i_2} \oplus u$ and considering that $x_i = m_i \oplus k_i \oplus u$ implies the following relation of the two involved key bytes:

$$k_{i_1} \oplus k_{i_2} = m_{i_1} \oplus m_{i_2} \; . \tag{6.1}$$

Description.

Practically, we encrypted N times the same message M and collected the N traces corresponding to the first AES round. For each of the N traces we identified the 16 instants t_i corresponding to the beginning of the computation $S'(x_i \oplus u)$. This allowed us to extract 16 segments from each trace and construct the series Θ_i used for collision-correlation as explained in subsection 6.3.1.

Performing $\texttt{Collision}(\Theta_{i_1}, \Theta_{i_2})$ for all the 120 possible pairs (i_1, i_2) yields a set of relations $(i_1, i_2, m_{i_1} \oplus m_{i_2})$ given by Eq. (Equation 6.1). By repeating this process for several random messages M one can accumulate enough relations so that the secret key is recovered up to a guess on one key byte.

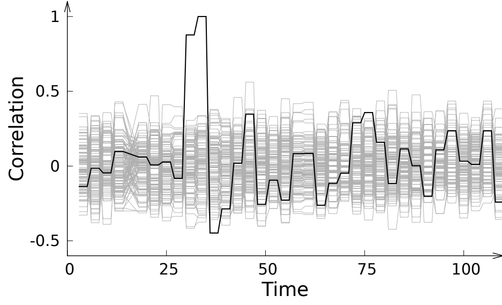


Figure 6.3: Correlation curves obtained for a message giving one collision (black curve)

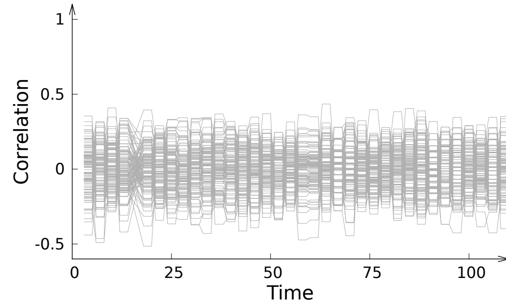


Figure 6.4: Correlation curves obtained for a message giving no collision

Based on 10000 simulations we observed that on average 59 random messages (each one being encrypted N times) provide enough relations to retrieve the key up to an unknown byte.

Practical Results.

We present hereafter our results on both simulated and real curves.

On simulated curves. The threshold of `Collision` was fixed to having at least one point among the l points correlation curve equal to 1. Under this condition our attack was successful for $N = 16$. Since a mean of 59 different messages are required, then $16 \times 59 = 944$ traces are sufficient on the average for the attack to succeed on simulated curves.

Figures Figure 6.3 and Figure 6.4 show the correlation curves obtained for two different messages. Both figures present the 120 outputs of $\hat{\rho}_{\Theta_{i_1},\Theta_{i_2}}(t)$, $i_1 < i_2$ for each message. The black curve on Fig. Figure 6.3 corresponds to a collision found for the first message, whereas the second message yields no collision.

On real curves. The attack was successful using $N = 25$ so that less than $1\,500$ traces allow to recover the key. Notice how few traces are needed to detect a collision by correlation. This confirms that the collision-correlation technique is much more efficient than classical model-based CPA which would not obtain high correlation levels with only 25 traces. The Figure 6.5 shows an example of a correlation peak when an equality between two S-Box outputs occurs, while Figure 6.6 shows the correlation curve when all S-Box outputs are different.

Note that in the case of real curves the threshold is slightly different. To identify a clear relation between two S-Box outputs the correlation curve must be greater than 0.8 in the interval $[130, 160]$. So only these $l = 30$ points must be considered when computing $\texttt{Collision}(\Theta_0, \Theta_1)$.

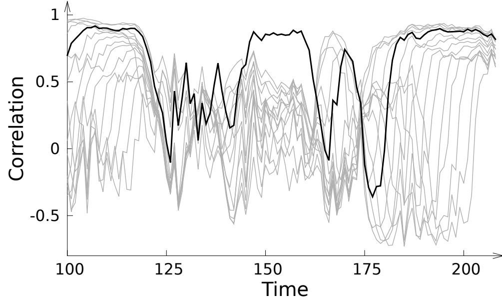


Figure 6.5: Correlation peak on real curves when a collision occurs (black curve)

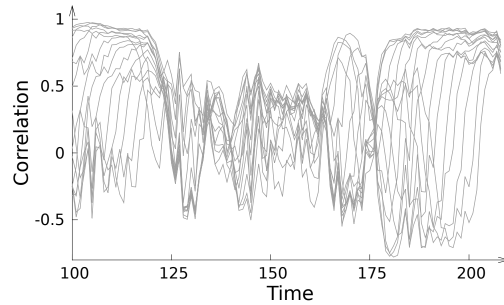


Figure 6.6: No correlation peak occurs on real curves when intermediate data differ

Attack Improvement.

The method for obtaining information about the key as described above basically exploits collision events where a pair (i_1, i_2) of indices gives a high correlation between Θ_{i_1} and Θ_{i_2} revealing the value of $k_{i_1} \oplus k_{i_2}$. While very informative, such collision events occur much less frequently than non-collision ones, that is when Θ_{i_1} and Θ_{i_2} show no significant correlation between each other. Non-collision events individually bring quite few information – namely that $k_{i_1} \oplus k_{i_2}$ is different from $m_{i_1} \oplus m_{i_2}$ – but they are so numerous that it appears worth trying to exploit them also.

As was already noted in [26, 17], the problem of solving a set of equations involving sub-parts of the key can be formulated in terms of a labelled undirected graph. Each vertex i represents a key byte index and the knowledge of the XOR between two key bytes is represented by an edge (i_1, i_2) labelled with $k_{i_1} \oplus k_{i_2}$. At the beginning the graph does not include any edges. Each time a collision occurs between two unrelated key bytes a new edge is put on the graph and results in the merge of two connected components into a single larger one. All key byte values belonging to the same connected component can be derived from each other, and the goal of the attacker is to end up with a fully connected graph.

For a given message, only 0, 1, or 2 from the 120 pairs (i_1, i_2) lead to collisions in most cases. All other pairs reveal some impossible value for each $k_{i_1} \oplus k_{i_2}$. Gathering all the information provided by these non-collisions, for each (i_1, i_2) we maintain a blacklist of impossible values for the XOR of the two key bytes[3].

Given the information provided by previous messages to the current graph and blacklists, we adaptively choose the next message in order to maximize its usefulness which we define as the number of pairs (i_1, i_2) where one can expect new information (either positive or negative) to be obtained. As a first idea we could define the penalty of a candidate message as the number of pairs (i_1, i_2) for which $m_{i_1} \oplus m_{i_2}$ is already blacklisted. Obviously the chosen message should minimize the penalty. Actually this is slightly more complex and the definition of the penalty of a message should be refined. Indeed we must also consider cases where the message is useful for (i_1, i_2) and (i_1, i'_2) – that neither $m_{i_1} \oplus m_{i_2}$ nor $m_{i_1} \oplus m_{i'_2}$ are blacklisted – but the value of $k_{i_2} \oplus k_{i'_2}$ is known to be precisely equal to $m_{i_2} \oplus m_{i'_2}$. In such a case the two usefulness opportunities brought by the message on pairs

[3]Some of these blacklists must also be updated when two connected components are merged.

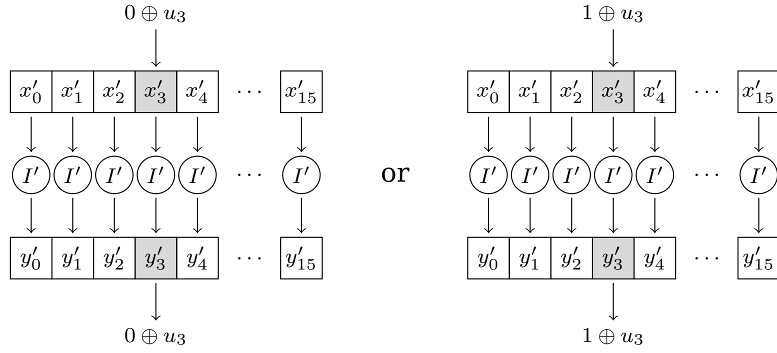


Figure 6.7: Collision between the input and the output on byte 3 of the blinded inversion I' (values 0 and 1 lead to a collision)

(i_1, i_2) and (i_1, i_2') would bring the same information so that they should count for a single one and the penalty of that message must be increased by one.

In order to find a message with minimal penalty we devised a heuristic which works in two steps. In the first step we consider some random messages (say a few hundred) and select the one with the lowest penalty. This first step ends with a somewhat good candidate. Then in a second step we repeatedly attempt to decrease further the penalty by trying small modifications on this candidate until no more improvements occur by small modifications.

We simulated our method for adaptively choosing the messages. In these simulations we assumed that the attacker is always able to correctly distinguish between collision and non-collision events. Based on 1 000 simulations with random keys, we show that the key is fully recovered (up to the knowledge of one of its bytes) with as few as 27.5 messages instead of 59 messages with the basic method. As distinguishing between a collision and a non-collision necessitates only 25 traces per message, a mere 700 executions would suffice to recover the key by analysing real curves.

6.3.3 Attack on the Blinded Inversion Implementation

The previous attack cannot be applied to the blinded inversion implementation described in subsection 6.2.2 since the different S-Box input and output bytes are masked with different values u_i. However there may exist a possible leakage leading to what we may call a *Zero & One value attack*.

One can notice that values 0 and 1 produce a collision between the input and the output of the masked pseudo-inversion stage I' as depicted on Figure 6.7. This is due to the following properties of the pseudo-inversion:

$$
\begin{aligned}
I(0) = 0 \quad &\Rightarrow \quad I'(0 \oplus u_i) = 0 \oplus u_i \\
I(1) = 1 \quad &\Rightarrow \quad I'(1 \oplus u_i) = 1 \oplus u_i
\end{aligned}
$$

The two cases leading to a collision are indistinguishable from one another. Detecting a collision between the input and the output of a blinded inversion gives either $x_i' = 0 \oplus u_i$ or $x_i' = 1 \oplus u_i$ which reveals a key byte except one bit:

$$k_i = m_i \quad \text{or} \quad k_i = m_i \oplus 1 \; .$$

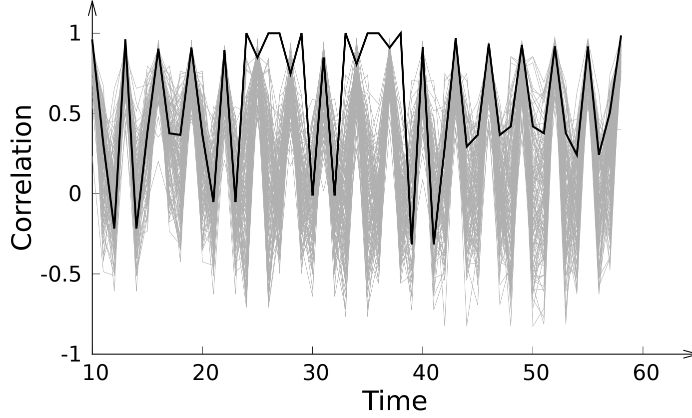


Figure 6.8: Collision-correlation curves in the pseudo-inversion of the first byte in $\mathrm{GF}(2^8)$

Description.

Assume we want to recover the 7 most significant bits of k_0. For every even byte value g we encrypt N times a single message M with $m_0 = g$ and collect the corresponding power consumption traces $T^{n,g}$, $0 \leq n \leq N-1$. Note that in this attack we only need to guess the 7 most significant bits because the least significant one is indistinguishable. Let's denote t_0 and t_1 the instants when $x_0 \oplus u_0$ is loaded before the pseudo-inversion I, and when the result is stored respectively. For each of the N traces we extract the two segments $T^{n,g}_{[t_0,t_0+l-1]}$ and $T^{n,g}_{[t_1,t_1+l-1]}$ and construct the series $\Theta^g_0 = (T^{n,g}_{[t_0,t_0+l-1]})_n$ and $\Theta^g_1 = (T^{n,g}_{[t_1,t_1+l-1]})_n$. For this step of our attack it is helpful to have some experience on the targeted implementation identify exactly where these two segments are located.

Applying the decision function $\texttt{Collision}(\Theta^g_0, \Theta^g_1)$ for all the 128 possible values g will reveal two possibilities for k_0. Repeating this step for all key bytes allows the key space to be reduced to 2^{16} values only. Note that a trick which allows to considerably reduce the number of traces is to encrypt the messages $M^g = (g, g, \ldots, g)$ with all bytes equal.

Results on Simulated Curves.

As for previous attack on simulated curves, a relation is established when at least one point among the l points correlation curve is equal to 1. The attack is successful using $N = 16$ curves for each key guess. The Figure 6.8 shows the 128 correlation curves for all possible guesses on k_0. The black curve corresponds to the correct guess for k_0.

The attack on this second implementation has thus been validated on simulated curves. We did not acquire real curves for this implementation. Based on what has been observed on the previous attack (successful results obtained using simulations have led to successful results on the chip in practice), we believe that the attack would be successful on the real chip too, using a value for N of the same order to what was necessary for the first attack.

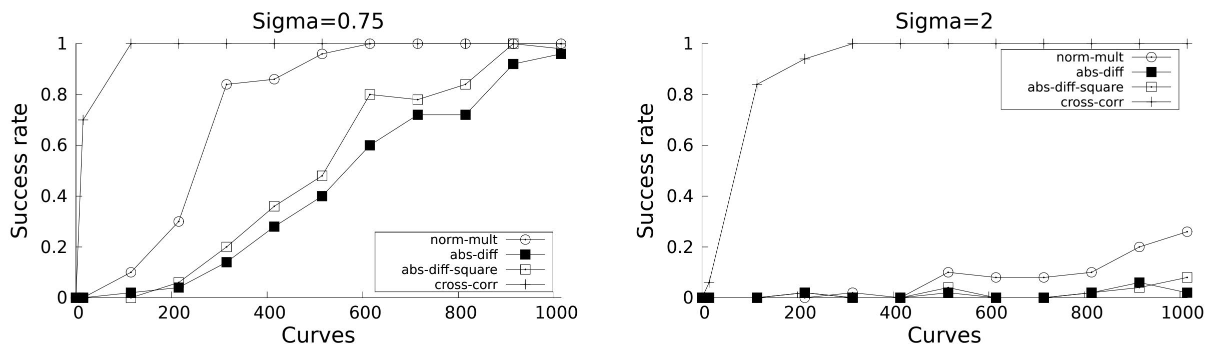


Figure 6.9: Success rates of different simulated second-order attacks

6.4 Comparison with Second Order Analysis

In this section, we present a brief comparison between the collision-correlation method and some known second-order attacks. Our analysis was inspired from the recent framework introduced by Standaert et al. in [100] and refined later in [99]. This comparison gives an overview on the efficiency of these different second-order techniques, and highlights how much the collision-correlation analysis improves on second-order attacks.

Our analysis targets the first implementation only. We compared the collision-correlation analysis with the second-order analysis involving the absolute difference combining function f_1, the squared absolute difference combining function f_2 and the normalized product combining function f_3, when using as distinguisher the Pearson linear correlation factor $\hat{\rho}$. Note that we did not used Mutual Information Analysis, whose results remain less efficient than the classical CPA in practice.

For sake of simplicity, we consider that the power consumption at instant t is the Hamming weight of the intermediate data involved in the computation plus a centered Gaussian noise ω_σ with standard deviation σ. Therefore $\mathrm{HW}_n(z)$ corresponds to the handling of the value z for the n-th encryption. We now define θ_0 and θ_1 as:

$$\theta_0 = (\mathrm{HW}_n(S(m_i \oplus k_i \oplus u) \oplus v) + \omega_\sigma)_{0 \le n \le N-1}$$

$$\theta_1 = (\mathrm{HW}_n(S(m_j \oplus k_j \oplus u) \oplus v) + \omega_\sigma)_{0 \le n \le N-1}$$

Let g_i (resp. g_j) denote a guess on k_i (resp. k_j). We compute the estimated values $w_{g_i,g_j} = \mathrm{HW}(S(m_i \oplus g_i) \oplus S(m_j \oplus g_j))$. Considering the N messages we obtain the series $W_{g_i,g_j} = (w^n_{g_i,g_j})_{0 \le n \le N-1}$. Using the combining function f_j, the right key bytes are obtained for the highest correlation value $\hat{\rho}(f_j(\theta_0, \theta_1), W_{g_i,g_j})$.

Then as in [99] we execute many times the attack with the different combining functions and calculate the success rate of each one. The Figure 6.9 shows two comparison graphs, one for $\sigma = 0.75$ and the other for $\sigma = 2$. Both graphs plot the success rates on 50 runs with respect to the number of curves used.

We emphasise that in this comparison the second-order attacks are shown in a very favorable light. Indeed the correlation model used here is exactly the one applied to simulate the curves. In practice an attacker would not have such good properties.

6.5 Countermeasures

The attacks presented in this chapter defeat first-order protected implementations. Therefore, an obvious countermeasure would be to apply second-order masking. To the best of our knowledge, the best solution should be the countermeasure presented by Rivain et al. [85]. It allows the implementation of proven d-order DPA resistant AES for any $d \geq 1$.

Another countermeasure against our first attack may simply consist in executing the SubBytes function in a random order. Even if this method is not theoretically perfect, it may be sufficient to practically resist to second-order attacks. Considering the second implementation, we think that its main weakness is the use of a same mask before and after each byte pseudo-inversion. If the result is masked with a different value then the collision-correlation attack is no longer feasible.

It is also necessary to consider that depending on the quality of the hardware countermeasures provided by the device, these attacks can become much more complicated in practice.

6.6 Masked Inverse

To simplify the notations we denote by a '$+$' the bitwise operation XOR. To compute the masked value $y'_i = {x_i}^{-1} \oplus u_i$ we define the functions $f_{x_{i,h}}$, $f_{x_{i,l}}$, f_{d_i} and $f_{d'_i}$ in $\mathrm{GF}(2^4)$ as follow :

$$
\begin{aligned}
((x_{i,h}+u_{i,h})X + (x_{i,l}+u_{i,l}))^{-1} &= (y_{i,h}+v_{i,h})X + (y_{i,l}+v_{i,l}) \\
y_{i,h}+v_{i,h} &= f_{x_{i,h}}((x_{i,h}+u_{i,h}), (d'_i+v_{d_i}), u_{i,h}, v_{i,h}, v_{d_i}) \\
&= x_{i,h} \times d_i^{-1} + v_{i,h} \\
y_{i,l}+v_{i,l} &= f_{x_{i,l}}((y_{i,h}+v_{i,h}), (x_{i,l}+u_{i,l}), (d'_i+v_{d_i}), x_{i,l}, v_{i,h}, v_{i,l}, v_{d_i}) \\
&= (x_{i,h}+x_{i,l}) \times d_i^{-1} + v_{i,l} \\
d_i + u_{d_i} &= f_{d_i}((x_{i,h}+u_{i,h}), (x_{i,l}+u_{i,l}), 14, u_{i,h}, u_{i,l}, u_{d_i}) \\
&= x_{i,h}^2 \times 14 + x_{i,h} \times x_{i,l} + x_{i,l}^2 + u_{d_i} \\
d'_i + v_{d_i} &= f_{d'_i}((d_i+u_{d_i}), u_{d_i}, v_{d_i}) \\
&= d_i^{-1} + v_{d_i}
\end{aligned}
$$

In [79] Oswald et al. consider that masks before and after inversion remain the same, then we have: $v_{i,h} = u_{i,h}$, $v_{i,l} = u_{i,l}$ and $v_{d_i} = v_{d'_i} = u_{i,h}$. The previous functions becomes:

$$
\begin{aligned}
f_{x_{i,h}} &= (x_{i,h} + u_{i,h}) \times (d'_i + u_{i,l}) + (d'_i + u_{i,l}) \times u_{i,h} \\
&+ (x_{i,h} + u_{i,h}) \times u_{i,l} + u_{i,h} + u_{i,h} \times u_{i,l} \\
\\
f_{x_{i,l}} &= (y_{i,h} + v_{i,h}) + (x_{i,l} + u_{i,l}) \times (d'_i + u_{i,h}) + (d'_i + u_{i,h}) \times u_{i,l} \\
&+ (x_{i,l} + u_{i,l}) \times u_{i,h} + u_{i,l} + u_{i,h} + u_{i,h} \times u_{i,l} \\
\\
f_{d_i} &= (x_{i,h} + u_{i,h})^2 \times 14 + (x_{i,h} + u_{i,h}) \times (x_{i,l} + u_{i,l}) + (x_{i,l} + u_{i,l})^2 \\
&+ (x_{i,h} + u_{i,h}) \times u_{i,l} + (x_{i,l} + u_{i,l}) \times u_{i,h} \\
&+ u_{i,h}^2 \times 14 + u_{i,l}^2 + u_{i,h} \times u_{i,l} + u_{i,h}
\end{aligned}
$$

6.7 Practical Results on Real Curves

The following figures show results of collision-correlation on first implementation. Every time a relation occurs and then $\texttt{Collision}(\Theta_0, \Theta_1) = \texttt{true}$, we can see a correlation peak as in Figure 6.5.Every time no such relation is present and then $\texttt{Collision}(\Theta_0, \Theta_1) = \texttt{false}$ we obtain curves as in Figure 6.6 where no correlation peak is present.

Chapter 7

ROSETTA

7.1 Introduction

Although crypto-systems are proven secure against theoretical cryptanalysis, they can be easily broken if straightforwardly implemented on embedded devices such as smart cards. Indeed, the so-called *Side-Channel Analysis* (SCA) takes advantage of physical interactions between the embedded device and its environment during the crypto-system execution to recover information on the corresponding secret key. Examples of such interactions are the device power consumption [74] or its electromagnetic radiation [43]. SCA can be mainly divided into two kinds: *Simple Side-Channel Analysis* (SSCA) and *Differential Side-Channel Analysis* (DSCA). The first kind aims at recovering information on the secret key by using only one execution of the algorithm whereas DSCA uses several executions of the algorithm and applies statistical analysis to the corresponding measurements to exhibit information on the secret key.

Amongst crypto-systems threatened by SCA, RSA [87] is on the front line since it is the most widely used public key crypto-system, especially in embedded environment. Therefore, many researchers have published efficient side-channel attacks and countermeasures specific to RSA over the last decade. Due to the constraints of the embedded environment, countermeasures must not only resist each and every SCA known so far but must also have the smallest impact in terms of performance and memory consumption. Nowadays, the most common countermeasure to prevent SSCA on RSA consists in using an exponentiation algorithm where the sequence of modular operations leaks no information on the secret exponent. Examples of such exponentiation are the square-and-multiply-always [34], the Montgomery ladder [55], the Joye ladder [56], the square-always [32] or the atomic multiply-always exponentiation [24]. The latter is generally favorite due to its very good performance compared to the other non-atomic methods. Regarding DSCA prevention, most common countermeasures consist in blinding the modulus and/or the message, and the exponent [61, 34]. Their effect is to randomize the intermediate values manipulated during the exponentiation as well as the sequence of squarings and multiplications. In this chapter we denote by *blinded exponentiation* an exponentiation using the atomic implementation presented in [24] where modulus, message and exponent are blinded.

Today blinded exponentiation remain resistant to most SCA techniques. Only the Big Mac attack presented by Walter [104] theoretically threatens this imple-

mentation, although no practical result has been ever published. Other attacks introduced later partially threaten this implementation. First, Amiel et al. [7] show how to exploit the average Hamming weight difference between squaring and multiplication operations to recover the secret exponent. Their technique is efficient when the modulus and the message are blinded. However it requires many exponentiation traces using a fixed exponent, so this attack can be thwarted by the randomization of the exponent. To circumvent the blinded exponentiation, they suggested to apply their attack on a single trace but did not try it in practice. Clavier et al. present in [30] the so-called *Horizontal Correlation Analysis*. They apply DSCA using the Pearson correlation coefficient [20] on a single exponentiation side-channel trace. The exponent randomization has no effect against this attack. Modulus and message blinding are efficient only if random masks are large enough (32 bits or more).

Other attacks on the RSA exponentiation are not mentioned in our study as they do not apply to the blinded exponentiation.

In this chapter we propose new attacks on the blinded exponentiation which make use of a single execution trace. We achieve this by introducing two new distinguishers — the Euclidean distance and the collision correlation applied to the long-integer multiplication — which allow to efficiently distinguish a squaring from a multiplication operation without the knowledge of the message or the modulus.

7.1.1 Roadmap

In section 7.2, we recall some basics on RSA implementations on embedded devices. In particular, we describe the attacks presented in [7, 30, 104] and we show that one of them can be extended using the collision-correlation technique. In section 7.3, we present the principle of the so-called *Rosetta* analysis using two different distinguishers. In section 7.4, we put into practice our attack and we demonstrate its efficiency using simulated side-channel traces of long-integer operations using a 32×32-bit multiplier. Moreover, we also compare our technique with previous attacks and show that it is more efficient especially on noisy measurements. We discuss in section 7.5 possible methods to counteract Rosetta analysis.

7.2 Background

In this section, after presenting some generalities on RSA implementation in the context of embedded environment, we present three of the most efficient side-channel attacks published so far on RSA: the *Big Mac attack* published by Walter at CHES 2001 [104], the one published by Amiel et al. at SAC 2008 [7] and the *Horizontal Correlation Analysis* published at ICICS 2010 by Clavier et al. [30]. Also, we explain how the latter can be extended using a collision-correlation technique.

7.2.1 RSA Implementation

The standard way of computing an RSA signature S of a message m consists of a modular exponentiation with the private exponent: $S = m^d \bmod N$. The corre-

sponding signature is verified by comparing the message m with the signature S raised to the power of the public exponent: $m \stackrel{?}{=} S^e \bmod N$.

In order to improve its efficiency, the signature is often computed using the Chinese Remainder Theorem (CRT). Let us denote by d_p (resp. d_q) the residue $d \bmod p-1$ (resp. $d \bmod q-1$). To compute the signature, the message is raised to the power of d_p modulo p then to the power of d_q modulo q. The corresponding results S_p and S_q are then combined using Garner's formula [44] to obtain the signature: $S = S_q + q(q^{-1}(S_p - S_q) \bmod p)$.

If used exactly as described above, RSA is subject to multiple attacks from a theoretical point of view. Indeed, it is possible under some assumptions to recover some information on the plaintext from the ciphertext or to forge fake signatures. To ensure its security, RSA must be used according to a protocol which mainly consists in formatting the message. Examples of such protocols are the encryption protocol OAEP and the signature protocol PSS, both of them being proven secure and included in the standard PKCS #1 V2.1 [54]. Note that, as they do not require the knowledge of the exponentiated value, the new attacks described in this contribution also apply when either OAEP or PSS scheme is used.

From a practical point of view, the RSA exponentiation is also subject to many attacks if straightforwardly implemented. For instance, SSCA, DSCA or collision analysis can be used to recover the RSA private key. SSCA aims at distinguishing a difference of behavior when an exponent bit is a 0 or a 1.

DSCA allows a deeper analysis than SSCA by exploiting the dependency which exists between side-channel measurements and manipulated data values [9]. To this end, thousands of measurements are generally combined using a statistical distinguisher to recover the secret exponent value. Nowadays, the most widespread distinguisher is the Pearson linear correlation coefficient [20].

Finally, collision analysis aims at identifying when a value is manipulated twice during the execution of an algorithm.

Algorithm 7.1 presents the classical atomic exponentiation which is one of the fastest exponentiation algorithms protected against the SPA.

Alg. 7.1 Atomic Multiply-Always Exponentiation

Input: $x, n \in \mathbb{N}$, $d = (d_{v-1}d_{v-2}\ldots d_0)_2$
Output: $x^d \bmod n$

1: $R_0 \leftarrow 1$
2: $R_1 \leftarrow x$
3: $i \leftarrow v-1$
4: $k \leftarrow 0$
5: **while** $i \geq 0$ **do**
6: $\quad R_0 \leftarrow R_0 \times R_k \bmod n$
7: $\quad k \leftarrow k \oplus d_i$ [$\oplus$ stands for bitwise X-or]
8: $\quad i \leftarrow i - \neg k$ [$\neg$ stands for bitwise negation]
9: **end while**
10: **return** R_0

When correctly implemented, Alg. 7.1 defeats SSCA since squarings cannot be distinguished from other multiplications on a side-channel trace, as depicted by Figure 7.1.

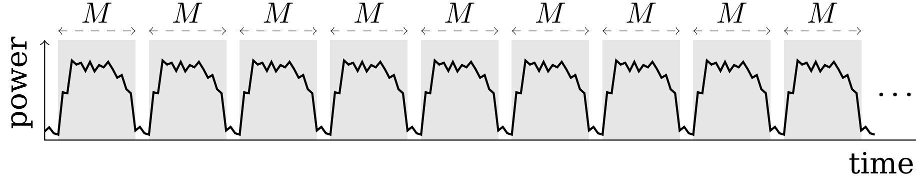


Figure 7.1: Atomic multiply-always side-channel leakage

To prevent the implementation of RSA exponentiation from DSCA, the two main countermeasures published so far are based on message and exponent blinding [34, 61]. Instead of computing straightforwardly $S = m^d \bmod n$, one rather computes $\tilde{S} = (m + k_0 \cdot n)^{d+k_1 \cdot \varphi(n)} \bmod 2^\lambda \cdot n$ where φ denotes the Euler's totient and k_0 and k_1 are two λ-bit random values, then finally reduce $\tilde{S}$ modulo N to obtain S. Using such a blinding scheme with a large enough λ (32 bits are generally considered as a good compromise between security and cost overhead), the relationship between the side-channel leakages occurring during an exponentiation and the original message and exponent values is hidden to an adversary, therefore circumventing DSCA.

As the modular exponentiation consists of a series of modular multiplications, it relies on the efficiency of the modular multiplication. Many methods have been published so far to improve the efficiency of this crucial operation. Amongst these methods, the most popular are the Montgomery, Knuth, Barrett, Sedlack or Quisquater modular multiplications [76, 35]. Most of them have in common that the long-integer multiplication is internally computed by repeatedly calling a smaller multiplier operating on t-bit words. A classic example is given in Alg. 7.2 which performs the schoolbook long-integer multiplication using a t-bit internal multiplier giving a $2t$-bit result. The decomposition of an integer x in t-bit words is given by $x = (x_{\ell-1}x_{\ell-2}\dots x_0)_b$ with $b = 2^t$ and $\ell = \lfloor \log_b(x) \rfloor + 1$.

Alg. 7.2 Schoolbook Long-Integer Multiplication

Input: $x = (x_{\ell-1}x_{\ell-2}\dots x_0)_b, y = (y_{\ell-1}y_{\ell-2}\dots y_0)_b$
Output: $x \times y$

```
1:  for i = 0 to 2ℓ − 1 do
2:      z_i ← 0
3:  end for
4:  for i = 0 to ℓ − 1 do
5:      R_0 ← 0
6:      R_1 ← x_i
7:      for j = 0 to ℓ − 1 do
8:          R_2 ← y_j
9:          R_3 ← z_{i+j}
10:         (R_5 R_4)_b ← R_3 + R_2 × R_1 + R_0
11:         z_{i+j} ← R_4
12:         R_0 ← R_5
13:     end for
14:     z_{i+ℓ} ← R_5
15: end for
16: return z
```

In the rest of this section we recall some previously published attacks on atomic

exponentiations which inspired our new technique detailed in section 7.3.

7.2.2 Attacks Background

Distinguishing Squarings from Multiplications in Atomic Exponentiation

In [7] Amiel et al. present a specific DSCA aimed at distinguishing squaring from other multiplications in the atomic exponentiation. They observe that the average Hamming weight of the output of a multiplication $x \times y$ has a different distribution whether:

- the operation is a squaring performed using the multiplication routine, i.e. $x = y$, with x uniformly distributed in $[0, 2^{\ell t} - 1]$;
- or the operation is an actual multiplication, with x and y independent and uniformly distributed in $[0, 2^{\ell t} - 1]$.

Thus, considering a device with a single long-integer multiplication routine used to perform either $x \times x$ or $x \times y$, a set of N side-channel traces computing multiplications with random operands can be distinguished from a set of N traces computing squarings, provided that N is sufficiently large to make the two distribution averages separable. This attack can thus target an atomic exponentiation such as Alg. 7.1 even in the case of message and modulus blinding. Regarding the exponent blinding, authors suggest that their attack should be extended to success on a single trace but do not give evidence of its feasibility. We thus study this point in the following of the chapter.

Horizontal Correlation Analysis

Correlation analysis on a single atomic exponentiation side-channel trace has been published in [30] where the message is known to the attacker but the exponent is blinded. This attack called *horizontal* correlation analysis requires only one exponentiation trace to recover the full RSA private exponent.

Instead of considering the whole k-th long-integer multiplication side-channel trace T^k as a block, the authors consider each inner side-channel trace segment corresponding to a single-precision multiplication on t-bit words. For instance, if the long-integer multiplication is performed using Alg. 7.2 on a device provided with a t-bit multiplier, then the trace T^k of the k-th long-integer multiplication $x \times y$ can be split into ℓ^2 trace segments $T^k_{i,j}$, $0 \leq i, j < \ell$, each of them representing a single-precision multiplication $x_i \times y_j$. More precisely, for each word y_j of the multiplicand y, the attacker obtains ℓ trace segments $T^k_{i,j}$, $0 \leq i, j < \ell$, corresponding to a multiplication by y_j. The slicing of T^k into trace segments $T^k_{i,j}$ is illustrated on Figure 7.2.

In the horizontal correlation analysis the attacker is able to identify whether the k-th long-integer operation T^k is a squaring or a multiplication by computing the correlation factor between the series of Hamming weights of each t-bit word m_j of the message m and the series of corresponding sets of ℓ trace segments $T^k_{i,j}$, $0 \leq i, j < \ell$. This correlation factor is expected to be much smaller when the long-integer operation is a squaring (i.e. $R_0 \leftarrow R_0 \times R_0$ in Alg. 7.1) than when it is a multiplication by m (i.e. $R_0 \leftarrow R_0 \times R_1$). The correlation factor can be computed

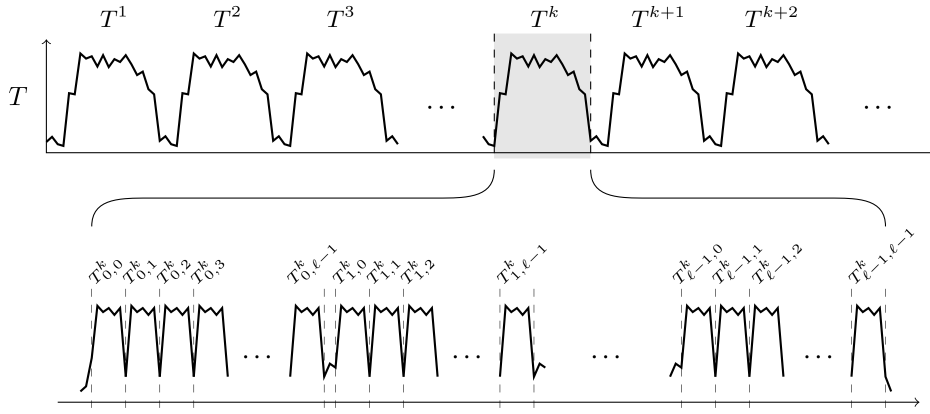


Figure 7.2: Horizontal side-channel analysis on exponentiation

by using the Pearson correlation coefficient $\rho(H, T^k)$ where $H = (H_0, \ldots, H_{\ell-1})$, with $H_j = (\mathrm{HW}(m_j), \ldots, \mathrm{HW}(m_j))$, $\mathrm{HW}(m_j)$ standing for the Hamming weight of m_j and $T^k = (T^k_0, \ldots, T^k_{\ell-1})$ with $T^k_j = (T^k_{0,j}, \ldots, T^k_{\ell-1,j})$.

Big Mac Attack

Walter's attack needs, as our technique, a single exponentiation side-channel trace to recover the secret exponent. For each long-integer multiplication, the Big Mac attack detects if the operation performed is either $R_0 \times R_0$ or $R_0 \times m$. The multiplications $x_i \times y_j$ — and corresponding trace segments $T^k_{i,j}$ — can be easily identified on the side-channel trace from their specific pattern which is repeated ℓ^2 times in the long-integer multiplication loop. A template side-channel trace is computed (either from the precomputations or from the first squaring operation) to characterize the manipulation of the message during the long-integer multiplication. The Euclidean distance between the template trace and each long-integer multiplication trace T^k is then computed. If it exceeds a threshold then the attack concludes that the operation is a squaring, or a multiplication by m otherwise.

Walter uses the Euclidean distance but we noticed that other distinguisher could be used. In the following section, we extend the Big Mac attack using a collision-correlation technique.

7.2.3 Big Mac Extension using Collision Correlation

A specific approach for SCA uses information leakages to detect collisions between data manipulated in algorithms. A side-channel collision attacks against a block cipher was first proposed by Schramm et al. in 2003 [93]. More recently Moradi et al. [78] proposed to use a correlation distinguisher to detect collisions in AES. The main advantage of this approach is that it is not necessary to define a leakage model as points of traces are directly correlated with other points of traces. Later, Clavier et al. [29] presented two collision-correlation techniques defeating different first order protected AES implementations. The same year, Witteman et al. [106] applied collision correlation to public key implementation. They describe an efficient attack on RSA using square-and-multiply-always exponentiation and message blinding. All these techniques require many side-channel

execution traces. In this section, we extend Walter's Big Mac attack using the collision correlation as distinguisher instead of the Euclidean distance.

We consider a blinded exponentiation and use the fact that the second and third modular operations in an atomic exponentiation are respectively $1 * \tilde{m}$ and $\tilde{m} * \tilde{m}$, where $\tilde{m}$ is the blinded message. The trace of the second long-integer multiplication yields ℓ multiplication segments for each word $\tilde{m}_j$ of the blinded message. Considering the k-th long-integer multiplication, $k > 3$, we compute the correlation factor between the series of ℓ trace segments T_j^2 — each one being composed of the ℓ trace segments $T_{i,j}^2$ involved in the multiplication by $\tilde{m}_j$ — and the series of ℓ trace segments T_j^k. Since the blinded value of the message does not change during the exponentiation, a high correlation occurs if the k-th long-integer operation is a multiplication, and a low correlation otherwise. Once the sequence of squarings and multiplications is found, the blinded exponent value is straightforwardly recovered. Notice that recovering the blinded value of the secret exponent is not an issue as it can be used to forge signature as well as its non-blinded value.

This attack also works if we use the trace segments T_j^3 of the third long-integer operation instead of the trace segments T_j^2. One can also combine the information provided by the second and third long-integer operations to improve the attack.

Remark

As the original Big Mac, this attack also applies to the CRT RSA exponentiation since no information is required on either the message or the modulus. This is of the utmost importance since, to the best of our knowledge, this is the first practical attack on a CRT RSA fully blinded (message, modulus and exponent) atomic exponentiation.

7.3 ROSETTA: Recovery Of Secret Exponent by Triangular Trace Analysis

7.3.1 Attack Principle

The long-integer multiplication $\mathrm{LIM}(x, y)$ in base $b = 2^t$ is given by the classical schoolbook formula:

$$x \times y = \sum_{i=0}^{\ell-1} \sum_{j=0}^{\ell-1} x_i y_j b^{i+j}$$

and illustrated, with for instance $\ell = 4$ by the following matrix M:

$$M = \begin{pmatrix} x_0y_0 & x_0y_1 & x_0y_2 & x_0y_3 \\ x_1y_0 & x_1y_1 & x_1y_2 & x_1y_3 \\ x_2y_0 & x_2y_1 & x_2y_2 & x_2y_3 \\ x_3y_0 & x_3y_1 & x_3y_2 & x_3y_3 \end{pmatrix}$$

In the case of a squaring, then $x = y$ and the inner multiplications become:

$$S = \begin{pmatrix} x_0x_0 & x_0x_1 & x_0x_2 & x_0x_3 \\ x_1x_0 & x_1x_1 & x_1x_2 & x_1x_3 \\ x_2x_0 & x_2x_1 & x_2x_2 & x_2x_3 \\ x_3x_0 & x_3x_1 & x_3x_2 & x_3x_3 \end{pmatrix}$$

We consider four observations to design our new attacks, assuming a large enough multiplier size $t \geq 16$:

(Ω_0) $\mathrm{LIM}(x, y)$ s.t. $x = y \Rightarrow \mathrm{Prob}(x_i \times y_i$ are squaring operations$) = 1 \ \ \forall i$

(Ω_1) $\mathrm{LIM}(x, y)$ s.t. $x \neq y \Rightarrow \mathrm{Prob}(x_i \times y_i$ are squaring operations$) \approx 0 \ \ \forall i$

(Ω_2) $\mathrm{LIM}(x, y)$ s.t. $x = y \Rightarrow \mathrm{Prob}(x_i \times y_j = x_j \times y_i) = 1 \ \ \forall i \neq j$.

(Ω_3) $\mathrm{LIM}(x, y)$ s.t. $x \neq y \Rightarrow \mathrm{Prob}(x_i \times y_j = x_j \times y_i) \approx 0 \ \ \forall i \neq j$.

From observations (Ω_0) and (Ω_1) one can apply the attack presented in [7] on a single trace as suggested by the authors. The main drawback is that only ℓ such operations are performed during a LIM which represents a small number of trace segments. It is likely to make the attack inefficient for small modulus lengths (with respect to the multiplier size t).

From observations (Ω_2) and (Ω_3) we notice that collisions between $x_i \times y_j$ and $x_j \times y_i$ for $i \neq j$ can be used to identify squarings from other multiplications. Moreover, $\mathrm{LIM}(x, y)$ provides $\ell^2 - \ell$ operations $x_i \times y_j$, $i \neq j$, thus $(\ell^2 - \ell)/2$ couples of potential collisions. This represents a fairly large number of trace segments. The principle of our new attack consists in detecting those internal collisions in a single long-integer operation to determine whether it is a squaring or not. Visually, we split the matrix M into an upper-right and a lower-left triangles of terms, thus we call this technique a *triangle trace analysis*.

We present in the following two techniques to identify these collisions on a single long-integer multiplication trace. The first analysis uses the Euclidean distance distinguisher and the second one relies on a collision-correlation technique.

7.3.2 Euclidean Distance Distinguisher

We use as distinguisher the Euclidean distance between two sets of points on a trace as Walter [104] in the Big Mac analysis. In order to exploit properties (Ω_2) and (Ω_3) we proceed as follows. For each $\mathrm{LIM}(x, y)$ operation we compute the following differential side-channel trace:

$$T_{\mathrm{ED}} = \frac{2}{\ell^2 - \ell} \sum_{0 \leq i < j < \ell} \sqrt{(T_{i,j} - T_{j,i})^2}$$

If the operation performed is a squaring then the single-precision multiplications $x_i \times y_j$ and $x_j \times y_i$ store the same value in the result register (or in the memory) at the end of the operation. The side-channel leakage of the result storage of both operations should thus be similar. On the other hand, if $x \neq y$, products differ and the side-channel leakage should present less similarities. Assuming a side-channel leakage function linear in the Hamming weight of the data manipulated, a squaring should result in $E(T_{\mathrm{ED}}) \approx 0$, whereas we should expect a significantly higher value (about $t/2$ for each of the product halves) in the case of a multiplication.

7.3.3 Collision-Correlation Distinguisher

We define the two following series of trace segments, where the ordering of couples (i,j) is the same for the two series:

$$\Theta_0 = \{T_{i,j} \text{ s.t. } 0 \le i < j \le \ell - 1\}$$
$$\Theta_1 = \{T_{j,i} \text{ s.t. } 0 \le i < j \le \ell - 1\}$$

Each set includes $N = (\ell^2 - \ell)/2$ trace segments of base b multiplications.

In order to determine the operation performed by the LIM we compute the Pearson correlation factor between the two series Θ_0 and Θ_1 as described in [29]:

$$\hat{\rho}_{\Theta_0,\Theta_1}(t) = \frac{\mathrm{Cov}(\Theta_0(t), \Theta_1(t))}{\sigma_{\Theta_0(t)}\sigma_{\Theta_1(t)}} = \frac{N\sum(T_{i,j}(t)T_{j,i}(t)) - \sum T_{i,j}(t)\sum T_{j,i}(t)}{\sqrt{N\sum(T_{i,j}(t))^2 - (\sum T_{i,j}(t))^2}\sqrt{N\sum(T_{j,i}(t))^2 - (\sum T_{j,i}(t))^2}}$$

where summations are taken over all couples $0 \le i < j \le \ell - 1$.

In case of a squaring operation, a much higher correlation value $\hat{\rho}_{\Theta_0,\Theta_1}$ is expected than in case of a multiplication. Computing this correlation value for each LIM operation allows to determine its nature and to recover the sequence of exponent bits.

Remark

Contrary to differential analysis on symmetric ciphers, each exponent bit requires to distinguish one hypothesis out of only two, instead of for instance 256 considering a differential attack on AES. Thus fixing a decision threshold is easier when dealing with the exponentiation. This has already been observed when applying DPA or CPA on RSA [9, 72] compared to DES or AES.

7.4 Comparison of the Different Attacks

In order to validate these two techniques, we generated simulated side-channel traces for a classical 32×32-bit multiplier. As generally considered in the literature, we assume a side-channel leakage model linear in the Hamming weight of the manipulated data — here x_i, y_j, and $x_i \times y_j$ — and add a white Gaussian noise of mean $\mu = 0$ and standard deviation σ. We build simulated side-channel traces based on the Hamming weight of the data manipulated in the multiplication operation such that each processed single-precision multiplication generates four leakage points $\mathrm{HW}(x_i)$, $\mathrm{HW}(y_j)$, $\mathrm{HW}(x_i \times y_j \bmod b)$, and $\mathrm{HW}(x_i \times y_j \div b)$, where $\div$ stands for the Euclidean quotient.

Besides validating our two Rosetta variants — the Euclidean distance distinguisher (Rosetta ED) and the collision-correlation one (Rosetta CoCo) — we compare Rosetta with other techniques discussed previously, namely the classical Big

Mac, the Big Mac using collision correlation (Big Mac CoCo), and the single trace variant of the Amiel et al. attack presented at SAC 2008.

We proceed in the following way: we randomly select two ℓ-bit integers x and y. Then we generate the side-channel traces of the multiplication $\mathrm{LIM}(x, y)$ and of the squaring $\mathrm{LIM}(x, x)$.

Each different attack is eventually applied and we keep trace of their success or failure to distinguish the squaring from the multiplication. Finally, we estimate the success rate of each technique by running $1\,000$ such experiments. These tests are performed for three different noise standard deviation values[1]: from no noise ($\sigma = 0$) to a strong one ($\sigma = 7$).

Characterisation and Threshold

A threshold for the attack must be selected for each technique to determine whether the targeted operation is a multiplication or a squaring. Using simulated side-channel traces, it was possible to determine the best threshold value for each technique. Without any knowledge on the component, it is more difficult to fix those threshold values. The attacks could be processed with guess on these thresholds, for instance selecting 0.5 for the collision correlation, but it could not reach optimal efficiency or fail. It is then preferable to determine the best threshold values through a characterization phase of the multiplier, either with an access to an open sample or using the public exponentiation calculation as suggested in [9].

Results

We obtain the success rates given in tables Table 7.1 ($\sigma = 0$), Table 7.2 ($\sigma = 2$) and Table 7.3 ($\sigma = 7$) for different key lengths ranging from 512 bits to 2048 bits. Figure 7.3 and Figure 7.4 present a graphic comparison of these results for $\sigma = 0$ and $\sigma = 7$.

Technique	512 bits	768 bits	1024 bits	2048 bits
Big Mac [104]	0.986	0.990	0.993	0.995
SAC 2008 [7]	0.533	0.618	0.734	0.897
Big Mac CoCo (§subsection 7.2.3)	0.999	1.00	1.00	1.00
Rosetta ED (§subsection 7.3.2)	1.00	1.00	1.00	1.00
Rosetta CoCo (§subsection 7.3.3)	1.00	1.00	1.00	1.00

Table 7.1: Success rate with a null noise, $\sigma = 0$

Results Interpretation

We observe that with no noise (cf. Table Table 7.1) all techniques are efficient when applied to large modulus bit lengths (1536 bits or more). For smaller modulus lengths, the SAC 2008 technique is inefficient (probability of success close to 0.5) as expected since the number of useful operations in that case is too small.

[1] Regarding the standard deviation of the noise, a unit corresponds to the side-channel difference related to a one bit difference in the Hamming weight.

Technique	512 bits	768 bits	1024 bits	2048 bits
Big Mac [104]	0.767	0.775	0.807	0.818
SAC 2008 [7]	0.546	0.629	0.717	0.855
Big Mac CoCo (§subsection 7.2.3)	0.981	0.998	0.999	1.00
Rosetta ED (§subsection 7.3.2)	1.00	1.00	1.00	1.00
Rosetta CoCo (§subsection 7.3.3)	1.00	1.00	1.00	1.00

Table 7.2: Success rate with a moderate noise, $\sigma = 2$

Technique	512 bits	768 bits	1024 bits	2048 bits
Big Mac [104]	0.557	0.577	0.621	0.632
SAC 2008 [7]	0.551	0.577	0.623	0.702
Big Mac CoCo (§subsection 7.2.3)	0.737	0.855	0.909	0.981
Rosetta ED (§subsection 7.3.2)	0.711	0.821	0.878	0.992
Rosetta CoCo (§subsection 7.3.3)	0.685	0.816	0.906	0.997

Table 7.3: Success rate with a strong noise, $\sigma = 7$

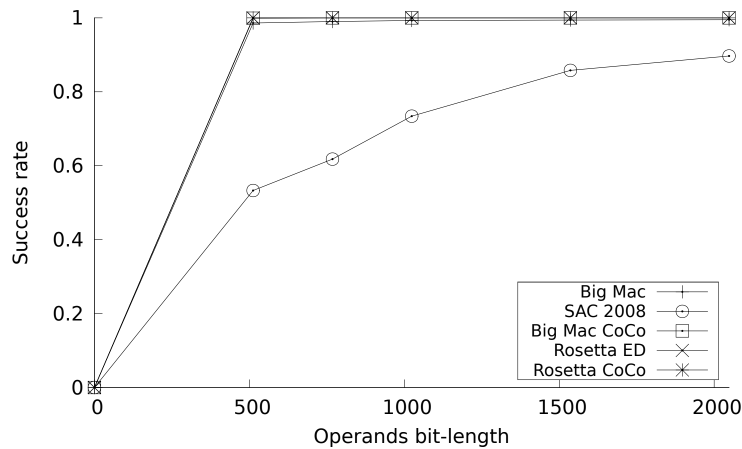


Figure 7.3: Success rate of the different attacks with no noise.

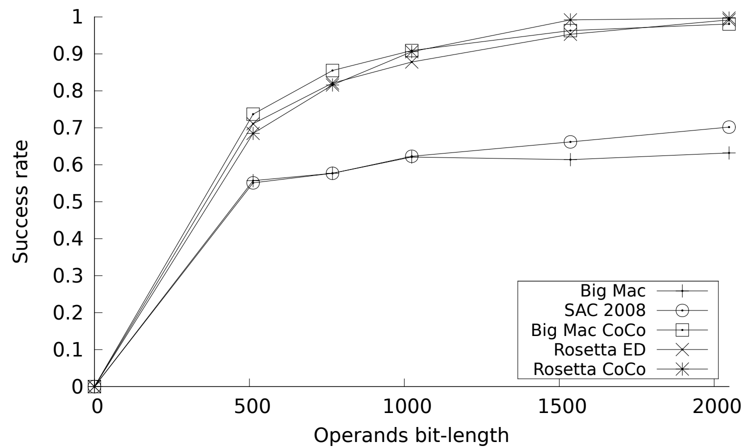


Figure 7.4: Success rate of the different attacks with a strong noise, $\sigma = 7$.

In case of a noisy component, we observe that the original Big Mac and the attack from SAC 2008 are not efficient, their probability of success is about 0.5–0.7. Big Mac analysis using collision correlation, and both Rosetta techniques start to be efficient from 1024-bit operands and are very efficient for 1536-bit and 2048-bit operands.

Our study demonstrates that these three last techniques are the most efficient ones and represent a more serious threat for blinded exponentiation than the original Big Mac.

From Partial to Full Exponent Recovery

Depending on the component, on the leakage and noise level of the chip, we observe that the success rate of the attack varies and may reveal too few information to recover the whole exponent value. In the case where uncertainty remains on some exponent bits, the attack from Schindler and Itoh [89] may help to reveal them. If necessary, Rosetta analysis can thus be advantageously combined with this technique to completely recover the exponent.

7.5 Countermeasures

As for the other attacks considered in this chapter, both Rosetta techniques we introduced present the following interesting properties: (i) they make use of a single side-channel trace and, (ii) they do not require the knowledge of the message nor of the modulus. As a consequence they are applicable even when the classical set of blinding countermeasures (message, modulus, exponent) is implemented and whatever the size of the random values used.

A first idea to prevent these attacks is to improve the message blinding by randomizing it before each long-integer multiplication, for instance by adding the modulus n or a multiple thereof to the message. At this point, it is worth noticing a specific difference between both Rosetta and other attacks. Rosetta can distinguish a squaring from a multiplication without using any template or previous leakage. This is not the case with the other techniques — except for the single trace variant of the SAC 2008 attack which we demonstrate not to be efficient in the previous section. The consequence is that Rosetta is still applicable even when this improved blinding is implemented.

We recall hereafter three existing countermeasures that we believe to withstand all the techniques presented in this contribution.

Shuffled Long-Integer Multiplication In [30], a long integer multiplication algorithm with internal single-precision multiplications randomly permuted is presented. More details are given in [102, Sec. 2.7]. This countermeasure makes Rosetta analysis virtually infeasible as indices i, j of multiplication $x_i \times y_j$ are not known anymore.

Always True Multiplication This solution consists in ensuring that multiplication operands are always different (or different with high probability). To achieve this objective, before each multiplication $\mathrm{LIM}(x, y)$, both operands x and y are randomized by $x^\star = x + r_1.n$ and $y^\star = y + r_2.n$. If $r_1 \neq r_2$, two equal operands x and y are traded for $x^\star$ and $y^\star$ with $x^\star \neq y^\star$ and the operation $\mathrm{LIM}(x^\star, y^\star)$ is not a squaring.

Square-Always algorithm The square-always algorithm presented in [32] processes any multiplication using two squarings. As for the solution of using multiplications of different terms only, Rosetta does not apply. Regular atomic square always algorithms can be used to prevent SSCA. Exponent blinding countermeasure must be associated with this solution.

Chapter 8

Passive and Active Combined Analysis

8.1 Introduction

Countermeasures against active and passive attacks are generally studied and proposed separately, and protecting a product from both techniques usually consists in superposing both kinds of countermeasures. In [6] the authors showed that simply superposing countermeasures is not sufficient as they succeeded in breaking an RSA implementation that used state of the art countermeasures against side-channel attacks and fault attacks. Their *Passive and Active Combined Attack*, Passive and Active Combined Attack (PACA), on an RSA implementation demonstrates that naively adding countermeasures together is not sufficient and that implementing these protections must be done carefully be means e.g. of the infective methodology.

In this chapter we present another passive and active combined attack on a state of the art SCA protected AES [40]. We combine a particular fault attack technique named *Collision Fault Analysis* (Correlation Fault Analysis (CFA)) that was introduced by Blömer and Seifert [16] in 2003, with the classic *Correlation side-channel analysis* (Correlation Side-Channel Analysis (CSCA)) introduced by Brier et al. [20] in 2004.

This chapter is organized as follows. The section 8.2 gives an overview of active and passive attacks with a focus on the collision fault analysis and the correlation side-channel analysis. We present in section 8.3 the AES state of the art implementation chosen for this study and explain why this implementation is resistant to the previously published CFA. In section 8.4 we introduce our combined attack and explain how, with the same fault model as the CFA, it can recover the secret key on our AES implementation. We discuss the countermeasures in section 8.5 and describe a safe-error variant of our attack which defeats these countermeasures in section 8.6.

8.2 Side Channel and Fault Analysis Background

Passive attacks consist in observing side-channel information, such as the power consumed by the chip while performing sensitive operations during a cryptographic computation. Active attacks consist in perturbing the device when it is processing sensitive data or calculations. Both techniques may result in the recovery of the secrets.

8.2.1 Side Channel Analysis

Power and electromagnetic analysis rely on the following physical property: a microprocessor is physically made of thousands of logical gates which switch differently depending on the operations being executed and the data being manipulated. The power consumption and the electromagnetic radiation, which depend on these gate switching may leak information on the executed instructions and the manipulated data. Consequently, by monitoring the power consumption or radiation of a device performing cryptographic operations, an observer can infer information on the implementation of the program executed and on the secret data involved. Basic SSCA consists in analyzing the secret key manipulations on a single power curve, for instance during the key scheduling operations in the DES or AES rounds. DPA is a more powerful method which consists in validating an hypothesis on some key bits, by a statistical treatment on many execution power curves of the targeted embedded implementation. The complexity is then reduced to very few calculations compared to a classical cryptanalysis or to a brute force attack on the key. For instance a classical DPA on a unprotected software implementation of AES on a standard microcontroller typically requires between 500 to few thousands power curves. Some improvements of DPA have been published in the previous years, among which the *Correlation Power Analysis* CPA technique requires far fewer curves for recovering the key than the original DPA. Recovering the key by CPA on unprotected AES implementations may require between only 50 and few hundred curves. Other recent methods have been published that also improve the side-channel attacks [45, 98, 82]. Most common countermeasures against power analysis, and particularly DPA and CPA, consist in using random values for masking the operations. In this case even if an attacker makes guesses on some secret key bits, he can not predict any intermediate value as another unknown variable, the random mask, is part of any intermediate data during the computation. However in this case a more complex but realistic attack, named *High Order Differential Power Analysis* (HODPA), presented by Messerges [71], is still applicable if the mask values are identical on different bytes, and/or if some different instants on a same power curve can be used to eliminate the random mask effect.

We now briefly present the CPA technique which is the passive component of our combined attack.

Correlation Power Analysis

The power consumption of the device is supposed to vary linearly with $\mathrm{HW}(D \oplus R)$, the Hamming distance between the data manipulated D and a *reference state* R. The power consumption model W is then defined as $W = \mu \cdot \mathrm{HW}(D \oplus R) + \nu$,

where ν captures both the experimental noise and the non modelized part of the power consumption. The linear correlation factor: $\rho_{C,H} = \frac{\text{cov}(C,H)}{\sigma_C \sigma_H}$ is then used to correlate the power curves C with this value $\text{HW}(D \oplus R)$. Knowing that the maximum correlation factor is obtained for the correct guess of secret key bits, an attacker can try all possible secret bits values and select the value corresponding to the highest correlation.

8.2.2 Fault Analysis

Fault effects and perturbations on electronic devices were first observed in the 1970's in the aerospace industry. Later the Differential Fault Analysis, DFA for attacking embedded symmetric cryptosystems was introduced by Biham and Shamir [15] in 1997. In this paper the authors explain how to recover the secret key by using between 50 and 200 ciphertexts. For years this threat was considered as only theoretical until the first practical results of light attacks were presented (on an RSA implementation) by Anderson and Skorobogatov [95]. The DFA has subsequently been studied and applied on DES in [47] where Giraud and Thiebeauld recover the key by means of only 2 faulty ciphertexts. In the case of the AES many attacks have been proposed [81, 37, 48] that allow the secret key to be recovered by using as few as 2 faulty ciphertexts.

We now present the Collision Fault Analysis technique which is the active component used in our combined attack.

Collision Fault Analysis

In [16] Blömer and Seifert first published a CFA on the first XOR of the AES. They assume a fault model where the attacker has the ability to force to zero any chosen bit of the result of this XOR operation. Then they compare a correct and a faulted AES execution for the same message. If both ciphertexts are equal the original value of the result bit is 0, otherwise it is 1. Knowing the message and scanning the different key bits, the whole 128-bit AES key is retrieved with 128 faulty executions. An interesting property is that the classical countermeasures which consist in checking the computation, for instance by executing the AES twice and comparing the results, do not prevent this attack. Indeed whether the card detects the fault or not will provide the attacker with the same information as whether or not the fault corrupted the ciphertext.

Later Hemme [52] presented the first CFA on the DES. His attack consists in introducing one bit errors in the first rounds of the algorithm. Then by computing chosen message encryption with the card (without injecting faults) the attacker obtains collisions that he can exploit to recover information on the secret key. With enough collisions he can recover the whole secret key. In this case, verifying the whole DES computation is an efficient countermeasure.

Another CFA analysis on the AES first XOR computation can be done when the fault effect resets a whole byte (or many bytes) instead of a bit. In this attack an induced fault resets the result of a XOR between one message byte M_j and one key byte K_j – with the other key addition byte results not being affected. The attacker stores the faulty ciphertext C' and asks the card to encrypt the 256 messages M with M_j taking all possible byte values. One of these 256 ciphertexts will be equal

to C'. This collision is produced for M_j verifying $M_j \oplus K_j = 0$, which indicates that $K_j = M_j$.

Amiel et al. [4] adapted this CFA to an AES protected from first order DPA by random masking. In this implementation the same random byte r_1 is used for masking all 16 message bytes and the same random byte r_2 is applied on all 16 key bytes. In that case a single random byte $r = r_1 \oplus r_2$ is applied on all the bytes of intermediate values throughout the computation. The authors succeeded in faulting 2 to 16 bytes of the result of the first XOR. Then by searching collisions they obtained relations between known input bytes and key bytes masked. Exploiting these relations allows them to recover the secret key. This attack needs the precomputation with the card of 2^{23} non faulty ciphertexts and in practice 112 faulty ciphertexts were used. Note that this attack is applicable only if the same random mask r is applied on all of the 16 bytes of the intermediate values. The targeted implementation was not protected against high order differential analysis and in particular against a second order analysis. State of the art implementations are thus not vulnerable to this CFA.

8.3 Targeted AES Implementation

We present here the implementation targeted by our attack. We have chosen a state of the art side channel resistant AES implementation. To prevent DPA and CPA attacks, a 16-byte random mask is used to mask the input message (and another one to mask the key). This random mask is composed of 16 different random bytes that can change at each round. This targeted implementation is designed to resist to the HODPA attack presented in [1] and [66].

To realize such an implementation it is not possible to use a 256-byte substitution table as randomizing this substitution table for each random byte $r_0, \ldots, r_{15}$ would necessitate precomputing and storing 16×256-byte substitution tables, one for each byte r_i. Moreover these tables would need to be recomputed at each round for changing the mask between each round. The chosen implementation is the one presented by Oswald et al. in [79], the inversion is here computed masked in $GF(2^4)$. In this case all the 16 bytes of the message and the intermediate calculations are masked with different random bytes. This implementation is described in Figure 8.1 in Appendix.

Note that, as previously stated, the CFA presented by Amiel et al. is not applicable on our targeted implementation.

We have carried out two implementations of a secure AES on an 8-bit microprocessor with different security levels. The first one is resistant to DPA attacks and takes 20000 cycles (2 ms at 10 MHz). Data are masked by the same byte which requires precomputing only one substitution table for one AES execution. This implementation is not resistant to HODPA attacks and is also vulnerable to Amiel et al. CFA. We also carried out the implementation described above which uses inversion in $GF(2^4)$. All data are masked by different bytes which change between each round. This implementation is resistant to HODPA attacks and takes 51000 cycles (5.1 ms at 10 MHz). We will refer to both these implementations as AES_{DPA} and $\text{AES}_{\text{HODPA}}$ respectively. The performance and memory footprint figures for both implementations are presented in Table Table 8.1. We introduce the AES_{DPA}

Table 8.1: Performance (cyles) and memory costs (bytes) for AES_{DPA} and $\text{AES}_{\text{HODPA}}$ implementations

	Cycles	ROM	RAM
AES_{DPA}	20 000	4 000	$256 + 65$
$\text{AES}_{\text{HODPA}}$	51 000	5 500	75

implementation here only for comparison purposes to illustrate the cost implied for protecting an AES from HODPA. Only the $\text{AES}_{\text{HODPA}}$ implementation will be referred to in our attacks.

8.4 Passive and Active Combined Attack on Masked AES

In this section we present an analysis that can be carried out on our $\text{AES}_{\text{HODPA}}$ implementation with the same fault model as in previous publications. Our proposed attack targets the first round calculations of the AES and necessitates choosing the input message and obtaining the ciphertext computed by the card. Compared to previous CFA on masked AES, it does not require a large number of messages to be encrypted by the card.

Notation: We denote by $M = (M_0, \ldots, M_{15})$ the input message and by $K = (K_0, \ldots, K_{15})$ the key used by the card for encrypting M. Given a message M, we also denote by $(M|\,\text{condition})$ the message M modified so that the condition holds. For instance, messages $(M|M_j = 0)$ and $(M|M_j = 1)$ are identical except on the jth byte which is 0 in first case and 1 in the other. We will also refer to a *faulty* computation or a result thereof by means of the superscript symbol $^{\lightning}$. For instance $C^{\lightning}$ will refer to a faulty ciphertext.

8.4.1 Fault Model

We consider the following fault model: the result of an operation XOR can be set to zero – or to a not necessarily known constant value – by the attacker. This model has previously been assumed in several fault analysis papers and can be considered as realistic since practical results were also presented.

In practice such kind of fault effect can be induced in a card by the following events:

- An operation can be bypassed. For instance the instruction to be executed is replaced by a NOP. As explained in [103] the opcode fetched by the microprocessor may be replaced by 0x00 that, in some products, corresponds to the NOP operation. In this case the expected computation is not done, and the result register keeps its previous value that may be either 0 or another constant value. If the value is not constant the attack will be not possible.
- A loop counter can be modified, for instance in [4] the AddRoundKey operation is bypassed on some bytes by modifying (reducing) a counter value.

- The processing in the ALU can be perturbed and an XOR result can thus be modified to zero or a constant value.

Figure 1 gives an example of a typical code on an 8-bit microprocessor which can be attacked by what has been presented above. It represents the XOR operation between the masked message and the masked key carried out at the beginning of a secure AES. We can observe in this code that our fault model can be used to model different effects, if a NOP operation is executed, if a XOR result is forced to 0; if the counter value R2 is modified or if the JNZ operation is perturbed, etc...

Listing 1 Example code of masked AES AddRoundKey

```
Addition:
  MOV  R0, #address_message_masked
  MOV  R1, #address_key_masked
  MOV  R2, #16

LoopMessageXorKey:
  MOV  A, @R0
  MOV  B, @R1
  XOR  A, B
  MOV  @R0, A
  INC  R0
  INC  R1
  DEC  R2
  JNZ  LoopMessageXorKey
```

8.4.2 Attack on the First Key Addition

As in [4] we assume that the key addition before the first round can be perturbed and one or many bytes resulting from this addition can be set to zero. We describe our analysis for the first bytes M_0 and K_0 of the message and the key. The analysis will be identical for the other byte indices.

We denote by $rm = (rm_0, \ldots, rm_{15})$ and by $rk = (rk_0, \ldots, rk_{15})$ the two 16-byte random masks on the message and the key respectively. The resulting mask of the XOR between the message and the key is denoted by $r = rm \oplus rk$.

For a normal execution the first byte of the XOR result is:

$$B_0 = (M_0 \oplus rm_0) \oplus (K_0 \oplus rk_0) = M_0 \oplus K_0 \oplus r_0$$

For a faulty execution we have:

$$B_0^{\lightning} = 0$$

The key observation is that the effect of the fault is to introduce a differential δ on the byte value just before the first round S-Box computation.

$$B_0^{\lightning} = B_0 \oplus \delta, \quad \text{with} \quad \delta = M_0 \oplus K_0 \oplus r_0$$

The same effect on the execution would have been obtained, without a fault, when using an input message which differs from the initial one by this differential.

Considering without loss of generality an initial message $M = (0, \dots, 0)$, the differential then reduces to $\delta = K_0 \oplus r_0$, and we have a collision opportunity corresponding to the following equation:

$$C^{\lightning} = AES^{\lightning}(M) = AES(M|M_0 = \delta)$$

Note that a normal execution with $(M|M_0 = \delta)$ will produce a collision with $C^{\lightning}$ whatever the random mask values for this execution.

For any ciphertext $C^{\lightning}$ obtained by injecting a fault we have the following properties:

1. $C^{\lightning}$ is characteristic of the mask value r_0 in the faulty execution.
2. We can recover the value $K_0 \oplus r_0$ of this execution by identifying which input message leads to a collision with $C^{\lightning}$ without fault.

By computing $AES(M|M_0 = u)$ for all $u = 0, \dots, 255$, we can identify the u value which verifies the relation:

$$u = \delta = K_0 \oplus r_0$$

At this point we are able, for any faulty execution, to recover the value of $\delta = K_0 \oplus r_0$ involved in that execution. It is then possible to reproduce this analysis many times to obtain k ($k \leq 256$) such relations for k different values δ_i, $i = 1, \dots, k$, and store the power consumption curve W_i of the faulty execution corresponding to each δ_i.

Now, observe the following property: for any possible guess g about K_0, $g = 0, \dots, 255$, we obtain a unique set $S_g = \{r_{0,1}, \dots, r_{0,k}\}$ of k random mask values. In the HODPA resistant implementation these random values are generated and manipulated in the card at different moments during the inversion in $GF(2^4)$ and during the MixColumn computation applied to the mask in the first round. It is then possible to correlate these random values with the power curves W_i. By computing the linear correlation factor between the set of curves $\{W_1, \dots, W_k\}$ and the set of random masks $\{r_{0,1}, \dots, r_{0,k}\}$, the most important correlation peak over the different guesses identifies the correct set of random values manipulated in the card and thus indicates that the corresponding guess g is equal to the secret key byte K_0.

The analysis can be repeated on the next bytes of the key addition operation to recover the other key bytes K_1 to K_{15}.

We summarize the different steps of the attack in Figure 8.1. Note that in phase 2 of the attack, the expected number of faulty executions needed to obtain a new informative δ_i grows constantly with i. The expected number of faults N_k required to gather k relations is equal to

$$N_k = \sum_{i=1}^{k} \frac{256}{256 - (i-1)}$$

so that an average of 126 faults generates 100 relations, while the complete set of 256 relations requires $N_{256} = 1\,568$ faults.

Remark: Our attack is also applicable when the fault effect does not result in a 0 value but in an unknown constant c. Instead of recovering the 16-byte key K we recover $K \oplus (c, \ldots, c)$. Then we just have to exhaust all 256 keys until a key matching some correct plaintext/ciphertext pair is found.

Alg. 8.1 The attack algorithm on key byte K_n

1: Phase 1: dictionary precomputation
2: $M = (M_0, \ldots, M_{15}) \leftarrow (0, \ldots, 0)$
3: **for** $u = 0$ to 255 **do**
4: $\quad C_u \leftarrow AES(M | M_n = u)$
5: **end for**
6: Phase 2: collision search
7: $\Gamma = \varnothing$
8: $i \leftarrow 1$
9: **while** $i < k$ **do**
10: $\quad C^{\lightning} = AES^{\lightning}(M)$
11: $\quad$ **if** $C^{\lightning} \notin \Gamma$ **then**
12: $\quad\quad \delta_i \leftarrow u$ s.t. $C^{\lightning} = C_u$ with $u \in \{0, \ldots, 255\}$
13: $\quad\quad W_i \leftarrow$ power curve of the faulted execution
14: $\quad\quad \Gamma \leftarrow \Gamma \cup \{C^{\lightning}\}$
15: $\quad\quad i \leftarrow i + 1$
16: $\quad$ **end if**
17: **end while**
18: Phase 3: correlation
19: **for** $g = 0$ to 255 **do**
20: $\quad$ **for** $i = 1$ to k **do**
21: $\quad\quad r_{n,i} \leftarrow \delta_i \oplus g$
22: $\quad\quad \rho_g \leftarrow$ correlation trace between $\{r_{n,1}, \ldots, r_{n,k}\}$ and $\{W_1, \ldots, W_k\}$
23: $\quad$ **end for**
24: **end for**
25: **return** $K_n \leftarrow g$ which gives the highest correlation peak

8.5 Countermeasures

While data randomization with a full mask (i.e. 16 different bytes) is enough to protect the AES algorithm against the Amiel et al. attack as well as high order differential analysis, it is not sufficient against our combined attack. Below we mention some possible countermeasures.

8.5.1 Inverse computation

A classical and efficient countermeasure used to protect cryptographic algorithms against fault attacks in smartcards is the verification of the computation done. Before returning the ciphertext the card performs the inverse computation on the

Table 8.2: Performance and memory costs (bytes) for two Fault Analysis resistant implementations

	Cycles	ROM	RAM
AES_{HODPA} with Inverse Computation	102 000	5 500	75
AES_{HODPA} with 6 Duplicated Rounds	81 000	5 900	91

result. If the value obtained corresponds to the input message then the computation is considered valid and the card can return the result.
However if the comparison is not successful, it means that a fault was generated during one of the two cryptographic computations. In this case, nothing is returned by the card. As our attack requires faulty ciphertexts, it is not applicable when the inverse computation countermeasure is implemented.

8.5.2 Duplicated Rounds

An alternative to the previous countermeasure consists in duplicating the execution of the rounds exposed to the attacks, for instance the first and the last rounds. The rounds which are duplicated must be performed with two different masks. Their executions are carried out together, and bytes are processed in a random order regardless of their masks. At the end of the round the following property must hold: the addition of the two results must be equal to the addition of the masks. If this property does not hold then a fault is detected. In order to protect against DFA, it is recommended to duplicate the first three and the last three rounds.

8.5.3 Data error

Another way to protect an algorithm may be the deliberate introduction of data errors appearing under some kinds of sequence flow disruptions. This notion of infective countermeasure can be applied to loop counters, round counters,. . . Some infectious data are supposed to be equal to zero or to a fixed value in a normal execution, but when a fault modifies a loop counter (for instance during the AddRoundKey operation in AES) these values become erroneous when they are XORed with this counter.

8.5.4 Checksums

Data used at the beginning of a cryptographic algorithm (message, key, mask, ...) may be associated with a checksum. After executing sensitive operations a checksum on the obtained data is computed and compared to these values stored in memory. A comparison error implies that a fault occurred during this part of the algorithm. These checksums can be carried out using hardware mechanisms.

We have implemented the first two countermeasures on the HODPA secure algorithm described in section 8.3. Their performances and memory costs are presented in the Table 8.2.

8.6 Passive and Active Combined Attack on Masked AES with Safe Errors

Here we present a variant of our combined attack which is not precisely based on collisions but rather on safe errors, also known as ineffective faults.

In this variant, the way to obtain the knowledge of $K_n \oplus r_n$ for some faulty execution differs from the attack described in subsection 8.4.2. Instead of comparing a faulty ciphertext $C^{\lightning} = AES^{\lightning}(M|M_n = 0)$ with a pool of normal ciphertexts $\{C_u = AES(M|M_n = u)\}$, the attacker repeatedly compares some normal ciphertext $C_u = AES(M|M_n = u)$ with a faulty one $C_u^{\lightning} = AES^{\lightning}(M|M_n = u)$ obtained with the *same* input, until both ciphertexts collide. Once this collision occurs the attacker knows that $K_n \oplus r_n = u$.

At first sight this variant may seem irrelevant since the distributions of the two random variables $C_u^{\lightning}$ and $C_0^{\lightning}$ are the same. Also this variant requires 256 faulty executions on average to obtain one single collision, while the previously described attack requires only one.

The great advantage of the safe errors variant becomes clear when the HODPA resistant implementation is also protected against CFA, either by means of the *inverse computation* countermeasure or by means of the *duplicated rounds* one. In both cases the attacker identifies the collision event each time a result is returned. Indeed the result is returned whenever the fault has not been detected, that is whenever it was safe and had no local effect on the XOR result. Note that it is not possible to distinguish a *safe error* event from a *no fault at all* event. Consequently the safe errors variant may be difficult to perform in practice if the fault injection tool is not highly reliable.

An interesting and unexpected property of the safe errors variant is that it is easier to perform when the computation checking countermeasures are implemented than when they are not. Indeed when either of these countermeasures is present the attack consists in a known message attack, otherwise it is a chosen message attack.

8.6.1 Countermeasures

The countermeasures presented in section 8.5 no longer work here as no data has been modified. Then the question is how can we prevent attacks which do not modify data processed by the card? It seems to be an open problem and we do not have any good response. However we can mention several mechanisms which can complicate the attacker's task.

We have not yet addressed hardware mechanisms. As stated previously by Blömer and Seifert [16] we must insist on the fact that efficient hardware mechanisms to detect or resist light injections help software implementations and help to prevent many fault attacks.

The randomization (time or order) during execution is also a good means to destabilize the attacker as he does not exactly know where the targeted data is manipulated.

Due to the somewhat large number of fault injections required to perform the safe errors variant, an efficient means to defeat the attacker could be to limit the number of possible faulty executions. If more than a specified number of faults is detected the card can kill itself or at least refuse to answer except under privileged conditions. Note however that this principle may be difficult to implement in practice depending on the card operating environment and requirements.

The best solution will be to mask the key with a value which is never manipulated during the processing. It is then not possible to correlate power curves with the mask and the only data that an attacker can find would is the masked key.

8.7 Targeted AES Implementation

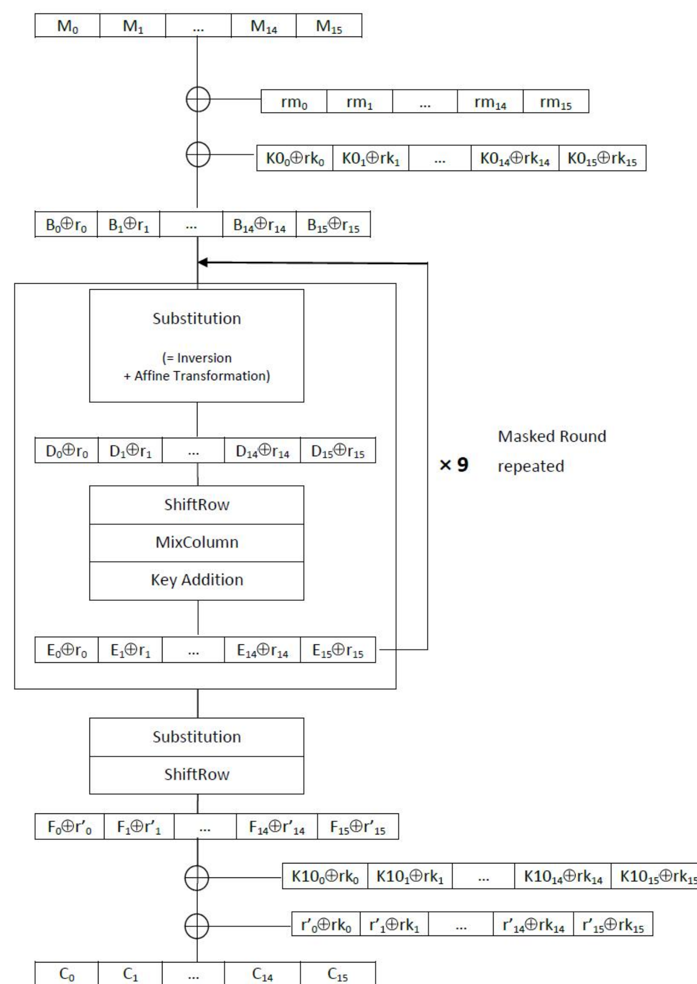


Figure 8.1: Secure HODPA Implementation

Chapter 9

Square Always

9.1 Introduction

Nowadays most embedded devices implementing public key cryptography use RSA [86] for encryption and signature schemes, or cryptographic primitives over $(\mathbb{F}_p, \times)$ such as DSA [41] and the Diffie-Hellman key agreement protocol [36]. All these algorithms require the computation of modular exponentiations. Since the emergence of the so-called *side-channel analysis*, embedded devices implementing these cryptographic algorithms must be protected against a wider and wider class of attacks.

Moreover, the cost and timing constraints are crucial in many applications of embedded devices (e.g. banking, transport, etc.). This often requires cryptographic implementors to choose the best compromise between security and speed. Improving the efficiency of algorithms or countermeasures generates thus a lot of interest in the industry.

An exponentiation is generally processed using a sequence of multiplications, some of them having different operands and some of them being squarings. In [7], Amiel et al. showed that this distinction can provide exploitable side-channel leakages to an attacker. Classical countermeasures consist of using exponentiation algorithms where the sequence of multiplications and squarings does not depend on the secret exponent.

Our contribution is to propose a new exponentiation scheme using squarings only, which is faster than the classical countermeasures. Also, we introduce new algorithms having a particularly low cost when two squarings can be parallelized.

This chapter is organized as follow: in section 9.2 we recall classical exponentiation algorithms and present some well-known side-channel attacks and countermeasures. Then we propose our new countermeasure in section 9.3 and study its efficiency from the parallelization point of view in section 9.4. Finally we present some practical results in section 9.5.

9.2 Background on Exponentiation on Embedded Devices

We recall in this section some classical exponentiation algorithms. First we present the *square-and-multiply* algorithms upon which are based most of the exponentiation methods. Then we introduce the *side-channel analysis* and in particular the *simple power analysis* (SPA). We present some algorithms immune to this attack, and we finally recall a particular side-channel attack aimed at distinguishing squarings from multiplications in an exponentiation operation.

9.2.1 Square-and-Multiply Algorithms

Many exponentiation algorithms have been proposed in the literature. Among the numerous references an interested reader can refer for instance to [69] for details. Alg. 9.1 and Alg. 9.2 are two variants of the classical square-and-multiply algorithm which is the simplest approach to compute an RSA exponentiation.

Alg. 9.1 Left-to-Right Square-and-Multiply Exponentiation

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\dots d_0)_2$
Output: $m^d \mod n$

1: $a \leftarrow 1$
2: **for** $i = k-1$ to 0 **do**
3:     $a \leftarrow a^2 \mod n$
4:     **if** $d_i = 1$ **then**
5:         $a \leftarrow a \times m \mod n$
6:     **end if**
7: **end for**
8: **return** a

Alg. 9.2 Right-to-Left Square-and-Multiply Exponentiation

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\dots d_0)_2$
Output: $m^d \mod n$

1: $a \leftarrow 1$; $b \leftarrow m$
2: **for** $i = 0$ to $k-1$ **do**
3:     **if** $d_i = 1$ **then**
4:         $a \leftarrow a \times b \mod n$
5:     **end if**
6:     $b \leftarrow b^2 \mod n$
7: **end for**
8: **return** a

Considering a balanced exponent d, these algorithms require on average $1S + 0.5M$ per bit of exponent to perform the exponentiation – S being the cost of a modular squaring and M the cost of a modular multiplication. It is generally considered in the literature – and corroborated by our experiments – that on cryptographic coprocessors $S \approx 0.8M$.

These algorithms are no longer used in embedded devices for security applications since the emergence of the side-channel analysis.

9.2.2 Side-Channel Analysis on Exponentiation

Side-channel analysis was introduced in 1996 by Kocher in [60] and completed in [61]. Many attacks have been derived in the following years.

On one hand, *passive* attacks rely on the following physical property: a microprocessor is physically made of thousands of logical gates switching differently depending on the executed operations and on the manipulated data. Therefore the power consumption and the electromagnetic radiation, which depend on those gates switchings, reflect and may leak information on the executed instructions and the manipulated data. Consequently, by monitoring such side-channels of a device performing cryptographic operations, an observer may infer information on the implementation of the program executed and on the – potentially secret – data involved.

On the other hand, *active* attacks intend to physically tamper with computations and/or stored values in memories. Such effects are generally obtained using clock or power glitches, laser beam, etc.

Finally some works [6] have highlighted the fact that passive and active attacks may be combined to threaten implementations applying countermeasures against both of them but not against their simultaneous use.

In the remainder of this section we focus on two passive attacks : the SPA presented hereafter with classical countermeasures, and a particular analysis from [7] discussed in subsection 9.2.3.

Simple Power Analysis

Simple side-channel analysis [59] consists in observing a difference of behavior depending on the value of the secret key on the component performing cryptographic operations by using a single measurement.

In the case of an exponentiation, the original SPA is based on the fact that, if the squaring operation has a different pattern than a multiplication, the secret exponent can be directly read on the curve. For instance, in Alg. 9.1, a 0 exponent bit implies a squaring to be followed by another squaring, while a 1 bit causes a multiplication to follow a squaring. Classical countermeasures consist of using *regular* algorithms or applying the *atomicity* principle, as detailed in the following.

Regular Algorithms

These algorithms include the well known *square-and-multiply always* and Montgomery ladder algorithms [75, 55]. The latter is presented hereafter in Alg. 9.3. It is generally preferred over the square-and-multiply always method since it does not involves dummy multiplications which makes it naturally immune to the C safe-error attacks [107, 55].

Such regular algorithms perform one squaring and one multiplication at every iteration and thus require $1M + 1S$ per exponent bit.

Alg. 9.3 Montgomery Ladder Exponentiation

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\dots d_0)_2$
Output: $m^d \mod n$
1: $R_0 \leftarrow 1$; $R_1 \leftarrow m$
2: **for** $i = k-1$ to 0 **do**
3: $\quad R_{1-d_i} \leftarrow R_0 \times R_1 \mod n$
4: $\quad R_{d_i} \leftarrow {R_{d_i}}^2 \mod n$
5: **end for**
6: **return** R_0

Alg. 9.4 Left-to-Right Multiply Always Exponentiation

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\dots d_0)_2$
Output: $m^d \mod n$
1: $R_0 \leftarrow 1$; $R_1 \leftarrow m$; $i \leftarrow k-1$; $t \leftarrow 0$
2: **while** $i \geq 0$ **do**
3: $\quad R_0 \leftarrow R_0 \times R_t \mod n$
4: $\quad t \leftarrow t \oplus d_i$; $i \leftarrow i - 1 + t$ [$\oplus$ is bitwise XOR]
5: **end while**
6: **return** R_0

Atomicity Principle

This method, presented by Chevallier-Mames et al. in [23], can be applied to protect the square-and-multiply algorithm against the SPA. It yields the so-called *multiply always* algorithm, since all squarings are performed as classical multiplications. We present a left-to-right multiply always algorithm in Alg. 9.4.

The interest of the multiply always algorithm is its better performances compared to the regular ones. Indeed it performs an exponentiation using on average $1.5M$ per exponent bit.

9.2.3 Distinguishing Squarings from Multiplications

Amiel et al. showed in [7] that the average Hamming weight of the output of a multiplication $x \times y$ has a different distribution whether:

- the operation is a squaring performed using the multiplication routine, i.e. $x = y$, x uniformly distributed in $[0, 2^k - 1]$,
- or the operation is an “actual” multiplication, i.e. x and y independent and uniformly distributed in $[0, 2^k - 1]$.

This attack can thus target an atomic implementation such as Alg. 9.4 where the same multiplication operation is used to perform $x \times x$ and $x \times y$.

First, many exponentiation curves using a fixed exponent but variable data have to be acquired and averaged. Then, considering the average curve, the aim of the attack is to reveal if two consecutive operations are identical – i.e. two squarings – or different – i.e. a squaring and a multiplication. As in the classical SPA, two consecutive squarings reveal that a 0 bit has been manipulated whereas a squaring followed by a multiplication reveals a 1 bit. This information is obtained using

the above-mentioned leakage by subtracting the parts of the average curve corresponding to two consecutive operations: peaks occur if one is a squaring and the other is a multiplication while subtracting two squarings should produce only noise. It is worth noticing that no particular knowledge on the underlying hardware implementation is needed which in practice increases the strength of this analysis.

A classical countermeasure against this attack is the randomization of the exponent[1], i.e. $d^* \leftarrow d + r\varphi(n)$, r being a random value. The result is obtained as $m^d \bmod n = m^{d^*} \bmod n$.

In spite of the possibility to apply the exponent randomization, this attack brings into light an intrinsic flaw of the multiply always algorithm: the fact that at some instant a multiplication performs a squaring ($x \times x$) or not ($x \times y$) depending on the exponent. In the rest of this chapter we propose new atomic algorithms that are exempt from this weakness.

9.3 Square Always Countermeasure

We present in this section new exponentiation algorithms which simultaneously benefit from efficiency of the atomicity principle and immunity against the aforementioned weakness of the multiply always method.

9.3.1 Principle

It is well known that a multiplication can be performed using squarings only. Therefore we propose the following countermeasure which consists in using either expression (Equation 9.1) or (Equation 9.2) to perform all the multiplications in the exponentiation. Combined with the atomicity principle, this countermeasure completely prevents the attack described in subsection 9.2.3 since only squarings are performed.

$$x \times y = \frac{(x+y)^2 - x^2 - y^2}{2} \tag{9.1}$$

$$x \times y = \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2 \tag{9.2}$$

At the first glance, (Equation 9.1) requires three squarings to perform a multiplication whereas (Equation 9.2) requires only two. Further analysis reveals however that using (Equation 9.1) or (Equation 9.2) in Alg. 9.1 and 9.2 has always the cost of replacing multiplications by twice more squarings. Indeed, notice that in the multiplication $a \leftarrow a \times m$ of Alg. 9.1 m is a constant operand. Therefore implementing $a \times m$ using (Equation 9.1) yields $y = m$, thus $m^2 \mod n$ can be computed only once at the beginning of the exponentiation. The cost of computing y^2 can then be neglected.

[1] Notice however that the randomization of the message has no effect on this attack, or even makes it easier by providing the required data variability.

This trick does not apply to Alg. 9.2 since no operand is constant in step 4. However $b \leftarrow b^2$ is the following operation. Using (Equation 9.1) in Alg. 9.2 then yields to store $t \leftarrow y^2$ and save the following squaring: $b \leftarrow t$. The resulting cost is thus equivalent as trading one multiplication for two squarings.

Remark In our context, (Equation 9.1) or (Equation 9.2) refer to operations modulo n. Notice however that divisions by 2 in these equations require neither inversion nor multiplication. For example, we recommend computing $z/2 \mod n$ in the following atomic way:

$t_0 \leftarrow z$
$t_1 \leftarrow z + n$
$\alpha \leftarrow z \mod 2$
return $t_\alpha/2$

9.3.2 Atomic Algorithms

Trading multiplications for squarings in Alg. 9.1 and 9.2 just requires to apply (Equation 9.1) or (Equation 9.2) at step 5 in Alg. 9.1 or step 4 in Alg. 9.2. However the resulting algorithms would still present a leakage since different operations would be performed when processing a 0 or 1 bit. Hence it is necessary to apply the atomicity principle on these algorithms.

This step is achieved by identifying a minimal pattern of operations to be performed on each loop iteration and rewrite the algorithms using this pattern. For the considered algorithms, the minimal pattern should obviously contain a single squaring since it is the only operation required by the processing of a 0 bit and performing dummy squarings would lessen the performances of the algorithm. An addition, subtraction and division by 2 should also be present to compute (Equation 9.1) or (Equation 9.2). Finally some more operations are required to manage the loop counter and the pointer on exponent bits.

Algorithm 9.5 presented hereafter details how to implement atomically the square always method in a left-to-right exponentiation using (Equation 9.1).

As in [23] we use a matrix for a more readable and efficient implementation:

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 2 & 1 & 1 & 1 & 2 & 1 \\ 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 0 \\ 1 & 1 & 3 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 3 & 3 & 3 & 0 & 3 & 3 & 1 & 1 & 3 & 1 \end{pmatrix}$$

The main loop of Alg. 9.5 can be viewed as a four state machine where each row j of M define the operands of the atomic pattern. The atomic pattern itself is given by the content of the loop, i.e. steps 4 to 9. An exponent bit d_i is processed by the state $j = 0$ (resp. $j = 3$) if the previous bit d_{i+1} is a 0 (resp. a 1). This state is followed by the processing of the next bit if $d_i = 0$, or by the states $j = 1$ and $j = 2$ if $d_i = 1$. For more clarity, we present below the four sequences of operations corresponding to each state. The dummy operations are identified by a $\star$.

Alg. 9.5 Left-to-Right Square Always Exponentiation with (Equation 9.1)

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\ldots d_0)_2$

Output: $m^d \mod n$

1: $R_0 \leftarrow 1$; $R_1 \leftarrow m$; $R_2 \leftarrow 1$; $R_3 \leftarrow m^2/2 \mod n$
2: $j \leftarrow 0$; $i \leftarrow k-1$
3: **while** $i \geq 0$ **do**
4: $\quad R_{M_{j,0}} \leftarrow R_{M_{j,1}} + R_{M_{j,2}} \mod n$
5: $\quad R_{M_{j,3}} \leftarrow {R_{M_{j,3}}}^2 \mod n$
6: $\quad R_{M_{j,4}} \leftarrow R_{M_{j,5}}/2 \mod n$
7: $\quad R_{M_{j,6}} \leftarrow R_{M_{j,7}} - R_{M_{j,8}} \mod n$
8: $\quad j \leftarrow d_i(1 + (j \mod 3))$
9: $\quad i \leftarrow i - M_{j,9}$
10: **end while**
11: **return** R_0

$j=0$ ($d_i = 0$ or 1)	
$R_1 \leftarrow R_1 + R_1 \mod n$	$\star$
$R_0 \leftarrow {R_0}^2 \mod n$	
$R_2 \leftarrow R_1/2 \mod n$	$\star$
$R_1 \leftarrow R_1 - R_2 \mod n$	$\star$
$j \leftarrow d_i$	[$\star$ if $d_i = 0$]
$i \leftarrow i - (1 - d_i)$	[$\star$ if $d_i = 1$]

$j=2$ ($d_i = 1$)	
$R_1 \leftarrow R_1 + R_3 \mod n$	$\star$
$R_0 \leftarrow {R_0}^2 \mod n$	
$R_0 \leftarrow R_0/2 \mod n$	
$R_0 \leftarrow R_2 - R_0 \mod n$	
$j \leftarrow 3$	
$i \leftarrow i - 1$	

$j=1$ ($d_i = 1$)	
$R_2 \leftarrow R_0 + R_1 \mod n$	
$R_2 \leftarrow {R_2}^2 \mod n$	
$R_2 \leftarrow R_2/2 \mod n$	
$R_2 \leftarrow R_2 - R_3 \mod n$	
$j \leftarrow 2$	
$i \leftarrow i$	$\star$

$j=3$ ($d_i = 0$ or 1)	
$R_3 \leftarrow R_3 + R_3 \mod n$	$\star$
$R_0 \leftarrow {R_0}^2 \mod n$	
$R_3 \leftarrow R_3/2 \mod n$	$\star$
$R_1 \leftarrow R_1 - R_3 \mod n$	$\star$
$j \leftarrow d_i$	
$i \leftarrow i - (1 - d_i)$	[$\star$ if $d_i = 1$]

We also present in Alg. 9.6 a right-to-left variant of the square always exponentiation using (Equation 9.2). This algorithm requires the following matrix:

$$M = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 & 2 & 1 \\ 2 & 1 & 2 & 2 & 1 & 0 & 1 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 & 1 \end{pmatrix}$$

As for the previous algorithm, the main loop of Alg. 9.6 has four states. Here, the state $j = 0$ corresponds to the processing a 0 bit and the sequence $j = 1$, $j = 2$, and $j = 3$ corresponds to the processing of a 1 bit, as detailed below.

$j=0$ ($d_i = 0$)	
$j \leftarrow 0$	[$\star$ if j was 0]
$R_0 \leftarrow R_0 + R_0 \mod n$	$\star$
$R_2 \leftarrow R_0/2 \mod n$	$\star$
$R_0 \leftarrow R_0 - R_2 \mod n$	$\star$
$R_0 \leftarrow {R_0}^2 \mod n$	
$i \leftarrow i + 1$	

$j=2$ ($d_i = 1$)	
$j \leftarrow 2$	
$R_0 \leftarrow R_2 + R_0 \mod n$	$\star$
$R_1 \leftarrow R_1/2 \mod n$	
$R_0 \leftarrow R_0 - R_2 \mod n$	$\star$
$R_1 \leftarrow {R_1}^2 \mod n$	
$i \leftarrow i$	$\star$

Alg. 9.6 Right-to-Left Square Always Exponentiation with (Equation 9.2)

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\ldots d_0)_2$
Output: $m^d \mod n$
1: $R_0 \leftarrow m$; $R_1 \leftarrow 1$; $R_2 \leftarrow 1$
2: $i \leftarrow 0$; $j \leftarrow 0$
3: **while** $i \leq k-1$ **do**
4: $\quad j \leftarrow d_i(1 + (j \mod 3))$
5: $\quad R_{M_{j,0}} \leftarrow R_{M_{j,1}} + R_0 \mod n$
6: $\quad R_{M_{j,2}} \leftarrow R_{M_{j,3}}/2 \mod n$
7: $\quad R_{M_{j,4}} \leftarrow R_{M_{j,5}} - R_{M_{j,6}} \mod n$
8: $\quad R_{M_{j,3}} \leftarrow {R_{M_{j,3}}}^2 \mod n$
9: $\quad i \leftarrow i + M_{j,7}$
10: **end while**
11: **return** R_1

$j = 1$ ($d_i = 1$)		$j = 3$ ($d_i = 1$)	
$j \leftarrow 1$		$j \leftarrow 3$	
$R_2 \leftarrow R_1 + R_0 \mod n$		$R_0 \leftarrow R_0 + R_0 \mod n$	$\star$
$R_2 \leftarrow R_2/2 \mod n$		$R_0 \leftarrow R_0/2 \mod n$	$\star$
$R_1 \leftarrow R_0 - R_1 \mod n$		$R_1 \leftarrow R_2 - R_1 \mod n$	
$R_2 \leftarrow {R_2}^2 \mod n$		$R_0 \leftarrow {R_0}^2 \mod n$	
$i \leftarrow i$	$\star$	$i \leftarrow i + 1$	

9.3.3 Performance Analysis

Algorithms 9.5 and 9.6 are mostly equivalent in terms of operations realized in a single loop. The number of dummy operations (additions, subtractions and halvings) introduced to fill the atomic blocks are the same in the two versions – it is generally considered that the cost of these operations is negligible compared to multiplications and squarings. Both algorithms require $2S$ per exponent bit on average or $1.6M$ if S/M =0.8 which represents a theoretical 11.1% speed-up over Alg. 9.3 which is the fastest known regular algorithm immune to the attack from [7]. Table 9.1 compares the efficiency of the multiply always, Montgomery ladder, and square always algorithms when $S = M$ and S/M =0.8.

In addition, our algorithms can be enhanced using the sliding window or m-ary exponentiation techniques [69, 51] while the Montgomery ladder cannot. These techniques are known to provide a substantial speed-up on Alg. 9.4 when extra memory is available. Though we did not investigate this path, we believe that a comparable trade-off between space and time can be expected.

Algorithm	General cost	S/M =1	S/M =0.8	#registers
Multiply always (9.4)	$1.5M$	$1.5M$	$1.5M$	2
Montgomery ladder (9.3)	$1M$+$1S$	$2M$	$1.8M$	2
L.-to-r. Square always (9.5)	$2S$	$2M$	$1.6\boldsymbol{M}$	4
R.-to-l. Square always (9.6)	$2S$	$2M$	$1.6\boldsymbol{M}$	3

Table 9.1: Comparison of the expected cost of SPA protected exponentiation algorithms (including the multiply always which is not immune to the attack from [7])

9.3.4 Security Considerations

Our algorithms are protected against the SPA by the implementation of the atomicity principle. The analysis from [7] cannot apply either since only squarings are involved. As a matter of comparison, notice that the exponent blinding countermeasure does not fundamentally remove the source of the leakage but only renders this attack practically infeasible. Embedded implementations should also be protected against the *differential power analysis* (DPA) which we do not detail in this study. However it is worth noticing that classical DPA countermeasures, like exponent or modulus randomization, can be applied as well. The interested reader may refer to [61, 34].

We recommend implementing Alg. 9.6 instead of Alg. 9.5 since left-to-right algorithms are vulnerable to the chosen message SPA and *doubling attack* [42], and more subject to combined attacks [6]. Besides, Alg. 9.6 requires one less register than Alg. 9.5.

It is well-known that algorithms using dummy operations generally succumb to safe-error attacks. Immunity to C and M safe-errors can be easily obtained by applying the exponent randomization technique, which also prevent the DPA. Nevertheless, special care has been taken in our algorithms to ensure that inducing a fault in any of the dummy operations would produce an erroneous result. For instance, in the following sequence of dummy operations in Alg. 9.6 ($j = 0$), no operation can be tampered with without corrupting R_0 and thus the result of the exponentiation:

$$
\begin{aligned}
R_0 &\leftarrow R_0 + R_0 \mod n \\
R_2 &\leftarrow R_0/2 \mod n \\
R_0 &\leftarrow R_0 - R_2 \mod n
\end{aligned}
$$

Only operations $i \leftarrow i$ and $j \leftarrow 0$, appearing in some instances of Alg. 9.5 and 9.6 patterns, have not been protected for readability reasons. It is easy to fix these points: perform $i \leftarrow i \pm M_{j,\cdot} + \alpha$ instead of $i \leftarrow i \pm M_{j,\cdot}$ in Alg. 9.5 and 9.6 and add a step $i \leftarrow i - \alpha$ in the loop. The $j \leftarrow d_i(1 + \dots)$ operation should be protected in the same manner. In the end, our algorithms are immune to C safe-error attacks.

Further work may focus on implementing on our algorithms the *infective computation* strategy presented by Schmidt et al. in [91] in order to counterfeit the combined attacks.

9.4 Parallelization

It is well known that the Montgomery ladder algorithm is well suited for parallelization. It is thus natural to ask if the square always algorithms have the same property. For example the two squarings needed to perform a classical multiplication using (Equation 9.2) are independent and can therefore be performed simultaneously. The same strategy applies for (Equation 9.1).

We believe that the interest of this section extends beyond the scope of embedded systems. Nowadays most of computers are provided with several processors which enables using parallelized algorithms to speed-up computations.

9.4.1 Parallelized Algorithms

We noticed that right-to-left exponentiations are more suited for parallelization than their left-to-right counterpart since more operations are independent. For example in Alg. 9.2 one can first perform all squarings (step 6), store all values corresponding to a $d_i = 1$, and then perform the remaining multiplications. We present in Alg. 9.7 a right-to-left square always algorithm using (Equation 9.2) and two parallel squaring blocks (i.e. two 1-operand multipliers). For a better readability Alg. 9.7 is not atomic and two operations o_1 and o_2 performed simultaneously are denoted $o_1 \,||\, o_2$.

Alg. 9.7 Right-to-Left Parallel Square Always Exponentiation with (Equation 9.2)

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1} \ldots d_0)_2$, require 5 k-bit registers a, b, R_0, R_1, R_2
Output: $m^d \mod n$

```
1: a ← 1 ; b ← m ; extra ← 0
2: for i = 0 to k − 1 do
3:     if d_i = 1 then
4:         if extra = 0 then
5:             R_0 ← (a − b)^2 mod n   ||   R_1 ← b^2 mod n
6:             a ← (a + b)^2 mod n   ||   R_2 ← R_1^2 mod n
7:             a ← (a − R_0)/4 mod n
8:             b ← R_1
9:             R_1 ← R_2
10:            extra ← 1
11:        else
12:            R_0 ← (a − b)^2 mod n   ||   a ← (a + b)^2 mod n
13:            a ← (a − R_0)/4 mod n
14:            b ← R_1
15:            extra ← 0
16:        end if
17:    else
18:        if extra = 0 then
19:            b ← b^2 mod n
20:        else
21:            b ← R_1
22:            extra ← 0
23:        end if
24:    end if
25: end for
26: return a
```

Algorithm 9.8 is an atomic variant of Alg. 9.7. It requires two extra registers compared to the non atomic version and the following matrices:

$$M = \begin{pmatrix} 1 & 1 & 5 & 6 & 5 & 5 & 5 & 0 & 1 \\ 0 & 6 & 4 & 3 & 0 & 1 & 3 & 1 & 1 \\ 2 & 5 & 3 & 1 & 5 & 5 & 5 & 0 & 0 \\ 2 & 5 & 0 & 6 & 0 & 1 & 5 & 0 & 1 \end{pmatrix} \qquad N = \begin{pmatrix} 1 & 1 & 0 \\ 5 & 2 & 2 \end{pmatrix}$$

Alg. 9.8 Right-to-Left Atomic Parallel Square Always Exp. with (Equation 9.2)

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\ldots d_0)_2$, require 7 k-bit registers R_0 to R_6
Output: $m^d \mod n$

1: $R_0 \leftarrow 1$; $R_1 \leftarrow m$; $v \leftarrow (0,0,0)$; $u \leftarrow 1$ [v_0 is i and v_1 is *extra* from Alg. 9.7]
2: **while** $v_0 \leq k-1$ **do**
3: $j \leftarrow d_{v_0}(v_1 + u + 1)$
4: $R_5 \leftarrow (R_0 - R_1)/2 \mod n$
5: $R_6 \leftarrow (R_0 + R_1)/2 \mod n$
6: $R_{M_{j,0}} \leftarrow {R_{M_{j,1}}}^2 \mod n$ || $R_{M_{j,2}} \leftarrow {R_{M_{j,3}}}^2 \mod n$
7: $R_{M_{j,4}} \leftarrow R_0 - R_2 \mod n$
8: $R_{M_{j,5}} \leftarrow R_3$
9: $R_{M_{j,6}} \leftarrow R_4$
10: $v_1 \leftarrow M_{j,7}$
11: $u \leftarrow M_{j,8}$
12: $t \leftarrow 1 - v_1(1 - d_{v_0+1})$
13: $R_{N_{t,0}} \leftarrow R_3$
14: $v_{N_{t,1}} \leftarrow 0$
15: $v_{N_{t,2}} \leftarrow v_{N_{t,2}} + 1$
16: $v_0 \leftarrow v_0 + u$
17: **end while**
18: **return** R_0

It is possible to further enhance the efficiency of Alg. 9.7 if more memory is available by storing more free squarings when 1's sequences are processed. This observation yields Alg. 9.9 which allows the storage of *extramax* simultaneous precomputed squarings using as many registers R_3, R_4, $\ldots R_{\mathit{extramax}+2}$. Alg. 9.9 with *extramax* $= 1$ is thus equivalent to algorithms 9.7 and 9.8. Though Alg. 9.9 is not atomic for readability reasons and because of the difficulty to write an atomic algorithm depending on a variable (here *extramax*), it should be possible to write an atomic version for each *extramax* value in the same way than we processed with Alg. 9.7.

Remark Notice that multiple assignments of steps 8, 13, and 20 may be traded for a cheap index increment if registers R_1, R_2, $\ldots$, $R_{\mathit{extramax}+2}$ are managed as a cyclic buffer.

9.4.2 Cost of Parallelized Algorithms

We demonstrate in Appendix section 9.6 that, as the length of the exponent tends to infinity, the cost per exponent bit of Alg. 9.9 tends to:

$$\left(1 + \frac{1}{4\,\mathit{extramax} + 2}\right) S$$

It yields a cost of $7S/6$ for Alg. 9.7, 9.8, and 9.9 with *extramax* $= 1$, $11S/10$ for *extramax* $= 2$, $15S/14$ for *extramax* $= 3$, etc. The difference between this limit

Alg. 9.9 Right-to-Left Generalized Parallel Square Always Exp. with (Equation 9.2)

Input: $m, n \in \mathbb{N}$, $m < n$, $d = (d_{k-1}d_{k-2}\dots d_0)_2$, $\mathit{extramax} \in \mathbb{N}^*$, require $\mathit{extramax}+4$ k-bit registers a, R_0, R_1, $\dots R_{\mathit{extramax}+2}$

Output: $m^d \mod n$

```
1:  a ← 1 ; R_1 ← m ; extra ← 0
2:  for i = 0 to k − 1 do
3:      if d_i = 1 then
4:          if extra < extramax then
5:              R_0 ← (a − R_1)^2 mod n  ||  R_{extra+2} ← R_{extra+1}^2 mod n
6:              a ← (a + R_1)^2 mod n  ||  R_{extra+3} ← R_{extra+2}^2 mod n
7:              a ← (a − R_0)/4 mod n
8:              (R_1, R_2, ... R_{extramax+1}) ← (R_2, R_3, ... R_{extramax+2})
9:              extra ← extra +1
10:         else
11:             R_0 ← (a − R_1)^2 mod n  ||  a ← (a + R_1)^2 mod n
12:             a ← (a − R_0)/4 mod n
13:             (R_1, R_2, ... R_{extramax+1}) ← (R_2, R_3, ... R_{extramax+2})
14:             extra ← extra −1
15:         end if
16:     else
17:         if extra = 0 then
18:             R_1 ← R_1^2 mod n
19:         else
20:             (R_1, R_2, ... R_{extramax+1}) ← (R_2, R_3, ... R_{extramax+2})
21:             extra ← extra −1
22:         end if
23:     end if
24: end for
25: return a
```

and costs actually observed in our simulations is negligible for 1024-bit or longer exponents.

It is remarkable that if S/M =0.8 these costs become respectively 0.93M, 0.88M, 0.86M, etc. per exponent bit. We believe that such performances cannot be achieved by binary algorithms using two parallelized 2-operands multiplication blocks. Indeed at least k multiplications have to be performed sequentially in Alg. 9.1 and 9.2, which requires at least $1M$ per exponent bit. Moreover when *extramax* tends to infinity, the cost of Alg. 9.9 tends to $1S$, which we believe to be the optimal cost of an exponentiation algorithm based on the binary decomposition of the exponent since k squarings at least have to be performed sequentially.

Table 9.2 summarizes the theoretical cost of parallelized algorithms cited in this study.

9.5 Practical Results

In this section, we briefly present practical implementation results of the non-parallelized square always algorithm. As discussed in subsection 9.3.4 we focused the right-to-left version.

Algorithm	General cost	$S/M = 1$	$S/M = 0.8$
Parallelized Montgomery ladder	$1M$	$1M$	$1M$
9.7, 9.8, 9.9 with *extramax* $= 1$	$7S/6$	$1.17M$	$\mathbf{0.93M}$
9.9 with *extramax* $= 2$	$11S/10$	$1.10M$	$\mathbf{0.88M}$
9.9 with *extramax* $= 3$	$15S/14$	$1.07M$	$\mathbf{0.86M}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
9.9 with *extramax* $\to \infty$	$1S$	$1M$	$\mathbf{0.8M}$

Table 9.2: Comparison of the expected cost of parallelized exponentiation algorithms

Algorithm	Key Length (b)	Code Size (B)	RAM used (B)	Timings (ms)
Montgomery ladder (9.3)	512	360	128	30
	1024	360	256	200
	2048	360	512	1840
Square Always (9.6)	512	510	192	28
	1024	510	384	190
	2048	510	768	1740

Table 9.3: On chip comparison of the Montgomery ladder and square always algorithms

We implemented this algorithm and the Montgomery ladder on an Atmel AT90SC smart card chip. This component is provided with an 8-bit AVR core and the AdvX coprocessor dedicated to long integer arithmetic. We used the Barrett reduction [11] to implement modular arithmetic.

We present in Table 9.3 the memory (code and RAM) and timing figures obtained with the chip and the AdvX running at 30 MHz. The observed speed-up of the square always algorithm over the Montgomery ladder is 5% on average. This is less than the predicted 11% but the difference can be explained by the neglected operations of the atomic pattern. Keep in mind that such results highly depend on the considered device and its hardware capabilities.

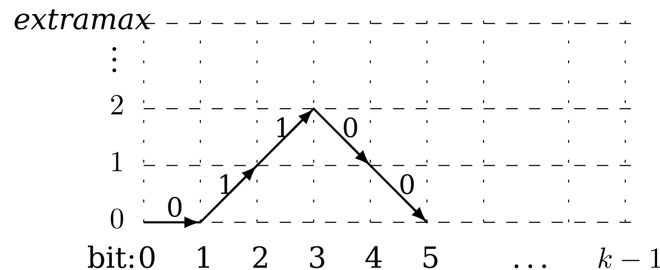
We performed careful SPA on both implementations and observed no leakage on power traces.

9.6 Cost of Algorithm 9.9

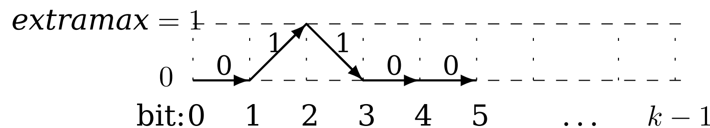
We present hereafter a demonstration of the claimed asymptotic cost of Alg. 9.9.

We first recall the principle of this algorithm: since 3 squarings are required to process a 1 bit, a fourth squaring slot is available at the same cost ($2S$). Thus, the algorithm scans the exponent from the right to the left and computes one squaring in advance at each 1 bit ($\nearrow$ in the following), within the limit of *extramax*. Then, as 0's are processed, the free squarings are consumed ($\searrow$) at null cost ($0S$). Two other cases may happen: first, a 1 bit can be processed but *extramax* squarings are already stored in registers, then one free squaring is consumed ($\searrow$) and $1S$ is enough to perform the two other squarings. Second, a 0 bit can be processed with no free squaring in registers (*extra* $= 0$). Only in this latter case one squaring is performed at the cost of $1S$ and the parallel squaring slot is wasted ($\rightarrow$).

We can consider the evolution of *extra* as exponent bits are processed using a diagram as below. For example, we have represented here the evolution of *extra* for the 5 first bits of an exponent $d = (d_{k-1} \ldots 00110)_2$ with *extramax* ≥ 2. The cost of the first 0 bit is $1S$ since *extra* $= 0$ at the beginning of the exponentiation, the cost of two next 1 bits is $2S$ each and *extra* is incremented, finally the two last 0 bits have cost $0S$ and *extra* is decremented. The total cost of the 5 bits is $5S$.



Observe now that the same bits have a higher cost if *extramax* $= 1$: as previously the two first bits 01 cost $1S$ and $2S$ respectively. However, the next 1 bit cannot lead to the computation of a second free squaring since *extramax* $= 1$. So the bit is processed at the cost of $1S$ and the free squaring is lost. Finally, the two last 0's cost $1S$ each since no free squaring is stored anymore. The cost of the sequence is $6S$.



For a given exponent and *extramax*, let's call a *c-cycle* a sequence of bits starting with *extra* $= c$, ending with *extra* $= c$, and inside which *extra* $> c$. In particular, we can decompose any exponent as a sequence of 0-cycles, except that the last one may be unterminated with *extra* > 0.

Then, let B_c^e stand for the expected number of bits of a c-cycle when *extramax* $= e$ and C_c^e its expected cost.

extramax $= 1$

For a random exponent and *extramax* $= 1$, a 0-cycle is "0" with probability 1/2 and "$1x$", $x \in \{0, 1\}$ otherwise. The cost of a 0-cycle "0" is $1S$ and the cost of a 0-cycle "$1x$" is $2S$ if $x = 0$ which happens with probability 1/2, or $3S$ if $x = 1$.

$$B_0^1 = 1/2 \times 1 + 1/2 \times 2 = 3/2$$

$$C_0^1 = 1/2 \times 1S + 1/2 \times (1/2 \times 2S + 1/2 \times 3S) = 7S/4$$

The expected cost of a 0-cycle with *extramax* $= 1$ is then $C_0^1/B_0^1 = 7S/6$ per bit. As the length of the exponent tends to infinity, the contribution of the possibly unterminated last 0-cycle becomes negligible. Therefore the cost per bit of a random exponent tends to the cost per bit of a 0-cycle as its length tends to infinity. So we can approximate the cost of algorithms 9.7, 9.8 and 9.9 to $7S/6$ for exponents of thousands of bits.

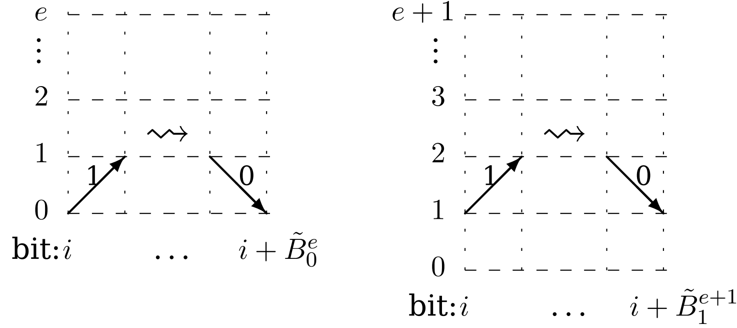
extramax $= e$

A 0-cycle starts with a 0 with probability 1/2 and with a 1 otherwise. In the first case its cost is $1S$ as previously. Let $\tilde{B}_c^e$, respectively $\tilde{C}_c^e$, denote the expected length, respectively the expected cost, of a c-cycle starting with a 1 bit when *extramax* $= e$.

$$B_0^e = 1/2 \times 1 + 1/2 \times \tilde{B}_0^e \tag{9.3}$$

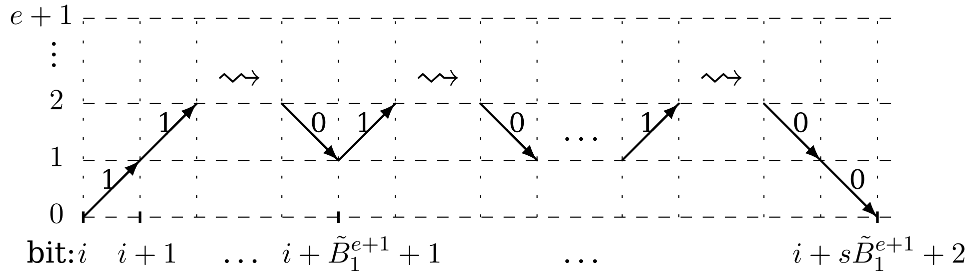
$$C_0^e = 1/2 \times 1S + 1/2 \times \tilde{C}_0^e \tag{9.4}$$

First we demonstrate that $\tilde{B}_0^e = 2e$. As depicted below, one can observe that $\tilde{B}_0^e = \tilde{B}_1^{e+1}$.



As depicted hereafter, the length $\tilde{B}_0^{e+1}$ of a 0-cycle with *extramax* $= e+1$ and starting by a 1 bit is $s\tilde{B}_1^{e+1} + 2$ where s is the number of inner 1-cycles starting by a 1 bit. Notice also that $s = i$ with probability $2^{-(i+1)}$, which gives:

$$\tilde{B}_0^{e+1} = 2 + \sum_{i=0}^{\infty} \frac{i\tilde{B}_1^{e+1}}{2^{(i+1)}} = 2 + \tilde{B}_1^{e+1} = 2 + \tilde{B}_0^e$$



$(\tilde{B}_0^e)_{e\geq 1}$ is thus an arithmetic progression with common difference 2 and $\tilde{B}_0^1 = 2$. This yields $\tilde{B}_0^e = 2e$.

In a same manner, we can observe that:

$$\tilde{C}_0^{e+1} = 2S + \sum_{i=0}^{\infty} \frac{i\tilde{C}_1^{e+1}}{2^{(i+1)}} = 2S + \tilde{C}_1^{e+1} = 2S + \tilde{C}_0^e$$

Since $\tilde{C}_0^1 = 5S/2$ we obtain that $\tilde{C}_0^e = (1/2 + 2e)S$.

Using the above results in (Equation 9.3) and (Equation 9.4), we obtain finally:

$$B_0^e = 1/2 \times 1 + 1/2 \times 2e = 1/2 + e$$

$$\text{and} \quad C_0^e = 1/2 \times 1S + 1/2 \times (1/2 + 2e)S = (3/4 + e)S$$

The expectation of the cost per bit of a 0-cycle is then:

$$\frac{C_0^e}{B_0^e} = \left(\frac{3/4 + e}{1/2 + e}\right) S = \left(1 + \frac{1}{4e + 2}\right) S$$

Therefore the expectation of the cost of Alg. 9.9 with *extramax* $= e$ tends then to $(1 + \frac{1}{4e+2})S$ as the length of the exponent tends to infinity.

Chapter 10

Simulation

10.1 Introduction

A typical product audit gets through different stages, from documentation to security test in order to validate its robustness. Two kind of attacks are usually used to assess the security of a device, SCA attacks and faults. Setting up a laser bench or a power consumption bench and running the required tests is a long task specially in the laser case and most often the laboratory have to choose some target IP and focus on those in order to bypass the security mechanism. Basically once a successful fault is found the analysis of the auditor ends and the company designing the product has to understand through the position and the time-frame where the fault where injected what exactly happened in the product in order to fix the flaw. It can be a really long and tedious process and the only way to verify if the patch is successful is to go through another laser test. Those tests are expensive and are time consuming.

We propose a novel method of simulating such attacks through software to greatly enhance the speed at which one can test the security of such products. Software power simulation is usually done by using software like Mathematica or python in order to generate a few points of interest like around a S-Box or around a key operation and then try to apply some attacks. We modelize the full system (smart-card) with all its buses and CPU in order to achieve a simulation closer to the one we could get through a real bench.

We also propose an implementation where software simulation is applied to fault based on a realistic fault model at every instruction. The effect on the program execution is then checked based on some heuristic rules in order to determine whether to save the mutant application or not for further analyze. That way we can proactively detect weak points in the software and fix them, or target more specifically some physical area or some time-frame in order to retrieve the same results that were obtained during a live analysis on a real product to understand how the fault happened and how to fix it. Software fault injection tries to measure the degree of confidence that one can have in a given system by evaluating what could happen when faults are injected. Traditionally, the software-based fault injection involves the modification of the software execution on the system under analysis in order to provide the capability to modify the system state according to the programmer model view of the system. All sorts of faults may be injected, from the register, flags, and memory faults.

Later enhancements of our simulator allow us to run Linux program and test them against faults and power leaks while the acquisition and signal processing required on such devices can be troublesome.

Such approach eases the validation of crypto library, JavaCard implementation or a whole OS. The complexity of the simulation increases with the number of instructions simulated but power is quite cheap nowadays, and cluster or Amazon EC2 like systems can be used in order to have a lot of CPU power available at a good price.

10.2 Power Simulation

Our first simulation engine was from a secure 16-bit microprocessor with proprietary language. It was the core powering some of Inside Secure chips while I was working there. We shall not be able to give much more details about the core besides it was a RISC microprocessor with 16-bits registers.

10.2.1 Core simulation

The main component of the simulator is represented by the simulation of the core itself. The algorithm looks like:

Alg. 10.1 CPU simulation

```
loadMemories()
setupPeripherals()
reset()
while true do
    ExecuteInstruction()
end while
finalize()
```

loadMemories() This function loads a specified ROM and NVM image into the core.

setupPeripherals() Initialize the different peripherals like UARTS, Crypto IPs, Timers, and all HW IPs in general around the microprocessor.

reset() Setup the CPU the way it is after going out of the loader. The loader ROM is not present in the simulator. The registers or memory for example are not encrypted while they are on the real product. It would be an enhancement for sure to implement that encryption/decryption mechanism to reproduce more closely the observed behaviors.

The main loop retrieves and executes each instruction. It is where most of the code of the simulator is present since all instructions have to be coded in detail, with flag and internal register management.

Memory Access

We simulate a read hardware bus, each read or write operations are caught and monitored. Peripherals register themselves to listen for read or write operation on the bus like real hardware does. This provides us with powerfull mechanism to monitor memory access or implement coprocessor.

Peripherals

The following peripherals were implemented in my first simulator:

- RF UART: this UART was responsible of simulating the radio-frequency transponder used in our product.
- Contact UART: this UART simulated the ISO-7816 interface present on some of our hardware, only the T=1 mode was implemented.
- DES coprocessor: we implemented also the coprocessor that was used for DES computations.
- RSA coprocessor: we also implemented some coprocessor that was used during some RSA computations on some products. Most of the code was the implementation of a multiplier.

Power Model

Different power models can be used during the simulation:

- Hamming Weight: We compute the sum of the Hamming Weights of all registers.
- Hamming Distance: We compute the sum of the Hamming Distance for each registers.
- Multi Points: For each instruction we compute multiple points, for example one point for Hamming weight and one point for the Hamming distance.

Noise can also be added to fit closer to real power measurements.

10.2.2 Enhancements

- Multi Architecture: We developed a nice working proof of concept while working for Inside Secure. After leaving the company we decided to rewrite from scratch a simulator. Using open source libraries we quickly built a simulator able to perform side channel analysis for different standard processor architecture like ARM, x86, Mips, Sparc, sh4...
- Linux Compatibility: We made the simulator compatible with standard Linux program. You can now for example use a program already compiled for a supported architecture and get nice power traces out of the box. No special work is required for the compilation. It is important to note though that kernel calls are not traced by the current implementation so you only get power consumption for the user space.

- Automatized Tests: Software crypto libraries providing the good interfaces can now be tested automatically by the tool on a first step. It is only preliminary work for an evaluator but gives surprisingly good results close to the target.

10.2.3 Usage

There are two different mode for the simulator. The first one is full simulation where each device has to be implemented and a full ROM loaded in the simulator, the second one is the Linux simulation where the host kernel is used (assuming we are running on a Linux host) to replace the guest OS.

While in the first case all hardware connected and used in the ROM have to be implemented in order to simulate either power or faults, in the second case we can simply compile a standard Linux program and then execute it as if we were executing it directly by passing him arguments. The simulator will take care of loading shared librairies and simulating kernel calls. The easiest way to setup a simulation environment if other architecture for host and guest are used is simply to compile the program with $-static$ gcc option and run it through our simulator.

A typical usage for a AES simulation for sample would be:

```
# gcc -static aes.c -o aes
# ./aes <plaintext> <key>
> <ciphertext>
# sim -p x86 -curve aes.cb ./aes <plaintext> <key>
> <ciphertext>
# display aes.cb
```

10.2.4 Showcase

In this section we will show pin program, AES, DES and RSA computation obtained for different hardware architecture.

Pin

This program corresponds to an insecure Pin verification.

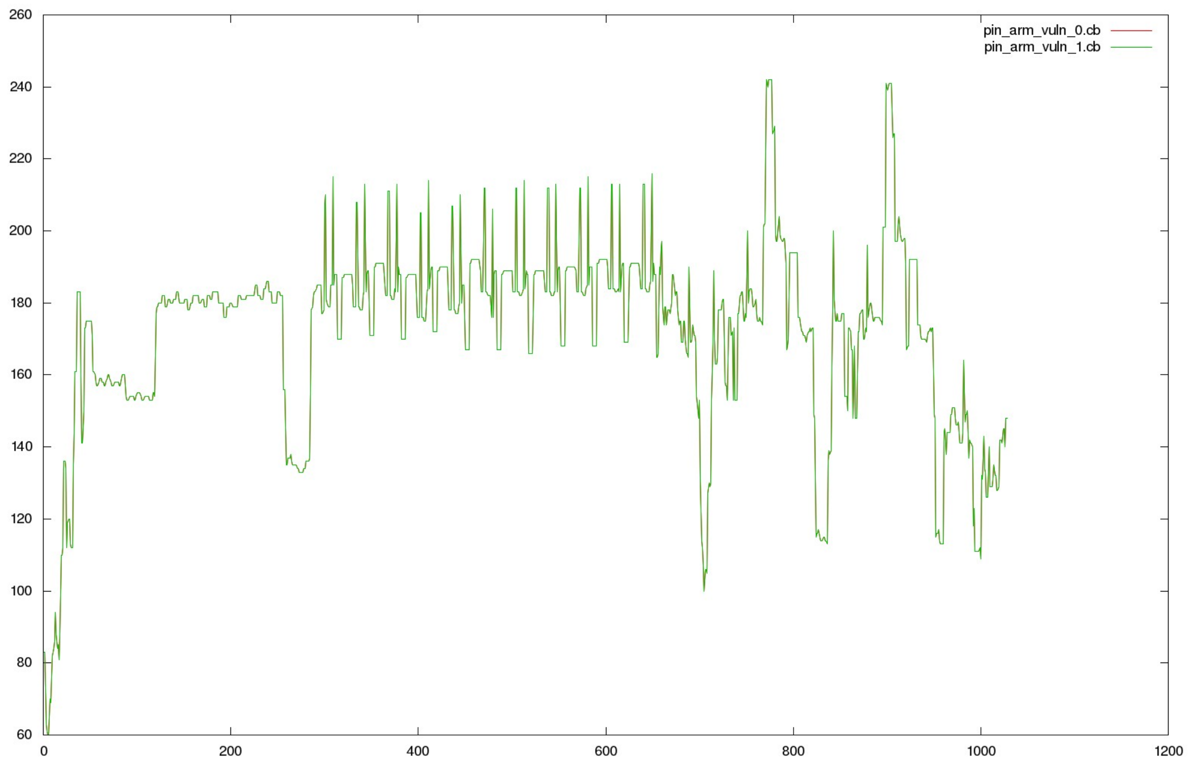


Figure 10.1: Simulated Pin on an ARM architecture

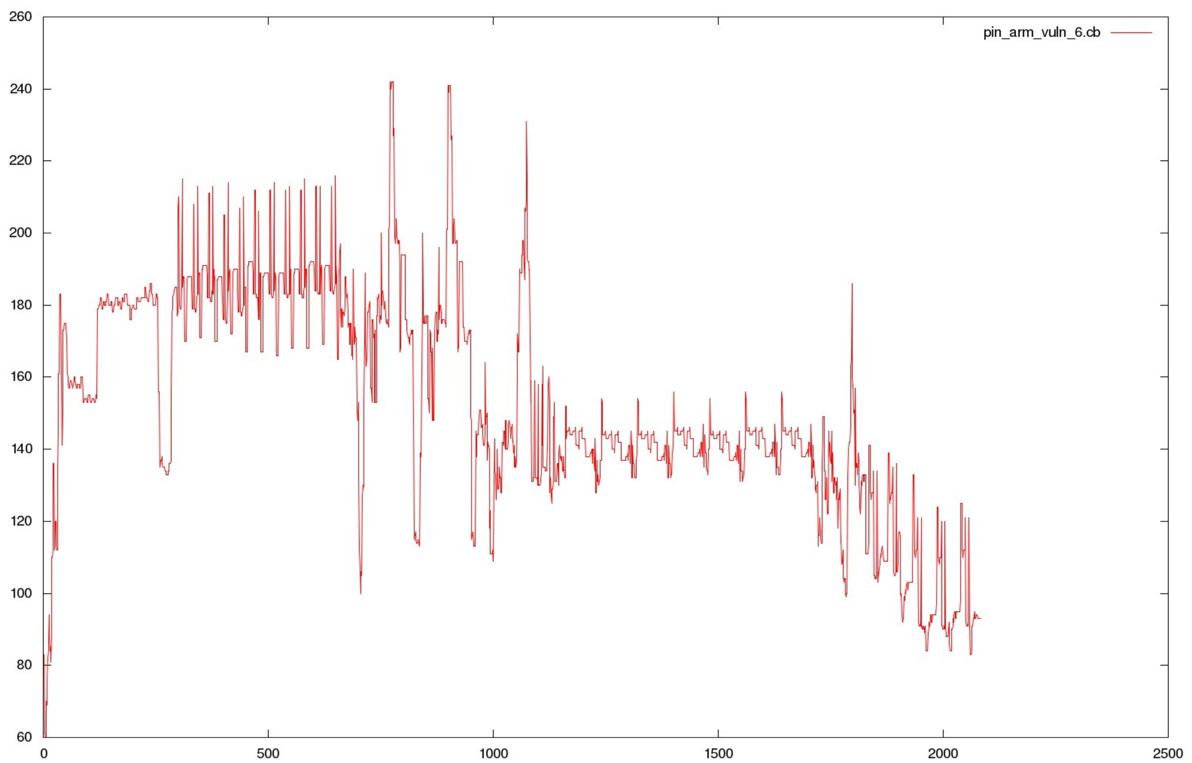


Figure 10.2: Simulated Pin on a x86_64 architecture

AES

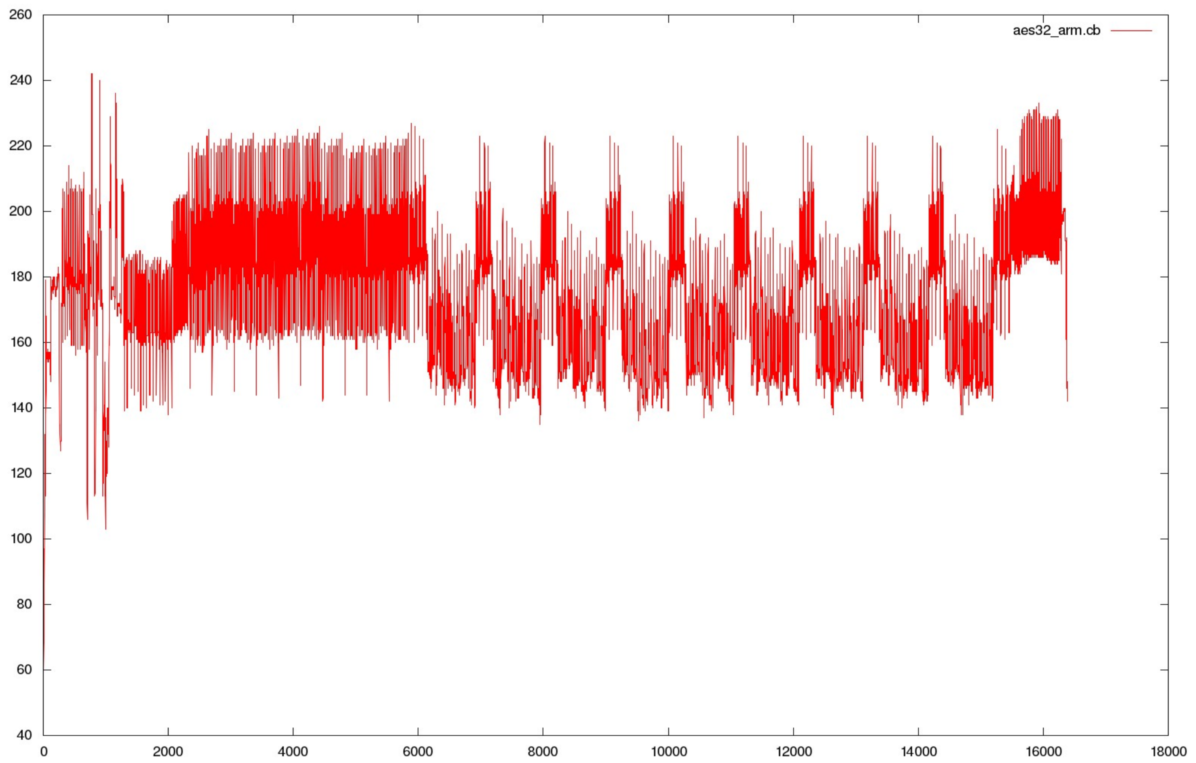


Figure 10.3: Simulated AES on an ARM architecture

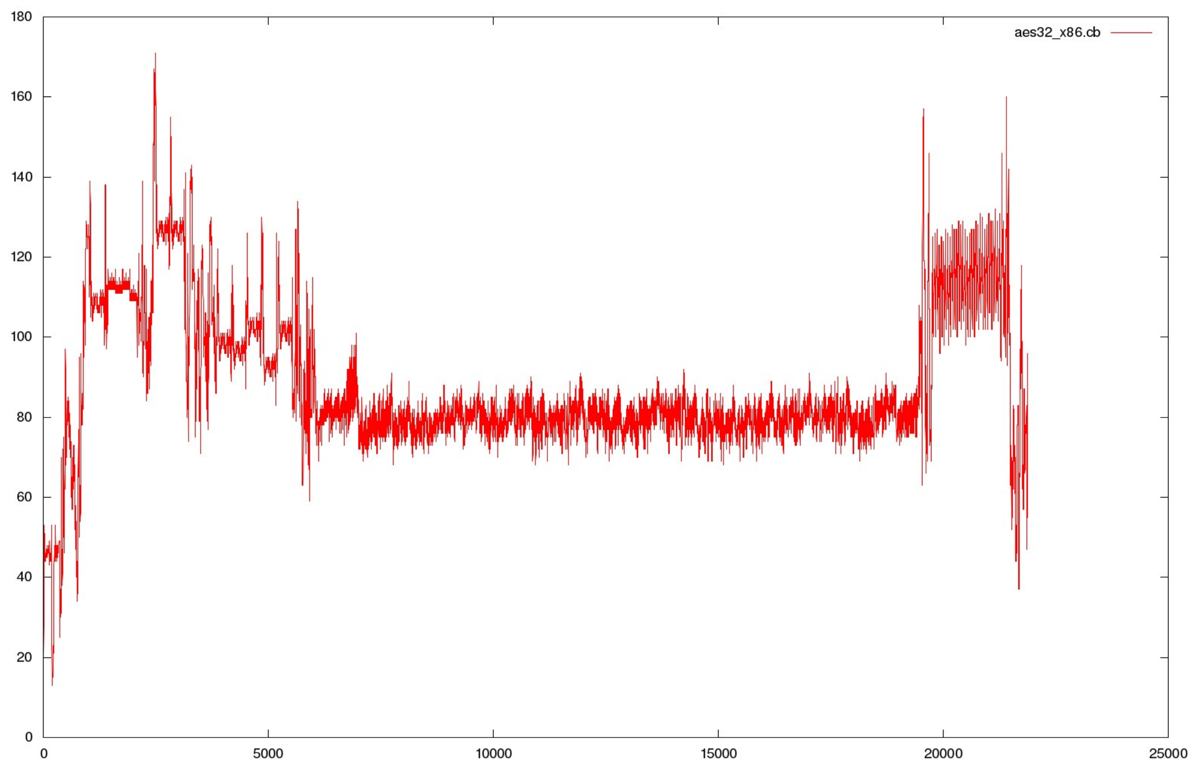


Figure 10.4: Simulated AES on a x86_64 architecture

DES

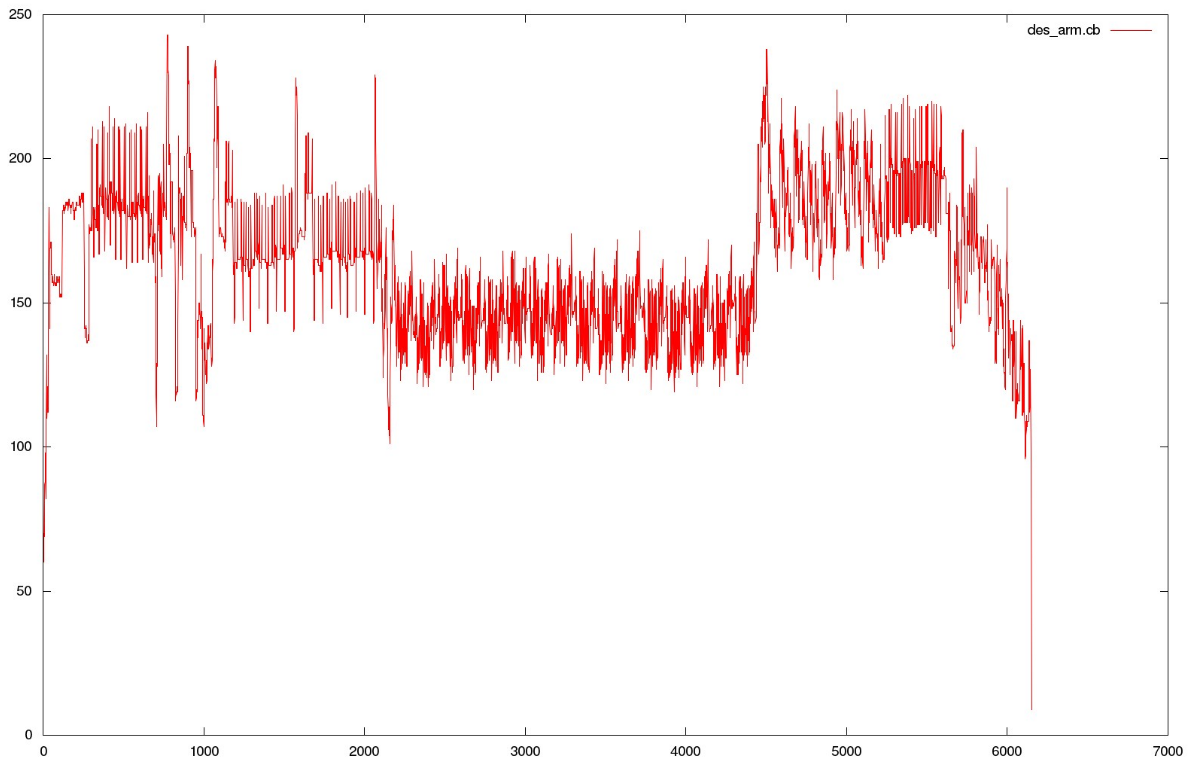


Figure 10.5: Simulated DES on an ARM architecture

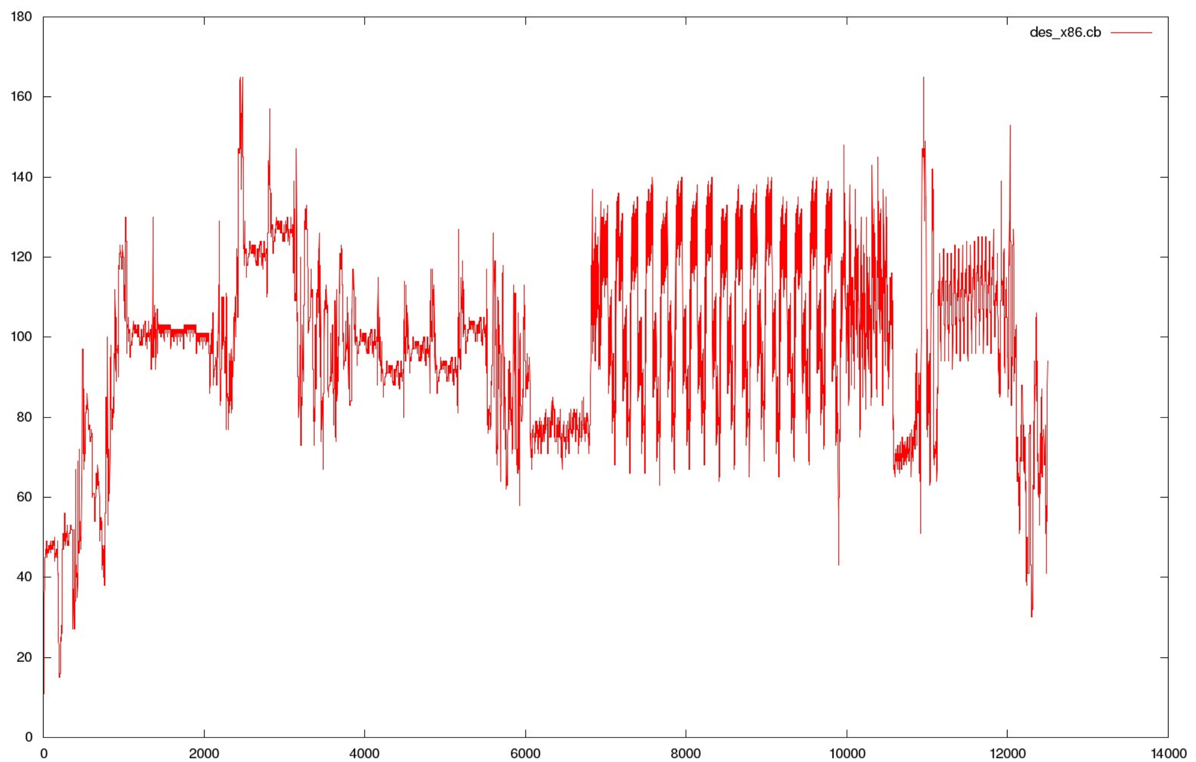


Figure 10.6: Simulated DES on a x86_64 architecture

RSA

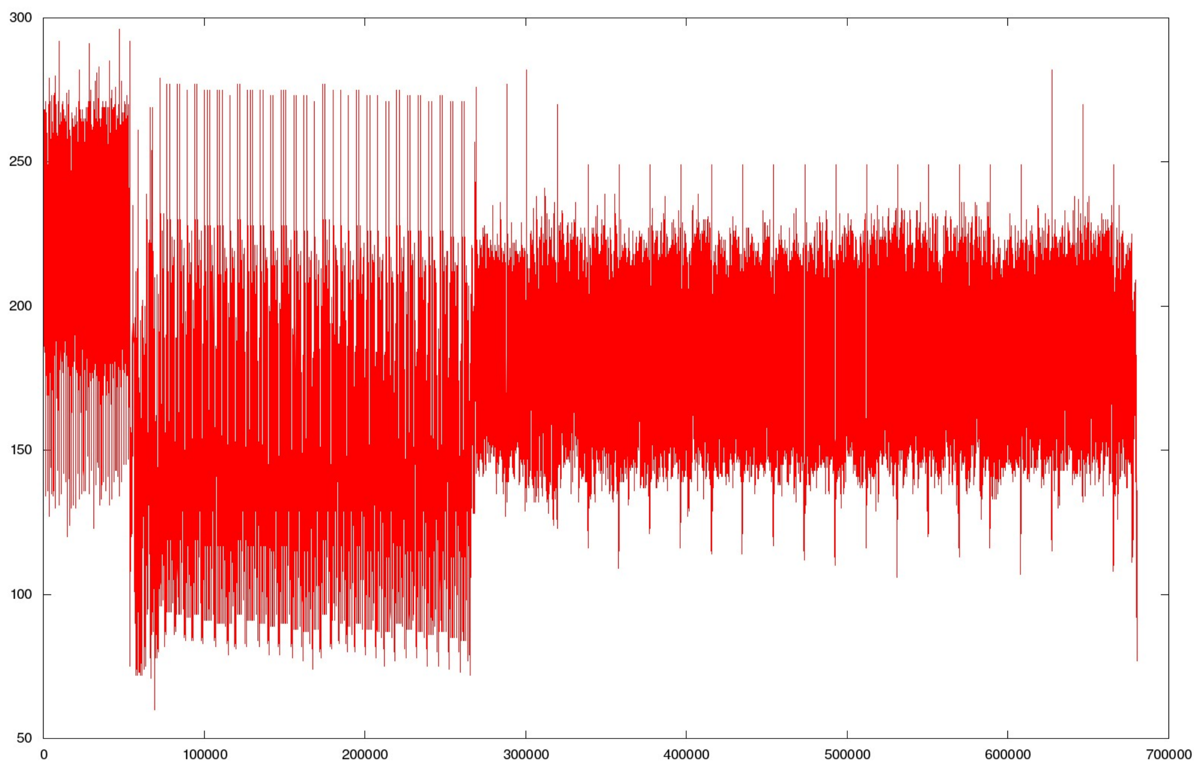


Figure 10.7: Simulated RSA on an ARM architecture

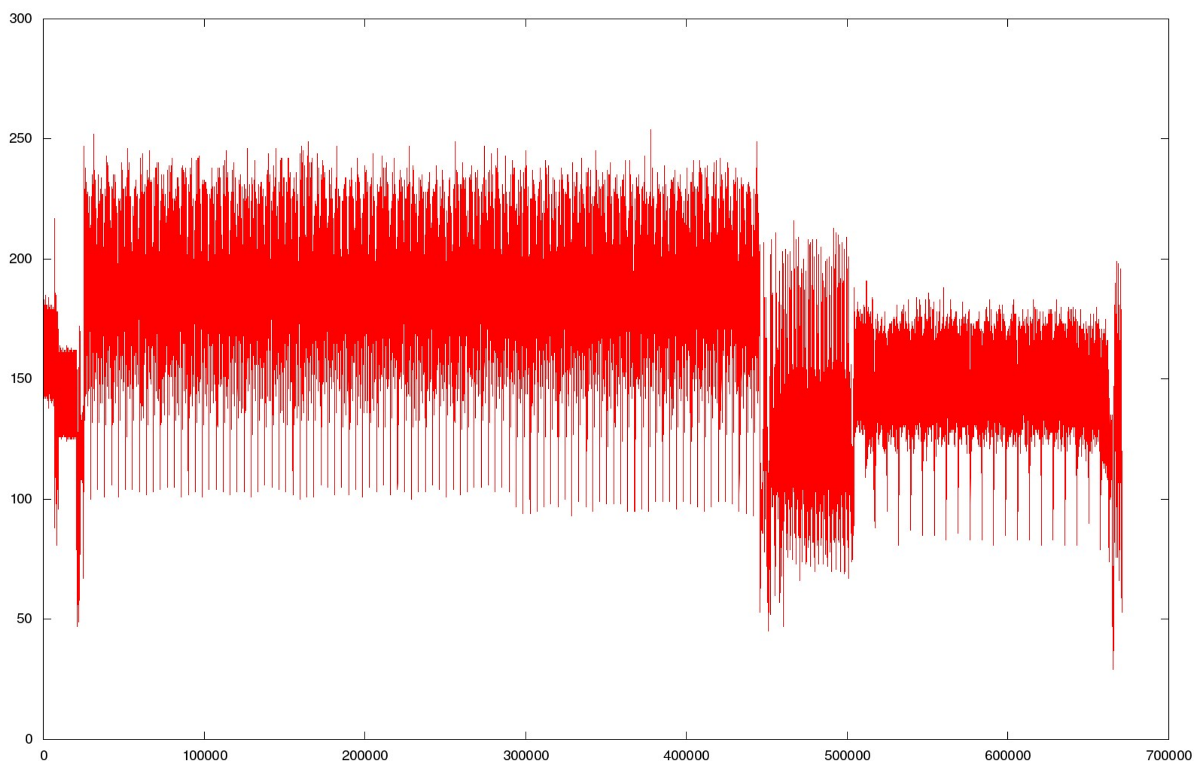


Figure 10.8: Simulated RSA on a x86_64 architecture

Attacks

The power simulator is a great tool to test new attacks on a device closer to the real one than usual power simulation. Usually one would generate a few points through Python or Matlab or any other tool and then assert if the attack is working or not only on those few points, biasing the attack since the simulation is too much time narrowed. Since we simulate the whole transaction, we can also take into account the usual counter measures and implement them in the simulator in order to be more realistic. Such counter measures like noise, random cycle, ... can then be added to the simulator.

In this part we will show some attacks results

DPA The following figures present a classical DPA attack on the output of the S-Box of an AES. We can see a good pic and a ghost one, the good guess is the highest one though.

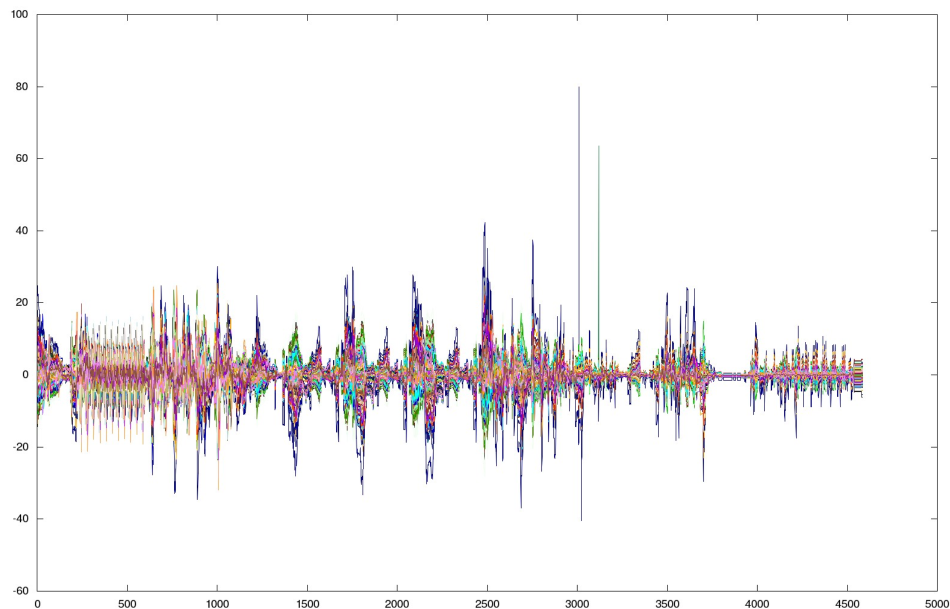


Figure 10.9: DPA simulation sample

CPA The following figures present classical CPA on the output of S-Box in the case of AES. The first figure shows all the 256 curves displayed at once, each one with a different color, while the second one shows all wrong guess displayed in black with the good guess displayed in red.

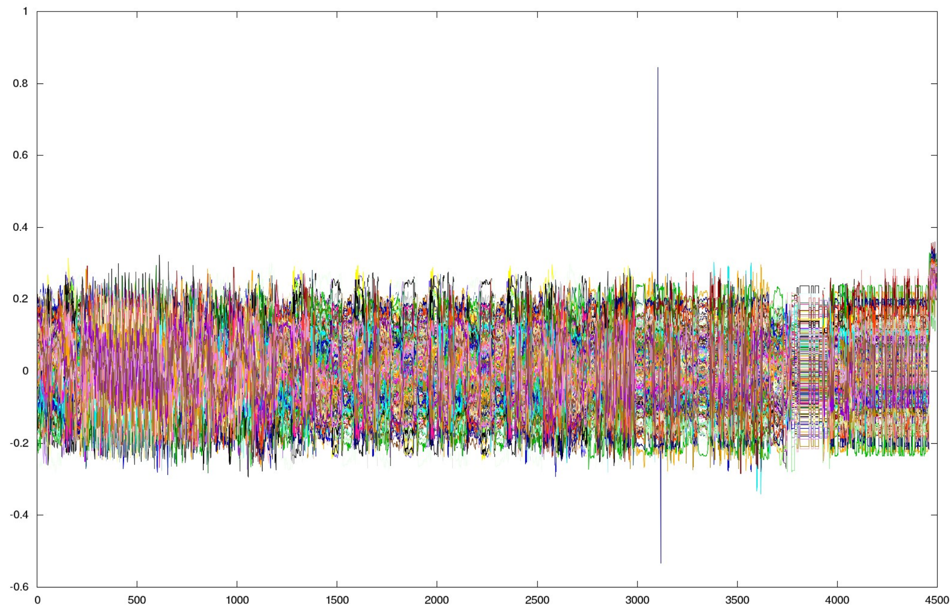


Figure 10.10: CPA simulation sample

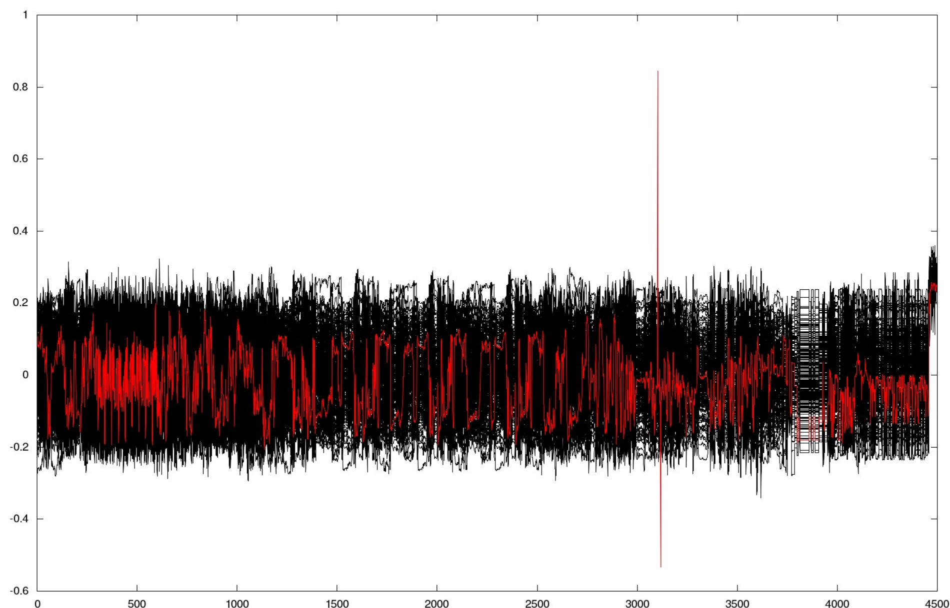


Figure 10.11: CPA simulation sample with good guess highlighted

Cross-Correlation such attacks were also tested through the simulator like in Figure 6.3 and Figure 6.4.

10.3 Fault Simulation

Fault attacks are pretty hard to protect against since their effect has almost always a random part, and it is hard to understand what the fault actually did even if we can know from a time curve where it occurred and can try to deduce where in the

code it happened. Using the same approach as previously we developed a fault simulator that allowed us to test a good proportion of the first order faults on our system. The simulator reuses the infrastructure used by the power simulator in order to simulate the core and its peripheral. Let C be the compiled code of either the ROM of the Linux program. Let T be the transaction we want to check against faults. The transaction can be simply the execution of a Linux program in case of a Linux one, or the processing of the core between two commands like between the entrance of a card in a field and its answer to a GenerateAC command. Let H be a set of state that can qualify the transaction as halted, for sample an infinite loop, being stuck in a OS protection or simply exiting in the case of a Linux process. Let S be a set of faults we want to apply. We chose that set of faults from different elements:

- CPU registers
- CPU flags
- CPU memory bus
- Memory Cell (Absolute Address)
- Memory Cell (Relative Address) Stack Cells for sample were relative to the stack register

Let I a set of informations saved used to check if the fault was successful or not, for example an element from I element could be the fact that every golden values in the OS are correct, another one could be the fact that the stack stay coherent or that the command returned by the card match the one from the nominal transaction (without fault). The simulator executes each instruction of C required to produce T, at each instruction it saves the current state, inserts a fault and then resumes until a point of halt then check if the fault had any effects. It then restores the state, tries to execute another fault. When no more fault are available it goes to the next instruction and starts again its fault until the execution of the last instruction.

The fault algorithm we use is the following one:

Alg. 10.2 Fault Algorithm

Input: C,S,T,H,I

Execute every instructions of C to have the behavior T and save I_t at different instants.
while $currentstate \notin H$ **do**
  Save $state$
  for all $Fault \in S$ **do**
    Apply S_i
    while $currentstate \notin H$ **do**
      Execute the next instruction
      save $I_{i,t}$
    end while
    if $(I_i \neq I)$ **then**
      Log the fault and the result
    end if
    Restore $state$
  end for
  Execute Instruction
end while

The simulator has to be fast since each instruction will be executed many times. For a nominal transaction of N instructions with S the number of faults simulated at each instruction, we would have N instructions faulted. Each fault requires the simulation to run until the end of the processing, so on the first instruction of the simulation of a fault we assume that a fault would require on average N more instructions to be executed until we hit a state $\in H$, on the second instruction $N-1$ and so on. By recurrence we could then prove than on average the number of faults executed would be around

$$TI(S,N) = S * \frac{N*(N+1)}{2} \tag{10.1}$$

Let a host instruction be defined as the native instruction understood by the hardware on which the simulation is done (a typical PC for example) and target the instruction that the smard-card understand. For 100 faults on a transaction of 100000 instructions this gives $5*10^{11}$ simulated instructions executed. Let us say that a standard target instruction is simulated with 100 host instructions, that gives $5*10^{13}$ host instructions to execute. A processor at 4 GHz computes $4*10^9$ operations per second meaning that we need at least 4 hours to execute the simulation. This is a very rough estimation, since the 4 GHz instructions per second assume no cache miss... But the point is that all the functions have to be thought with efficiency in mind. Lots of profiling need to be done too in order to improve the most critical code path.

10.3.1 Results

The log file of every faulty $I_{i,t}$ has then to be reviewed in order to understand if the fault is dangerous or not. Different scripts are then being used on the Raw Logs in order to extract meaningful information for the attackers. We used for example a script that allowed us to find ROM dumps from raw log since it can yield to a failure

to pass the security test depending on the evaluation targeted. The original log simply displayed the faults and the faulty ciphertext generated by the processing of the message. The algorithm used was quite simple:

Alg. 10.3 Find ROM Dump

Input: A log file L,C,The minimum size of considered ROM dump k

Compute the hashmap H of ROM image for all block of size k
for all inputs I in L **do**
  for all blocks of size $k \in I$ named B **do**
    if $B \in H$ **then**
      Echo the ROM dump found
    end if
  end for
end for

For every contiguous sequence of ROM S of bytes of size k, we computed the hash h of S and stored it in H. Then we looked up on every contiguous sequence of k bytes present in the dumps, we computed its hash and checked if it was present in H.

The resulting log file were looking like:

```
[At: @offset1] Found [DUMP1]
[At: @offset2] Found [DUMP2]
[At: @offset3] Found [DUMP3]
[At: @offset4] Found [DUMP4]
```

The information obtained there was really important for us. We managed for example to find through simulation some ROM dumps that happened during an evaluation. From that information we went back to the FAULT-ID that caused it. The simulator then generates a script file to load in our development tool in order to give us a debugger at the exact point where the fault happened with the context already modified to include the fault.

Standard debugging method then allowed us to track back what really happened and how to fix it. Once the patch was available a new cycle of simulation was then launched in order to validate it and prevent another potential flaw that could have been introduced. The product were then tested again on the laser bench and got a green light!

10.3.2 Showcase

By mixing at the same time the fault and the consumption simulator, one can get the power trace corresponding to the execution of the faulted transaction like the ones depicted below. It can be only for information like in the first PIN case, or can be useful to mount other attacks like in chapter 8.

PIN Fault Simulation

We present some quick results on the fault simulation of the PIN verification. We run the nominal case program with 2 good digits on 4 the program returns KO.

```
# sim -p arm bin/pin_arm 2
> KO
```

Then we execute the fault simulator and we quickly find that some fault do manages to return OK.

```
# sim -p arm -faults bin/pin_arm 2
> [0:8024:r1:0]
> KO
> [0:8024:r2:0]
> KO
> ...
> [1:8028:r1:0]
> KO
> ...
> [1511:830c:r3:0]
> OK
> ...
```

We can now look inside the binary source in order to understand what happened.

```
# arm_objdump -D bin/pin_arm | grep -B 2 -A 1 -i 830c
> 8304: eb000559 bl 9870 <__GI_strlen>
> 8308: e1a03000 mov r3, r0
> 830c: e1540003 cmp r4, r3
> 8310: 3affffea bcc 82c0 <pin_verification_vuln+0x20>
```

```c
/* Extract from pin.c program */
int pin_verification_vuln(const char *value,const char *secret)
{
        int i;
        for(i=0;i<strlen(secret);i++){
                if(value[i] != secret[i] ){
                        return 1;
                }
        }
        return 0;
}
```

In fact at $0x830C$ the register $r3$ contains the length of the secret stored in the card, by setting it to zero we ensure that both strings match and we get the OK results! In the following figure, we can see that 3 bytes comparison are done in the KO case, each comparison highlighted by a gray band, while in the faulted case leading to OK no comparison are done.

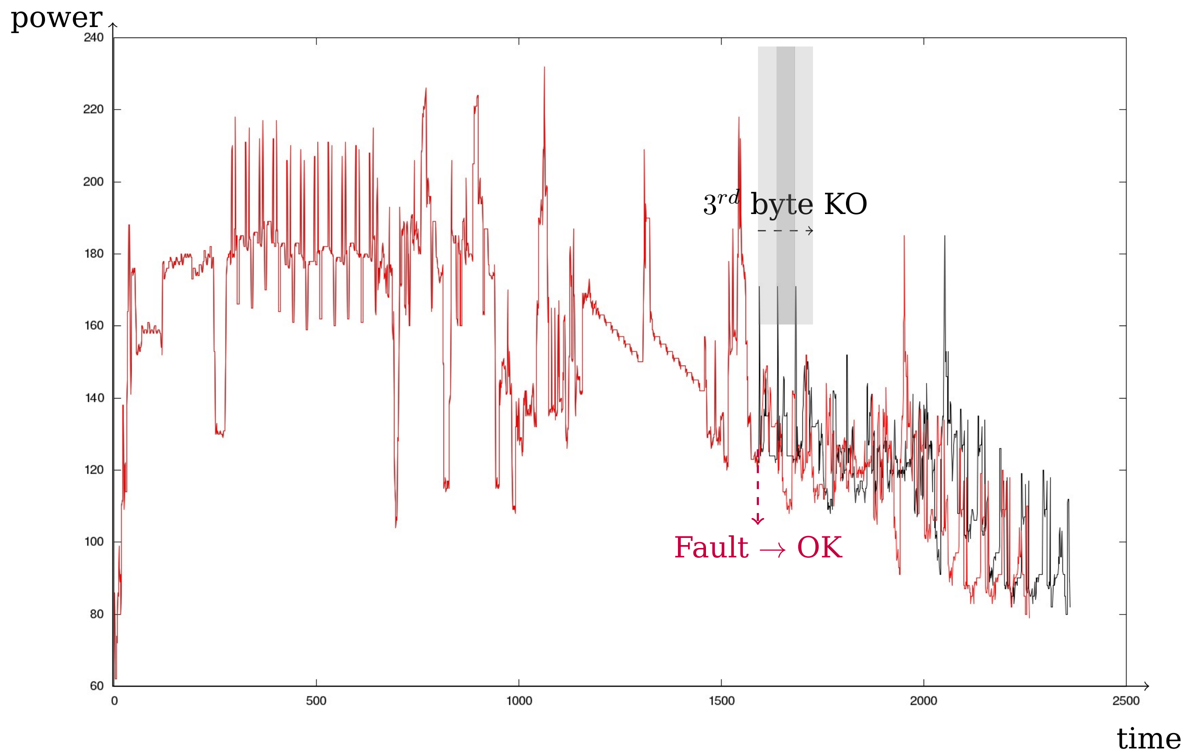


Figure 10.12: Pin Fault Simulation

AES Fault Simulation

We can see another example where we skip some rounds of AES computation. We could do the same things we did for the PIN in order to retrieve the code that lead us not doing the full computation. On this fault simulation, only the last round is executed all the others are skipped. In that case the attack proposed in [25] for sample could be applied to break the key.

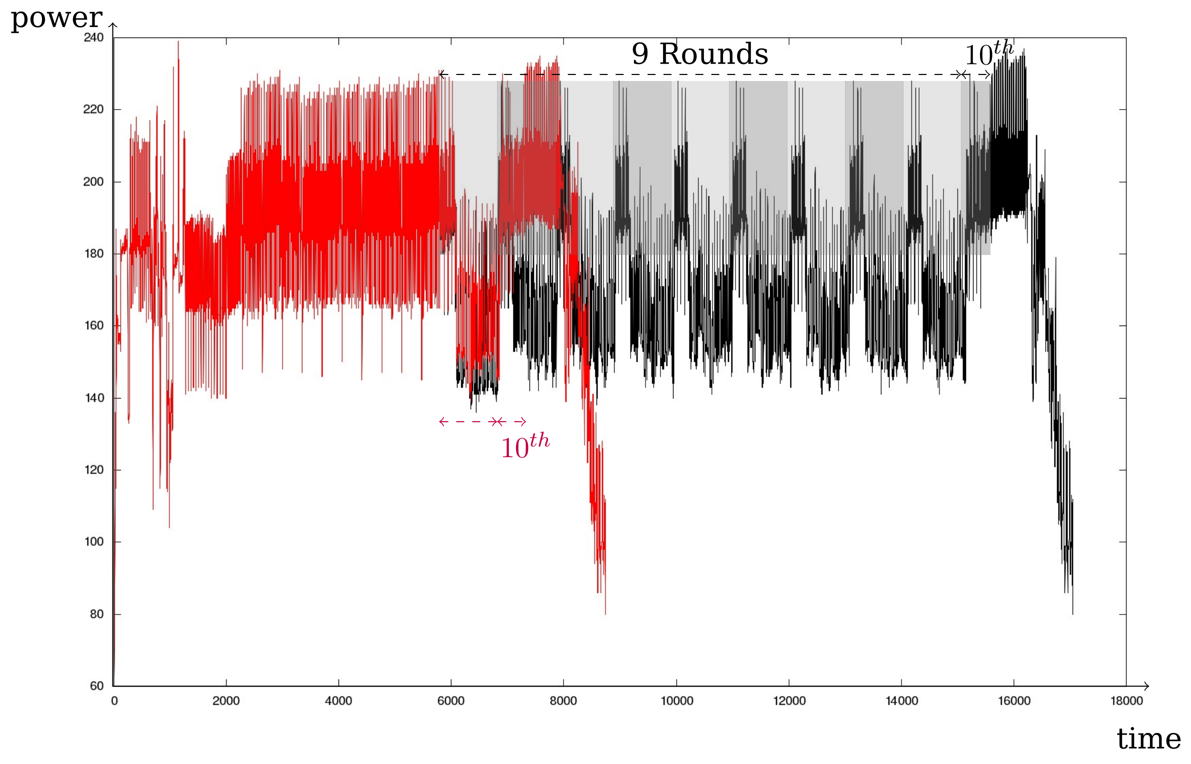


Figure 10.13: AES Fault Simulation

Conclusion

In the first part of this thesis, we provide an overview of the state of the art on some of the attacks actually used on embedded devices like smartcards. We reviewed the Side Channel Analysis and Faults Attacks and presented the classical way to protect against those threats.

We then presented our contributions to the field:

In chapter 5 we presented a way to apply classical power analysis techniques such as CPA on a single curve to recover the secret key in some public key implementations – e.g. non CRT RSA, DSA or Diffie-Hellman – protected or not by exponent randomization. We also applied our technique in practice and presented some successful results obtained on a 16-bit RISC microprocessor. However even with bigger multiplier sizes (32 or 64 bits) this attack can be successful depending on the key size, cf. subsection 5.4.1. We discussed the resistance of some countermeasures to our analysis and introduced three secure multiplication algorithms.

Our contribution enforces the necessity of using sufficiently large random numbers for blinding in secure implementations and highlights the fact that increasing the key lengths in the next years could improve the efficiency of some side-channel attacks. The attack we presented threatens implementations which may have been considered secure up to now. This new potential risk should then be taken into account when developing embedded products.

Further work could target the use of other values and distinguishers for the horizontal correlation analysis and then improve its efficiency. Possible ideas include: using more intermediate values, some likelihood tests, guessing simultaneously many bits of the secret exponent to increase the number of available curves for the analysis, using different models like the bivariate one for correlation factor computation on curves.

In chapter 6 we presented a new collision-correlation analysis method on first-order secured AES implementations. We highlighted the fact that this kind of attack is more powerful and practicable than previous second-order power analyzes, and increases the risk of these implementations being broken in practice. This confirms the necessity for developers to take into account how collisions of masked data may be unsafe in cryptographic implementations. A possible countermeasure could be the use of second (or higher) order resistant schemes.

Though we presented practical results on software implementations, we believe that this technique may also be a threat for hardware coprocessors. Therefore the collision-correlation threat should be taken into consideration

by developers and designers during their embedded cryptographic design.

In chapter 7 we presented new side-channel methods — the Big Mac using collision correlation and the two Rosetta techniques — allowing to distinguish a squaring from a multiplication when the same long-integer multiplication algorithm is used for both operations. They can be used to recover an RSA secret exponent — both in standard or CRT mode — with a single execution side-channel trace. We compare our new techniques with other single trace side-channel analyzes and demonstrate that they are more efficient than previous ones, especially on noisy measurements. We show that classical combination of message, modulus and exponent blindings is not sufficient to counteract our analysis and we suggest more advanced countermeasures. As a conclusion, we quote Colin Walter to recall the very interesting property of these attacks: "The longer the key length, the easier the attacks.".

In chapter 8 we have shown that even if sound countermeasures are known for protecting embedded cryptographic implementations from either high order side-channel analysis or differential and collision fault analysis, simply putting them together may not be sufficient. We have presented a Combined Active – CFA – and Passive – CPA – Attack (PACA) which breaks a proposed state of the art side-channel resistant AES implementation with a limited number of faults. We have enumerated some possible countermeasures, but remark that a safe errors variant of our attack can defeat most of them, such as executing and comparing twice a HODPA resistant implementation – though it requires a significant number of fault injections and a highly reliable fault injection tool. Although we have given some hints about how our last attack may be rendered more difficult, it seems to be an open problem how to protect implementations from ineffective faults, which are informative even though they do not alter the computations.

In chapter 9 we have shown that trading multiplications for squarings in an exponentiation scheme together with the atomicity principle provides a new countermeasure against side-channel attacks aimed at distinguishing squarings from multiplications. Moreover, this countermeasure is intrinsically more secure against such analysis than the classical multiply always atomic algorithm with exponent blinding, and provides better performances and flexibility towards space/time trade-offs than regular algorithms such as the Montgomery ladder or the square-and-multiply always.

As a complementary work, we present new algorithms using two parallel squaring blocks, and show how to write them atomically. We point out that, as far as we know, it leads to the fastest results in terms of speed. On the hardware side, an interesting conclusion is that two parallel squaring blocks enable faster exponentiation algorithms than two parallel multiplication blocks. We believe that these observations are of great interest for the embedded devices industry and for everyone looking for fast exponentiation.

In chapter 10 we devised and implemented a SCA and fault simulator able to provide to developers or evaluators tools to asset the resistance of a specified code before even putting it on any hardware.

Our Patents

[1] Microprocessor protected against memory dump
Benoit Feix and Georges Gagnerot.
Publication date: Feb 28, 2013.
Publication numbers: US20130055025 , CN102968392 , EP2565810A1.

[2] Integrated circuit protected against horizontal side channel analysis
Benoit Feix, Georges Gagnerot, Mylene Roussellet, Vincent Verneuil
Publication date: Oct 6, 2011.
Publication number: US20110246789.

[3] Process for testing the resistance of an integrated circuit to a side channel analysis
Benoit Feix, Georges Gagnerot, Mylene Roussellet, Vincent Verneuil
Publication date: Oct 6, 2011.
Publication number: US20110246119.

[4] Encryption method comprising an exponentiation operation Benoit Feix, Georges Gagnerot, Mylene Roussellet, Vincent Verneuil, Christophe Clavier
Publication date: Aug 30, 2012.
Publication number: US20120221618,CN102684876A,EP2492804A1.

[5] PROCEDE DE FOURNITURE D'UN SERVICE SECURISE
Gary Chew, Georges Gagnerot.
Publication request date: Nov 26, 2012.
Publication request: I190.

[6] METHOD OF ACQUIRING BIOLOGICAL INFORMATION BY MEANS OF AN INTRA BODY CURRENT
Bruno Charrat, Georges Gagnerot
Publication request: 100791FR.

Our Publications

[1] Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, and Vincent Verneuil. Rosetta for single trace analysis. In *Progress in Cryptology-INDOCRYPT 2012*, pages 140–155. Springer, 2012.

[2] Christophe Clavier, Benoit Feix, Georges Gagnerot, and Mylene Roussellet. Passive and active combined attacks on aes combining fault attacks and side channel analysis. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop* on, pages 10–19. IEEE, 2010.

[3] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In *Information and Communications Security*, pages 46–61. Springer, 2010.

[4] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Improved collision-correlation power analysis on first order protected aes. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 49–62. Springer, 2011.

[5] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Square always exponentiation. In *Progress in Cryptology–INDOCRYPT 2011*, pages 40–57. Springer, 2011.

Bibliography

[1] M.-L. Akkar, R. Bevan, and L. Goubin. Two Power Analysis Attacks against One-Mask Methods. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2004.

[2] 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, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

[3] Ali Alaeldine, Richard Perdriau, Mohamed Ramdani, J-L Levant, and M'hamed Drissi. A direct power injection model for immunity prediction in integrated circuits. *Electromagnetic Compatibility, IEEE Transactions on*, 50(1):52–62, 2008.

[4] F. Amiel, C. Clavier, and M. Tunstall. Fault Analysis of DPA-Resistant Algorithms. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *FDTC*, volume 4236 of *Lecture Notes in Computer Science*, pages 223–236. Springer, 2006.

[5] F. Amiel and B. Feix. On the BRIP Algorithms Security for RSA. In J. A. Onieva, D. Sauveron, S. Chaumette, D. Gollmann, and C. Markantonakis, editors, *Information Security Theory and Practices. Smart Devices, Convergence and Next Generation Networks - WISTP 2008*, volume 5019 of *Lecture Notes in Computer Science*. Springer, 2008.

[6] F. Amiel, B. Feix, L. Marcel, and K. Villegas. Passive and Active Combined Attacks. In *Workshop on Fault Detection and Tolerance in Cryptography - FDTC 2007*. IEEE Computer Society, 2007.

[7] F. Amiel, B. Feix, M. Tunstall, C. Whelan, and W. P. Marnane. Distinguishing Multiplications from Squaring Operations. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 2008.

[8] F. Amiel, B. Feix, and K. Villegas. Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. In *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2007.

[9] F. Amiel, B. Feix, and K. Villegas. Power analysis for secret recovering and reverse engineering of public key algorithms. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2007.

[10] R-M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Verkauteren. Handbook of Elliptic and Hyperelliptic Curve Cryptography, 2006.

[11] P. 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, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1986.

[12] Pierre Bayon, Lilian Bossuet, Alain Aubert, Viktor Fischer, François Poucheret, Bruno Robisson, and Philippe Maurine. Contactless electromagnetic active attack on ring oscillator based true random number generator. In *Constructive Side-Channel Analysis and Secure Design*, pages 151–166. Springer, 2012.

[13] Daniel J Bernstein. Cache-timing attacks on aes, 2005.

[14] R. Bevan and E. Knudsen. Ways to Enhance Differential Power Analysis. In P. J. Lee and C. H. Lim, editors, *ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2003.

[15] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In B. S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[16] J. Blömer and J.-P. Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In Rebecca N. Wright, editor, *Financial Cryptography, 7th International Conference, 2003*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.

[17] Andrey Bogdanov. Improved Side-Channel Collision Attacks on AES. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2007.

[18] Andrey Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. In Oswald and Rohatgi [80], pages 30–44.

[19] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *EUROCRYPT*, pages 37–51, 1997.

[20] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Joye and Quisquater [57], pages 16–29.

[21] D. Canright and L. Batina. A Very Compact "Perfectly Masked" S-Box for AES. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 446–459, 2008.

[22] Çetin Kaya Koç. Analysis of sliding window techniques for exponentiation. *Computers and Mathematics with Applications*, 30(10):17–24, 1995.

[23] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004.

[24] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *Computers, IEEE Transactions on*, 53(6):760–768, 2004.

[25] Hamid Choukri and Michael Tunstall. Round reduction using faults. *FDTC*, 5:13–24, 2005.

[26] Christophe Clavier. An Improved SCARE Cryptanalysis Against a Secret A3/A8 GSM Algorithm. In *International Conference on Information Systems Security – ICISS'07*, pages 143–155, 2007.

[27] Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, and Vincent Verneuil. Rosetta for single trace analysis. In *Progress in Cryptology-INDOCRYPT 2012*, pages 140–155. Springer, 2012.

[28] Christophe Clavier, Benoit Feix, Georges Gagnerot, and Mylene Roussellet. Passive and active combined attacks on aes combining fault attacks and side channel analysis. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop on*, pages 10–19. IEEE, 2010.

[29] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In *Information and Communications Security*, pages 46–61. Springer, 2010.

[30] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Soriano et al. [96], pages 46–61.

[31] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Improved collision-correlation power analysis on first order protected aes. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 49–62. Springer, 2011.

[32] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Square always exponentiation. In *Progress in Cryptology–INDOCRYPT 2011*, pages 40–57. Springer, 2011.

[33] P. G. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Syst. J.*, 29(4):526–538, 1990.

[34] J-S Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.

[35] J-F. Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards.* PhD thesis, Université catholique de Louvain, Louvain, 1998.

[36] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[37] P. Dusart, G. Letourneux, and O. Vivolo. Differential Fault Analysis on A.E.S. In J. Zhou, M. Yung, and Y. Han, editors, *Applied Cryptography and Network Security, First International Conference, ACNS 2003*, volume 2846 of *Lecture Notes in Computer Science*. Springer, 2003.

[38] Horst Feistel. Cryptography and computer privacy. 228(5):15–23, May 1973.

[39] Federal Information Processing Standards Publication (FIPS). Data Encryption Standard - DES, FIPS PUB 46-3, 1999.

[40] Federal Information Processing Standards Publication (FIPS). Advanced Encryption Standard - AES, FIPS PUB 197, 2001.

[41] FIPS PUB 186-3. *Digital Signature Standard*. National Institute of Standards and Technology, october 2009.

[42] P-A. Fouque and F. Valette. The Doubling Attack - *why upwards is better than downwards*. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2003.

[43] Karine Gandolfi, D. Naccache, C. Paar, Karine G, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results, 2001.

[44] Harvey L Garner. The residue number system. *Electronic Computers, IRE Transactions on*, (2):140–147, 1959.

[45] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual Information Analysis. In Oswald and Rohatgi [80], pages 426–442.

[46] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems–CHES 2008*, pages 426–442. Springer, 2008.

[47] C. Giraud and H. Thiebeauld. A Survey on Fault Attacks. In J.-J. Quisquater, P. Paradinas, Y. Deswarte, and Anas Abou El Kalam, editors, *Smart Card Research and Advanced Applications VI, CARDIS 2004*, pages 159–176. Kluwer, 2004.

[48] Christophe Giraud. DFA on AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Advanced Encryption Standard, 4th International Conference, AES 2004*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[49] J. Dj. Golic and C. Tymen. Multiplicative Masking and Power Analysis of AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.

[50] Sudhakar Govindavajhala and Andrew W Appel. Using memory errors to attack a virtual machine. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 154–165. IEEE, 2003.

[51] D. Hankerson, A.J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing Series, January 2003.

[52] L. Hemme. A Differential Fault Attack Against Early Rounds of (Triple-)DES. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 254–267. Springer, 2004.

[53] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir. Collision-Based Power Analysis of Modular Exponentiation Using Chosen-Message Pairs.

volume 5154 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2008.

[54] Jakob Jonsson and Burt Kaliski. Public-key cryptography standards (pkcs)# 1: Rsa cryptography specifications version 2.1. 2003.

[55] M. Joye and S-M. Yen. The Montgomery Powering Ladder. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 291–302, 2002.

[56] Marc Joye. Highly regular m-ary powering ladders. In *Selected Areas in Cryptography*, pages 350–363. Springer, 2009.

[57] Marc Joye and Jean-Jacques Quisquater, editors. *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*. Springer, 2004.

[58] A. A. Karatsuba and Y. P. Ofman. Multiplication of multidigit numbers on automata. *Doklady Akademii Nauk SSSR*, 45(2):293294, 1962.

[59] P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks. 1998.

[60] P. 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. Springer, 1996.

[61] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO ’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[62] P.C. Kocher, J.M. Jaffe, and B.C. June. DES and Other Cryptographic Processes with Leak Minimization for Smartcards and other CryptoSystems. *US Patent 6,278,783*, 1998.

[63] Oliver Kömmerling and Markus G Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, pages 2–2. USENIX Association, 1999.

[64] Joseph Lee Rodgers and W Alan Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[65] S. Mangard, E. Oswald, and T Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[66] S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[67] Stefan Mangard and François-Xavier Standaert, editors. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*. Springer, 2010.

[68] Mitsuru Matsui. The first experimental cryptanalysis of the data encryption standard. In *Advances in Cryptology—Crypto'94*, pages 1–11. Springer, 1994.

[69] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[70] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999.

[71] Thomas S Messerges. Using second-order power analysis to attack dpa resistant software. In *Cryptographic Hardware and Embedded Systems—CHES 2000*, pages 238–251. Springer, 2000.

[72] Thomas S Messerges. Using second-order power analysis to attack dpa resistant software. In *Cryptographic Hardware and Embedded Systems-CHES 2000*, pages 238–251. Springer, 2000.

[73] Thomas S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2001.

[74] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of power analysis attacks on smartcards. In *In USENIX Workshop on Smartcard Technology*, pages 151–162, 1999.

[75] P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *MC*, 48:243–264, 1987.

[76] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[77] P.L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation, 44(170)*, pages 519–521, April 1985.

[78] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-Enhanced Power Analysis Collision Attack. In Mangard and Standaert [67], pages 125–139.

[79] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A Side-Channel Analysis Resistant Description of the AES S-Box. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2005.

[80] Elisabeth Oswald and Pankaj Rohatgi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*. Springer, 2008.

[81] G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

[82] E. Prouff and M. Rivain. Theoretical and Practical Aspects of Mutual Information Based Side Channel Analysis. In M. Abdalla, D. Pointcheval, P-A. Fouque, and D. Vergnaud, editors, *Applied Cryptography and Network Security, ACNS 2009*, volume 5536 of *Lecture Notes in Computer Science*, pages 499–518, 2009.

[83] Emmanuel Prouff and Matthieu Rivain. Theoretical and practical aspects of mutual information-based side channel analysis. *International Journal of Applied Cryptography*, 2(2):121–138, 2010.

[84] Jean-Jacques Quisquater and David Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of Esmart*, volume 2002, 2002.

[85] M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In Mangard and Standaert [67], pages 413–427.

[86] R. L. Rivest, A Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM 21*, pages 120–126, 1978.

[87] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[88] Alexis Roche, Grégoire Malandain, Xavier Pennec, and Nicholas Ayache. The correlation ratio as a new similarity measure for multimodal image registration. In *Medical Image Computing and Computer-Assisted Interventation—MICCAI'98*, pages 1115–1124. Springer, 1998.

[89] Werner Schindler and Kouichi Itoh. Exponent blinding does not always lift (partial) spa resistance to higher-level security. In *Applied Cryptography and Network Security*, pages 73–90. Springer, 2011.

[90] Jörn-Marc Schmidt and Michael Hutter. Optical and em fault-attacks on crt-based rsa: Concrete results. In *Proceedings of the Austrochip*, pages 61–67. Citeseer, 2007.

[91] Jörn-Marc Schmidt, Michael Tunstall, Roberto Maria Avanzi, Ilya Kizhvatov, Timo Kasper, and David Oswald. Combined implementation attack resistant exponentiation. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 305–322. Springer, 2010.

[92] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In Joye and Quisquater [57], pages 163–175.

[93] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003.

[94] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal, Vol 28, pp. 656-715*, 1949.

[95] S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[96] Miguel Soriano, Sihan Qing, and Javier López, editors. *Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*, volume 6476 of *Lecture Notes in Computer Science*. Springer, 2010.

[97] Youssef Souissi. *Méthodes optimisant l'analyse des cryptoprocesseurs sur les canaux cachés*. PhD thesis, Télécom ParisTech, 2011.

[98] F-X. Standaert, B. Gierlichs, and I. Verbauwhede. Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2008.

[99] F-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010.

[100] François-Xavier Standaert, Tal Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In A. Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[101] Wim Van Eck. Electromagnetic radiation from video display units: an eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[102] V Verneuil. *Elliptic Curve Cryptography and Security of Embedded Devices*. PhD thesis, PhD thesis, Université de Bordeaux, Bordeaux, 2012.

[103] E. Vetillard and A. Ferrari. Combined Attacks and Countermeasures. In D. Gollman and J.-L. Lanet, editors, *Ninth Smart Card Research and Advanced Application IFIP Conference - CARDIS 2010*, volume 6035 of *Lecture Notes in Computer Science*. Springer, 2010.

[104] C. D. Walter. Sliding Windows Succumbs to Big Mac Attack. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2001.

[105] Colin D. Walter. Longer keys may facilitate side channel attacks. In *Selected Areas in Cryptography, SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2003.

[106] Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. Defeating rsa multiply-always and message blinding countermeasures. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2011.

[107] S-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.

[108] S-M. Yen, W-C. Lien, S. Moon, and J. Ha. Power Analysis by Exploiting Chosen Message and Internal Collisions - Vulnerability of Checking Mechanism for RSA-decryption. In E. Dawson and S. Vaudenay, editors, *Mycrypt*

2005, volume 3715 of *Lecture Notes in Computer Science*, pages 183–1956. Springer, February 2005.