PhD-FSTC-7-2009
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on September 22th 2009 in Luxembourg
in candidature for the degree of

**DOCTOR OF THE UNIVERSITY OF LUXEMBOURG
IN COMPUTER SCIENCES**

by

Matthieu Rivain

On the Physical Security of Cryptographic Implementations

Dissertation defense committee:

Alex Biryukov
Professor, Université du Luxembourg

Jean-Sébastien Coron (dissertation supervisor)
Professor, Université du Luxembourg

Louis Goubin
Professor, Université de Versailles St-Quentin-en-Yvelines

Marc Joye
Cryptographer, Thomson R&D

Franck Leprévost (chairman)
Professor, Université du Luxembourg

François-Xavier Standaert
Professor, Université Catholique de Louvain

Cryptography Team, Technology & Innovation Department, Oberthur Technologies

Remerciements

En première instance, je remercie Jean-Sébastien Coron pour m'avoir offert l'opportunité de faire cette thèse en collaboration avec Oberthur Technologies (anciennement Oberthur Card Systems). Il a ainsi joué un rôle primordial dans mon insertion dans le monde industriel ainsi que dans celui de la recherche. Je le remercie d'avoir dirigé cette thèse et de m'avoir soutenu durant ces trois années tout en me laissant une grande liberté dans mon travail de recherche. Je le remercie enfin de m'avoir apporté certaines idées qui ont donné lieu à des publications.

Je tiens ensuite à remercier Emmanuel Prouff pour avoir accompagné mes débuts de chercheur et notamment pour m'avoir formé à la rédaction d'articles scientifiques. Je le remercie également pour sa grande disponibilité, ses nombreux conseils et le suivi de mon travail tout au long de ces trois années. Je le remercie enfin pour nos nombreuses et fructueuses collaborations qui font, en grande partie, de cette thèse ce qu'elle est aujourd'hui.

Je remercie Alex Biryukov, Jean-Sébastien Coron, Louis Goubin, Marc Joye, Franck Leprévost et François-Xavier Standaert de me faire l'honneur de constituer mon jury de thèse.

Je suis également reconnaissant envers Paul Dischamp, manager du groupe Cryptographie & Sécurité d'Oberthur Technologies, de m'avoir accueilli au sein de son équipe et d'avoir ainsi permis à cette thèse d'exister. Je le remercie d'avoir su me laisser le temps nécessaire à mon travail de recherche.

Ces trois années au sein de l'équipe Cryptographie d'Oberthur Technologies ont été ponctuées de nombreux échanges qui m'ont beaucoup apporté tant sur le plan professionnel que sur le plan personnel. J'ai particulièrement apprécié la bonne ambiance régnant au sein de la "zone crypto". Je remercie donc mes collègues (actuels comme anciens) pour leur bonne humeur, leur disponibilité, leurs enseignements et nos travaux communs mais aussi pour nos pauses café, nos rigolades quotidiennes et nos buvages de coups mémorables. Merci donc à Sébastien Aumônier, Régis Bévan, Arnaud Boscher, Clément Capel, Guillaume Dabosville, Jean-Loup Depinay, Paul Dischamp, Quoc-Dung Do, Julien Doget, Emmanuelle Dottax, Magali Fourcade, Gerald Galan, Laurie Genelle, Christophe Giraud, Peter Lee, Antoine Lemarechal, Thomas Malherbes, Sarra Mestiri, Nicolas Morin, Robert Naciri, Gilles Piret, Emmanuel Prouff, Franck Rondepierre, Coraline Streiff, Yannick Sierra, Marie Ténégal, Hugues Thiebeauld et Mehdi Ziat.

Je remercie tout particulièrement mes co-auteurs Régis Bévan, Jean-Sébastien Coron, Julien Doget, Emmanuelle Dottax, Christophe Giraud, Emmanuel Prouff et Yannick Sierra pour nos collaborations.

J'adresse également mes remerciements aux différentes personnes ayant pris le temps de relire les versions préliminaires de ce mémoire. Leurs remarques et leurs suggestions m'ont grandement permis d'en améliorer la qualité. Merci donc à Paul Dischamp, Julien Doget, Emmanuelle Dottax, Laurie Genelle, Christophe Giraud, Emmanuel Prouff, Franck Rondepierre et Yannick Sierra. Merci en particulier à Emmanuelle Dottax pour sa relecture intégrale, et à Christophe Giraud pour m'avoir fourni certaines des figures illustrant ce mémoire. Merci aussi à Ian Dickson, Nicolas Morin et Gilles Piret pour leurs corrections.

Je tiens également à exprimer ma gratitude aux personnes m'ayant soutenu et encouragé à faire une thèse à l'heure où mes doutes quant à mes choix professionnels m'auraient presque fait renoncer. Je tiens particulièrement à remercier Philippe Letellier pour m'avoir incité à prendre cette voie ainsi que pour m'avoir aidé dans mes démarches vers le monde professionnel. Merci également à Guillaume Fumaroli et Stéphanie Salgado pour m'avoir offert l'opportunité d'une première expérience dans le domaine de la cryptographie embarquée ainsi que pour m'avoir encouragé dans mon orientation vers la recherche. Je tiens aussi à remercier Enrico Marazzi et Jean-Louis Roch pour leurs encouragements.

Je remercie également les différentes personnes avec qui j'ai eu d'intéressantes discussions scientifiques, que ce soit lors de conférence ou bien dans le cadre du projet "Secured Algorithm". Merci notamment à Laurent Albanèse, Claude Carlet, Jean-Luc Danger, Aziz El Aabid, Guillaume Fumaroli, Louis Goubin, Sylvain Guilley, Philippe Guillot, Jean-Bernard Fischer, Philippe Hoogvorst, Ange Martineli, Michael Quisquater, Laurent Sauvage et François-Xavier Standaert.

***

Je n'aurais jamais eu l'énergie nécessaire à l'aboutissement de cette thèse sans tous les bons moments partagés avec ma famille et mes amis. Je les remercie d'être présents autour de moi et je leur dédie cette thèse.

Je remercie tout d'abord mes parents pour m'avoir permis d'exister et de devenir l'homme que je suis aujourd'hui. Je les remercie pour leur soutien continuel et pour le grand dévouement dont ils ont toujours fait preuve à l'égard de leurs enfants. Je les remercie enfin pour les personnes qu'ils sont et que j'ai toujours grand plaisir à voir, ainsi que pour l'exemple qu'ils représentent pour moi. Je remercie également ma soeur pour m'avoir sorti de l'enfance unique et de l'ennui qui l'accompagne. Je la remercie pour tous les bons moments que nous partageons régulièrement. Je suis heureux de notre grande complicité.

Je remercie mes grands-parents sans qui je ne serais pas de ce monde. Je salue également mes cousins, cousines, oncles et tantes que j'ai toujours beaucoup de plaisir à voir.

Merci enfin à tous mes amis, bretons et d'ailleurs, pour tous les bons moments que nous avons partagés. Merci pour toutes nos sorties bar, resto, ciné, nos soirées (trop) arrosées, nos week-ends rennais, nantais, parisiens, lillois, les festivals, les vacances au ski, à Lancieux, à Hoedic, à Barcelone, en Lozère, dans le Verdon, et ailleurs. Merci d'avance pour tout ce qu'il nous reste à vivre ensemble. Je remercie spécialement mes trois collocs successifs qui ont supporté mes périodes de travail et de stress au cours desquelles je n'ai certainement pas été des plus facile à vivre.

Résumé

En cryptographie moderne, un système de chiffrement est traditionnellement étudié dans le modèle dit *en boîte noire*. Dans ce modèle, le *cryptosystème* est vu comme un oracle répondant à des requêtes de chiffrement (et/ou déchiffrement) de messages à partir d'une valeur secrète : la clé. La sécurité du cryptosystème est alors définie suivant un simple jeu. Un *adversaire* interroge l'oracle sur le chiffrement (et/ou le déchiffrement) de messages de son choix et, selon les réponses obtenues, tente de déterminer la valeur de la clé secrète (ou encore de chiffrer/déchiffrer un message pour lequel il n'a pas questionné l'oracle). Si, tout en suivant une stratégie optimale, l'adversaire n'a qu'une chance de gain négligeable, la sécurité est alors établie. Plusieurs cryptosystèmes existants ont été prouvés sûrs dans le modèle en boîte noire. Cependant, ce modèle n'est pas toujours suffisant pour établir la sécurité d'un cryptosystème en pratique. Prenons l'exemple de la carte à puce qui est utilisée comme support pour le cryptosystème dans de nombreuses applications telles le bancaire, le contrôle d'accès, la téléphonie mobile, la télévision à péage ou encore le passeport électronique. De par la nature de ces applications, un cryptosystème implanté sur carte à puce est physiquement accessible à de potentiels attaquants. Cet accès physique invalide l'abstraction du cryptosystème par un oracle de chiffrement car il permet à l'adversaire d'en observer et/ou d'en perturber le comportement physique. De nouvelles attaques cryptanalytiques deviennent alors possibles se regroupant sous le terme de *cryptanalyse physique*.

La cryptanalyse physique se compose essentiellement de deux familles principales d'attaques: *les attaques par canaux auxiliaires* et *les attaques par fautes*. L'objet des attaques par canaux auxiliaires est l'analyse des différentes *fuites physiques* d'une implémentation cryptographique durant ses calculs. On compte parmi ces fuites le temps d'exécution, la consommation électrique ainsi que les émanations d'ondes électromagnétiques. L'observation de ces dits canaux auxiliaires fournit de l'information sensible sur le calcul cryptographique. La valeur de la clé peut alors facilement être déterminée par traitement statistique bien que le cryptosystème soit sûr dans le modèle en boîte noire. L'accès à une implémentation cryptographique permet plus qu'une simple observation passive de son comportement physique ; il devient également possible d'en perturber le calcul. Partant de ce principe, les attaques par fautes consistent en la corruption de calculs cryptographiques en vue de l'obtention de résultats erronés. De manière tout à fait surprenante, ces derniers peuvent alors être traités afin d'en extraire de l'information sur la clé secrète.

Cette thèse se focalise sur l'étude de la cryptanalyse physique et de l'implémentation sécurisée de primitives cryptographiques. Nous examinons dans une première partie les attaques par canaux auxiliaires d'un point de vue théorique. Différentes techniques d'attaques se basant sur différents outils statistiques sont abordées. Nous analysons leur taux de succès, nous comparons leur efficacité et nous proposons certaines améliorations. Nos analyses sont illustrées par des résultats de simulations d'attaques ainsi que d'attaques mises en pratique sur carte à puce. La deuxième partie de cette thèse est consacrée à l'une des contre-mesures les plus utilisées contre les attaques par canaux auxiliaires : le masquage de données. Nos investigations se concentrent sur les schémas de masquage génériques pour les chiffrements par blocs tels les standards de chiffrement DES et AES. Nous étudions les schémas existants, exhibant des attaques sur certains d'entre eux, et nous proposons de nouveaux designs. La troisième et dernière partie de cette thèse concerne les attaques par fautes. Nous décrivons tout d'abord une nouvelle attaque sur le chiffrement DES exhibant certains pré-requis à son implémentation sécurisée. Nous étudions ensuite le cas du cryptosystème RSA pour lequel nous proposons une nouvelle contre-mesure, pouvant dans un cadre plus large s'appliquer à tout algorithme d'exponentiation. Nous adressons finalement un problème plus pratique mais tout aussi nécessaire à la sécurité : celui de l'implémentation d'un contrôle de cohérence.

Abstract

In modern cryptography, an encryption system is usually studied in the so-called *black-box model*. In this model, the *cryptosystem* is seen as an oracle replying to message encryption (and/or decryption) queries according to a secret value: the key. The security of the cryptosystem is then defined following a simple game. An *adversary* questions the oracle about the encryption (and/or decryption) of messages of its choice and, depending on the answers, attempts to recover the value of the secret key (or to encrypt/decrypt a message for which he did not query the oracle). If by following an optimal strategy the adversary only has a negligible chance of winning, the system is considered as secure. Several cryptosystems have been proved secure in the black-box model. However, this model is not always sufficient to ensure the security of a cryptosystem in practice. Let us consider the example of smart cards which are used as platforms for cryptosystems in various applications such as banking, access control, mobile telephony, pay TV, or electronic passport. By the very nature of these applications, a cryptosystem embedded on a smart card is physically accessible to potential attackers. This physical access invalidates the modeling of the cryptosystem as a simple encryption oracle since it allows the adversary to observe and disrupt its physical behavior. New attacks then become possible which are known as *physical cryptanalysis*.

Physical cryptanalysis includes two main families of attacks: *side channel attacks* and *fault attacks*. The purpose of side channel attacks is to analyze the different *physical leakages* of a cryptographic implementation during its computation. Chief among these rank timing, power consumption, and electromagnetic radiation. Observing these so-called side channels provides sensitive information about the cryptographic computation. The secret key value can then be easily recovered by statistical treatment although the cryptosystem is secure in the black-box model. The access to a cryptographic implementation enables more than a simple observation of its physical behavior; it is also possible to disrupt its computation. Working on this assumption, fault attacks consist in corrupting cryptographic computations so that they produce erroneous results. Surprisingly, these results can be used in order to recover information about the secret key.

This thesis focuses on physical cryptanalysis as well as on the secure implementation of cryptographic primitives. We examine in the first part side channel attacks from a theoretical viewpoint. Various techniques of attack based on different statistical tools are addressed. We analyze their success rate, we compare their efficiency and we propose some improvements. Our analyses are illustrated by results of simulated attacks as well as practical attacks on smart cards. The second part of this thesis is devoted to one of the most widely used countermeasures to side channel attacks: *data masking*. Our investigations concentrate on generic masking schemes for block ciphers such as the encryption standards DES and AES. We analyze existing schemes, exhibiting some attacks against certain of them and we propose new designs. The third and last part of this thesis deals with fault attacks. First, we describe a new attack on the DES cipher which exhibits some requirements to its secure implementation. We then provide a case study based on the RSA cryptosystem where we propose a new countermeasure which can also be applied to secure any exponentiation algorithm. We finally address an important issue for practical security: the implementation of coherence checks.

Contents

Preliminaries

Part I Side channel analysis

Part II Masking schemes, design and cryptanalysis

Preliminaries

Chapter 1

Introduction to cryptography

Contents

1.1 Introduction

1.1.1 Terminology

Cryptography (from the Greek *kryptos* and *gráphein*, literally "hidden/secret writing") initially refers to the art of *encrypting* (or *ciphering*) a message, that is writing a message under an unintelligible form for anyone unaware of the encryption process. A message under an encrypted form is called a *ciphertext* (or *cryptogram*) while in contrast a non-encrypted message is called a *plaintext*. The operation consisting in restoring the plaintext from the ciphertext is the *decryption* (or *deciphering*). Today, cryptography is a brad discipline whose purpose is to provide security features for communication. *Cryptanalysis* confronts cryptography by trying to break ciphers, namely to retrieve plaintexts from ciphertexts without *a priori* knowledge of the decryption process. The study of cryptography goes hand in hand with the study of cryptanalysis; together they make up *cryptology*: "the science of secrecy".

1.1.2 Brief history

Although being a modern science, cryptography is an ancient art. A common historical example of the use of cryptography dates back to the time of Julius Caesar who encrypted messages for military purpose. The so-called *Caesar's cipher* was very basic: each letter of the message was replaced by the letter at three positions down the alphabet (looping back at the end). Such a cipher is a particular example of a *monoalphabetic substitution cipher* which is a cipher that substitutes every letter of the plaintext by a different letter (or by some character). Such ciphers were widely used until the development of the *frequency analysis* by the Arab mathematician Al-Kindi in the ninth century. Based on the fact that in all languages the different letters have different occurrence frequencies, this technique consists in examining the frequency of the different characters in the ciphertext to determine the corresponding plaintext letters. In order to overcome frequency analysis, *polyalphabetic substitution ciphers* were designed around the sixteenth century and consisted in alternating different monoalphabetic substitutions. A famous example is the *Vigenère's cipher* which was thought to be unbreakable for many years before being shown to be vulnerable to an extension of frequency analysis. More sophisticated polyalphabetic ciphers were then developed in the nineteenth and twentieth centuries, certain based on mechanical devices. A particular example is the *Enigma machine* which was used by the Germans during World War II for radio communications. The Enigma machine was partly broken by the Allies code-breakers, including, among others, the British mathematician Alan Turing (considered as the father of modern computer science). Several communications were decrypted which gave a certain advantage to the Allies.

While most ciphers were sooner or later broken by cryptanalysts, some design principles started to appear. One of the most famous is the *Kerckhoffs' law* (nineteenth century) that is widely accepted today in cryptography: the security of a cryptosystem must only reside in a small secret parameter called the *key*. In other words, the principle states that a cryptosystem should be secure even if everything about its design is publicly available except the key. In 1949, the American mathematician Claude Shannon published a seminal paper for the theory of cryptography [209] based on *information theory* (previously developed by the same author). In particular, he showed that the only way to obtain *perfect secrecy* is to encrypt every message using a different key with the requirement that, for each encryption, the number of possible keys equals the number of possible plaintexts. As such a requirement renders impractical the broad-based deployment of cryptography, Claude Shannon proposed some design principles to obtain *practical secrecy i.e.* improved resistance to cryptanalysis. His work introduced some of the foundations of the design of modern ciphers.

Cryptography has undergone a revolution in the late twentieth century with the development of computers and other electronic devices. These new technologies made it possible to design more complicated ciphers. Such a cipher was designed in the 1970's by the team of IBM cryptographers, including among others Horst Feistel and Don Coppersmith. Their cipher yielded the first data encryption standard (DES) which has been (and is still) widely used and analyzed since then. In 1976, the publication of a paper by the American cryptographers Whitfield Diffie and Martin Hellman put a new face on cryptography [87]. They introduced the concept of *public key cryptography* and they linked cryptography to *complexity theory*. The recent ideas and designs provoked a real effervescence among the cryptographers in the 1980's. Shortly afterwards a real scientific community was born, which has been expanding quickly since that time.

1.1.3 Modern cryptography

Modern cryptography is a branch of applied mathematics and computer science: modern cryptosystems are computer programs (or electronic circuits) whose algorithmic structure is based on mathematical tools. The goal of modern cryptography is to provide some security functionalities for secure communications over an *insecure channel* (*e.g.* telephone, radio, internet, *etc.*) that may be compromised by an eavesdropper. By compromised, one usually means that the eavesdropper may spy the communication, modify its content or even steal the identity of one the two communicating entities. In such a context, cryptography provides technical solutions to ensure the following security features:

— *confidentiality*: a message is unintelligible for anyone except the receiver (and the sender),

— *message authentication*: the receiver can check the identity of the sender of a message,

— *data integrity*: the receiver can verify that the message has not been modified during its transfer.

Asymmetric cryptography (see Section 1.3) also renders possible the *digital signature* of electronic documents which is analogous to the handwritten signature for paper documents. The digital signature has the same properties of message authentication but it further has the property of *non-repudiation*: the author of a signature cannot deny it afterwards. With the development of modern cryptography, many other security features were rendered possible such as *secure multiparty computations*, *group signatures*, *etc.*

Modern cryptography is based on *computational impossibility*. Every cryptosystem can theoretically be broken using an *exhaustive key search* (or *brute force attack*). A single plaintext-ciphertext pair often suffices to non-ambiguously identify the secret key that has been used for ciphering. If an adversary can obtain such a pair (which may occur in usual contexts), then every possible key can be tested by checking whether or not the plaintext encryption yields the corresponding ciphertext. Although such an exhaustive search is always possible, a cipher can use a key space large enough to render the exhaustiveness computationally impossible. For instance, if a key is coded in 128 bits (which is tiny compared to modern computer memories) then 2^{128} key values are possible. In that case, an exhaustive search merely requires performing 2^{128} encryptions. Assuming that an attacker has two billion computers (which is more than the number of existing computers on the planet) that can perform five billions encryptions per second (which is faster than the fastest existing computers), such a computation would require about one thousand billion years (which is about a hundred times longer than the age of universe). Such an exhaustive search is hence considered computationally impossible to this day.

Cryptography divides in two main branches: *symmetric cryptography* and *asymmetric cryptography*. Symmetric ciphers are usually considered secure if no cryptanalytic attack exists that can break them more efficiently than exhaustive search. However, the non-existence of such an attack is not proven in general. As a result, the confidence in the security of a symmetric cipher usually comes with several years of public existence without cryptanalysis. On the other hand, asymmetric cryptosystems are usually based on hard problems from *number theory*. A problem is considered hard if it is computationally impossible to solve. Classical examples are the factorization of large integers or the discrete logarithm computation in a large multiplicative group. An asymmetric cryptosystem is usually provided with a *security proof*. Such a proof consists of a reduction in the sense of *complexity theory*: it is shown that if an algorithm exists that breaks the cryptosystem in a reasonable time, then this algorithm can be used to efficiently solve a given hard problem. Since the problem is considered as computationally insolvable, the non-existence of the breaking algorithm is conjectured and the cryptosystem security follows. Nevertheless, it should be noted that the hardness of the considered problems is not any proven. Here again the confidence in an asymmetric cryptosystem depends on how much the underlying problem has been investigated.

In the rest of this chapter, we present the outlines of modern cryptography, laying stress on the notions that shall be useful to the present thesis. For the interested reader, more details about cryptography can be found in the reference books [161, 203, 221].

1.2 Symmetric cryptography

Symmetric cryptography, also called *secret key cryptography*, is based on the assumption that two communicating entities share a common secret key k.

1.2.1 Symmetric ciphering

A *symmetric cipher* is a bijective function enc parameterized by a secret key k and that operates on messages of arbitrary lengths. The encryption of a plaintext p, into a ciphertext c by a secret key k is then defined as:

$$c = \mathsf{enc}_k(p) \ . \tag{1.1}$$

The decryption of a ciphertext consists in computing the inverse function enc_k^{-1}. For the cipher to be secure, the computation of enc_k and enc_k^{-1} must be computationally impossible without k (even if a large

amount of plaintext-ciphertext pairs are available). For the interested reader, the security of symmetric ciphers is formalized in [35].

Two types of symmetric ciphers can be distinguished which are presented hereafter: *stream ciphers* and *block ciphers*.

1.2.1.1 Stream ciphers

The principle of stream ciphers is derived from the *one-time pad cipher* (also called *Vernam cipher*) which encrypts every plaintext with a different secret key using a simple XOR operation. Assuming that the different keys are fully random, such a cipher has been shown to provide perfect secrecy [209]. However, the one-time pad requires the generation and the sharing of a significant number of secret keys which poses practical issues. To overcome these issues, a stream cipher makes use of a *pseudorandom bit stream generator* g parameterized by a fixed-length secret key k and an initialization vector iv, and it generates a *key stream* ks of arbitrary length. Each bit of the plaintext p_i is then encrypted with a key stream bit ks_i:

$$c_i = p_i \oplus ks_i \ .$$

In practice an internal state σ_i is used that is initialized by $\sigma_0 = f_k(iv)$ for some function f. Then each invocation of the generator updates the internal state and produces a key stream bit: $(ks_i, \sigma_i) = g_k(\sigma_{i-1})$. Note that the usage of a different initialization vector for every encryption is mandatory for the stream cipher security. Otherwise the same key stream would be used for every encryption which would clearly destroy the security.

1.2.1.2 Block ciphers

A block cipher is a bijective function E parameterized with a secret key k that takes as input a n-bit plaintext block and that outputs a n-bit ciphertext block. When a plaintext p of arbitrary length must be ciphered, it is split into several n-bit blocks $p_1, \ldots, p_N$ that are each encrypted into a ciphertext block c_i. This encryption is defined according to a *mode of operation* such as:

— the *electronic codebook (ECB) mode*: $c_i = E_k(p_i)$,

— the *cipher-block chaining (CBC) mode*: $c_i = \begin{cases} E_k(p_1 \oplus iv) & \text{if } i = 1 \\ E_k(p_i \oplus c_{i-1}) & \text{if } i > 1 \end{cases}$,

— the *counter mode*: $c_i = E_k(iv + i) \oplus p_i$,

where iv is a n-bit initialization vector.

The ECB mode is usually considered insecure since identical plaintext blocks yield identical ciphertext blocks, which enables potential attacks. The two other modes are secure provided that a different iv is used for every encryption. Such a requirement is mandatory for the counter mode which is similar to a stream cipher. A fixed iv can be used for the CBC mode but it introduces a potential security weakness (two plaintexts with the same prefix yield two ciphertexts with the same prefix).

A secure symmetric cipher enables confidential communication between two entities sharing a secret key. It further enables user authentication thanks to a *challenge-response protocol*. One party sends a *challenge* (that is a random message) to the other (the challenger). The challenger ciphers the challenge and sends it back as a *response*. The challenge author then deciphers the obtained response and checks whether or not it is equal to the original challenge. If this is the case, then the challenge author knows that the challenger shares the same secret key as him which therefore enables the challenger authentication. Such user authentication should not be mistaken for message authentication where the receiver of a message can check the sender identity. Symmetric ciphering does not provide message authentication nor data integrity. Regarding these issues, one usually makes use of *hash functions* and *message authentication codes*.

1.2.2 Hash functions and message authentication codes

A *cryptographic hash function* is a function that maps a message of arbitrary length into a fixed-length digest value. For a hash function to be secure, it must be computationally impossible to find an antecedent of any given hash value and it must be computationally impossible to find two different messages with the same hash value. Hash functions are used as a building block in many cryptographic schemes (such as signature schemes, see Section 1.3.2). Alone, a hash function provides a partial solution to data integrity. The hash value of a message makes it possible to check its integrity (since no message can be produced with the same hash value) but it has to be securely transmitted (which is an issue over an insecure channel). In order to provide data integrity (as well as message authentication) over an insecure channel, message authentication codes are usually involved.

A message authentication code (MAC) is a function parameterized with a secret key that maps a message of arbitrary length into a fixed-length digest value. MACs are also referred to as *keyed hash functions*. For a MAC to be secure, it must be computationally impossible to compute the MAC value of a message without knowing the secret key (even if a large number of messages with corresponding MAC values are available). A secure MAC provides message authentication and data integrity. Indeed, by verifying that the MAC value of a message is correct, the receiver checks that the message has been sent by someone sharing his secret key and he checks that the message has not been modified by someone lacking knowledge of the secret key. Several MAC constructions exist that are based on hash functions [34] or on block ciphers [45].

1.2.3 Standard block ciphers

Usual block ciphers process an encryption through the repetition of key-dependent permutations, called *round transformations*. In practice, the block ciphers are *iterative*, which means that they apply the same round transformation several times. This round transformation is parameterized by a *round key* that is derived from the secret key by iterating a *key scheduling function*.

We present hereafter the two main standard block ciphers[1]: the data encryption standard (DES) [93] and the advanced encryption standard (AES) [92].

1.2.3.1 The data encryption standard (DES)

The data encryption standard [94] is a block cipher that was selected by the US national bureau of standards in 1976 as an official standard for data encryption. DES uses a 56-bit key (usually represented on 64 bits including 8 parity check bits) and it operates on 64-bit blocks. It has an iterative structure applying 16 times the same round transformation F which is preceded by a bit-permutation IP and followed by a bit-permutation FP. Every round transformation is parameterized by a 48-bit round key k_r that is derived from the secret key k through a key schedule process. To summarize, a ciphertext c is computed from a plaintext p as follows:

$$c = \mathsf{FP} \circ \left(\bigcirc_{r=1}^{16} F_{k_r}\right) \circ \mathsf{IP}(p) \ .$$

The round transformation follows a *Feistel scheme*, namely, the block is split into two 32-bit parts L (the left part) and R (the right part), and F is defined as:

$$F_{k_r}(L, R) = (R, L \oplus f_{k_r}(R)) \ ,$$

where f is a function parameterized with a 48-bit key and operating on a 32-bit block. This structure is illustrated in Figure 1.1.

The function f of the DES first applies an *expansion layer* that expands the 32 input bits into 48 output bits by duplicating 16 of them. The round key is then introduced by bitwise addition afterwards the block is split into eight 6-bit blocks, each entering into a different *substitution box* (S-box) producing a 4-bit output. Finally, the 32 bits from the eight S-box outputs are permuted through a bit-permutation which yields the 32-bit output block.

[1] We only present the core of the ciphers omitting the key scheduling functions (see [92, 93] for further details).

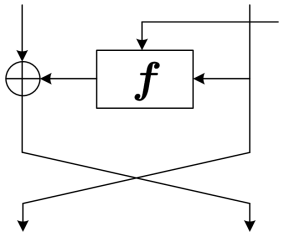


Figure 1.1: Round transformation in the Feistel scheme.

1.2.3.2 The advanced encryption standard (AES)

The advanced encryption standard, also known as Rijndael, was designed by the Belgian cryptographers Joan Daemen and Vincent Rijmen. It is the winner of a five-year standardization process and was announced by the US national institute of standards and technology in 2001 to be the successor of the data encryption standard. AES is a block cipher working on 128-bit blocks with either a 128, 192 or 256-bit key. AES is known to be quite efficient in software due to its byte-oriented design.

AES operates on a 4×4 array of bytes called the state. The state is initialized by the plaintext value and holds the ciphertext value at the end of the encryption. Each round of AES is composed of four stages: AddRoundKey, SubBytes, ShiftRows, and MixColumns (except the last round that omits the MixColumns). AES is composed of either 10, 12 or 14 rounds, depending on the key length (the longer the key, the higher the number of rounds) plus a final AddRoundKey stage.

The SubBytes stage applies the AES S-box to each byte of the state. The AES S-box, which is a bijection, is constructed by composing the multiplicative inversion over the field $\mathbb{F}_2[x]/(x^8+x^4+x^3+x+1)$ with an affine transformation over $\mathbb{F}_2^8$. In the ShiftRows transformation, the bytes in the last three rows of the state are cyclically shifted over different numbers of bytes (1 for the second row, 2 for the third row and 3 for the fourth row). The MixColumns transformation operates on the state column-by-column. Each column is treated as a four-term polynomial over $\mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$ and is multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by

$$a(x) = 3 \cdot x^3 + x^2 + x + 2 \ .$$

Finally, the AddRoundKey transformation simply adds the 16-byte round key to the state by bitwise addition.

The interested reader is referred to [84] for detailed explanations about the design of AES.

1.2.4 The key exchange issue

We have presented how symmetric cryptography provides technical solutions to confidentiality, integrity and authentication over an insecure channel based on the assumption that two communicating entities share a secret key. However an issue still remains. How can a secret key be exchanged in a secure way over an insecure channel? Asymmetric cryptography provides a solution to this issue.

1.3 Asymmetric cryptography

Asymmetric cryptography, also called public key cryptography, was invented by the American cryptographers Whitfield Diffie and Martin Hellman as a solution to the key exchange issue as well as to the digital signature [87]. Its principle lies in the use of two dual keys: a *public key pk* that is used for ciphering (or signature) and a *private key sk* that is used for deciphering (or signature verification). The public key is publicly deployed so that anyone can use it to encrypt a message (or verify a signature). On the other hand, the private key is kept secret so that only its owner can decrypt messages encrypted with the corresponding public key (or produce signatures verifiable by the corresponding public key).

1.3.1 Public key cryptosystem

A public key cryptosystem includes a pair of functions enc (the encryption) and dec (the decryption) parameterized by a public pk and a private key sk respectively, and satisfying $\mathsf{dec}_{sk} = \mathsf{enc}_{pk}^{-1}$. The encryption of a plaintext p into a ciphertext c by the public key pk is then defined as:

$$c = \mathsf{enc}_{pk}(p) \ .$$

And the decryption of a ciphertext c into a plaintext p by the private key sk is defined as:

$$p = \mathsf{dec}_{sk}(c) \ .$$

For a public key cryptosystem to be secure it must be computationally impossible to evaluate dec_{sk} without sk, even if pk is available. For the interested reader, the security of asymmetric ciphering is formalized in [111, 112].

It is worth noting that a public key cryptosystem operates on fixed-length messages. When a longer message must be ciphered, one often use a hybrid ciphering: a symmetric cipher is involved in encrypting the message with a randomly generated secret key k which is itself encrypted using the public key cryptosystem and then attached to the message. The receiver uses his private key to recover k which is then involved in decrypting the message. Hybrid ciphering is widely used in practice which is mostly due to efficiency reasons: a symmetric encryption is in general much faster than an asymmetric encryption.

1.3.2 Digital signature

A public key cryptosystem also renders possible the digital signature of electronic documents. The private key is used to sign and the public key is used to check the signature validity. More precisely, a signature scheme includes a pair of functions sign (the signature) and verif (the verification), respectively parameterized by a private key sk and a public key pk. The signature function takes as input a message of arbitrary length m and computes a signature s. The verification function takes as input a message and a signature and checks whether the signature indeed corresponds to the message. That is, verif_{pk} satisfies:

$$\mathsf{verif}_{pk}(m, s) = \begin{cases} \text{true} & \text{if } \mathsf{sign}_{sk}(m) = s \\ \text{false} & \text{if } \mathsf{sign}_{sk}(m) \neq s \end{cases} \ .$$

For a signature scheme to be secure, it must be computationally impossible to evaluate sign_{sk} without sk, even if pk is available. For the interested reader, the security of signature schemes has been formalized in [36, 113].

It is worth noting that a public key cryptosystem yields a signature scheme by defining $\mathsf{sign}_{sk}(m) = \mathsf{dec}_{sk}(m)$ and $\mathsf{verif}_{pk}(m, s) =$ "$\mathsf{enc}_{pk}(s) = m$". For security reasons and in order to operate on messages of arbitrary length, such a signature scheme often applies a hash function to the message before applying dec_{sk} [36].

1.3.3 Public key infrastructure (PKI)

Asymmetric cryptography solves the key exchange issue: one only needs to make its public key publicly available to enable anyone to send encrypted messages. However a further issue appears: an eavesdropper may publish a key while pretending to be someone else. *Public key infrastructures (PKI)* were created to tackle this issue.

A PKI is a system in which the ownership of a public key by a user is certified by a *certificate authority* that is a trusted third party (*e.g.* a government). The certificate authority issues and manages a set of *digital certificates* (or *public key certificates*) which are electronic documents certifying that a public key belongs to a certain user. A digital certificate contains both the user's public key and identity and it is signed by the certificate authority. When two users want to communicate, they first exchange their certificates. Each user verifies the signature of the received certificate thanks to the certificate authority public key (that is known to every user). If the signature is accepted, the certificate is considered valid and the two users feel confident about their mutual identities and public keys. A secure communication can then be initiated.

1.3.4 The RSA cryptosystem

When the principle of public key cryptography was introduced in 1976, no practical solution was provided and the existence of a public key cryptosystem or signature scheme was conjectured. The first public key cryptosystem was discovered the following year by three cryptographers at Massachusetts Institute of Technology: Ron Rivest, Adi Shamir, and Leonard Adleman [197]. Their cryptosystem, known as RSA (from the initials of the authors' names), is nowadays the most widely used public key cryptosystem.

1.3.4.1 Description

An RSA public key is composed of a public modulus N which is the product of two large secret primes p and q and of a public exponent e which is co-prime with the Euler's totient of N, namely

$$\varphi(N) = (p-1) \cdot (q-1) \ .$$

The corresponding RSA private key is composed of the public modulus N and the secret exponent d that is defined as the inverse of e modulo $\varphi(N)$.

An RSA signature (or deciphering) s of a message $m < N$ is obtained by computing:

$$s = m^d \bmod N \ .$$

The signature verification (or message ciphering) is the inverse operation that can be performed publicly since, according to Euler's theorem, we have:

$$m = s^e \bmod N \ .$$

The security of the RSA cryptosystem is based on the fact that to recover the secret exponent d from the public key (e, N), one must be able to factorize the public modulus N [81] which is commonly believed computationally impossible for a large modulus[2]. Therefore, an entity that only knows the public key cannot *a priori* sign a given message.

1.3.4.2 RSA with Chinese remainder theorem (RSA-CRT)

For the efficient implementation of RSA, one often makes use of the Chinese remainder theorem (CRT). This theorem implies that $m^d \bmod N$ can be computed from $m^d \bmod p$ and $m^d \bmod q$. Consequently, the RSA-CRT consists in performing the following two exponentiations:

$$s_p = m^{d_p} \bmod p \ ,$$

and

$$s_q = m^{d_q} \bmod q \ ,$$

where $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. By Fermat's little theorem, we have $s_p = m^d \bmod p$ and $s_q = m^d \bmod q$. Therefore, once s_p and s_q have been computed, s can be recovered from s_p and s_q by applying a so-called *recombination step*:

$$s = \mathsf{CRT}(s_p, s_q) \ .$$

Two methods exist for CRT recombination: the one from Gauss and the one from Garner. Garner's recombination is the most memory efficient and is defined as:

$$\mathsf{CRT}(s_p, s_q) = s_q + q \cdot \big(i_q \cdot (s_p - s_q) \bmod p\big) \ ,$$

where $i_q = q^{-1} \bmod p$. The entire RSA-CRT is approximately four times faster than the standard RSA which makes its use very common, especially in the context of embedded cryptography (see Section 1.4) where computation time is often critical.

[2] This concerns the security of RSA against full key recovery. Stronger security notions exist (see for instance [36, 111–113]).

1.3.5 Other public key cryptosystems

Since the publication of RSA in 1978 [197], several other public key cryptosystems and signature schemes have been proposed in the literature. Many of them are based on the *discrete logarithm problem*, such as the *ElGamal cryptosystem* [97] and the *ElGamal signature scheme* [97] as well as its widely-used variants: the *digital signature algorithm (DSA)* [91] and the *elliptic curve DSA (ECDSA)* [27].

1.4 Embedded cryptography

1.4.1 The key storage issue

In the previous sections, we have outlined modern cryptography. We have seen some technical solutions to secure communications over an insecure channel. All these solutions make use of secret keys: the knowledge of the secret key enables its owner to decipher encrypted messages, authenticate its messages to other parties or to electronically sign documents. The security of the entire system is based on the secrecy of the key. If the key is exposed then all the security features may be violated. It is therefore crucial to store secret keys in a safe place out of the way of any potential eavesdropper.

Since secret keys are represented as bit strings that can take a huge number of possible values, it is not conceivable to expect users to memorize them. Secret keys must therefore be stored in a physical medium such as a piece of paper or a computer memory. A piece of paper does not provide a satisfactory solution since the key would have to be typed each time: a rather tedious process for the user. Moreover, the secure storage of the piece of paper would still be an open issue. A natural solution is to store the key on a personal computer memory. However, such a solution compels the user to restrict the access to his computer to very trusted people. In addition, most of the time personal computers are connected to the Internet, thereby exposing the content of their memory. Regarding these issues, a common measure is to store the key in an encrypted form according to a password chosen by the user. When the key is required, the user is asked for the password in order to decrypt the key. Such a solution provides double security: an eavesdropper must access to the encrypted key and guess the password in order to break the system. Nevertheless, it should be noted that a password-based encryption only offers a limited degree of security since most of people choose their date of birth as a password or other easily guessable words. In addition to this security issue, computer storage poses an important practical issue: users must have their personal computer with them each time a secure communication is required. Such a requirement is not practical for certain applications (see for instance Section 1.4.3).

In order to give a secure and practical answer to the key storage issue, people imagined small security tokens to store secret keys so that users could easily carry around with them everywhere. In order to avoid key exposure, such tokens would not deliver secret keys but would rather perform cryptographic operations themselves. Such security tokens exist and are widely used at present: they are known as smart cards.

1.4.2 Smart cards

1.4.2.1 History

The concept of smart cards appeared in the 1970's in several countries (France, Germany, Japan and United States) and was the subject of several patents. Its paternity cannot really be attributed to a single inventor but several contributors can be cited such as the Americans Thomas Pomeroy, Jules Ellingboe, Paul Castrucci and John Halpern, the Japanese Kunitaka Arimura, the Germans Jürgen Dethloff and Helmut Gröttrup and the French Roland Moreno, Michel Ugon and Louis Guillou.

Although many patents have been registered, only a few of them gave rise to actual products. The patent of Roland Moreno [171] describes a portable memory equipped with so-called inhibitors that protect the memory access (*e.g.* by requesting a secret code). The first smart cards, known as *memory cards*, resulted from this invention. This technology was in particular developed in France in the early 1980's for the telephone cards used in public telephone boxes. Soon, the success of telephone cards spread throughout Europe and then worldwide. At the same time, the French economic interest group "Carte

Bancaire" (banking card) was created with a view to making smart cards a new means of payment. The first banking memory cards were marketed in 1984. But soon banks turned to a more secure solution: the *microprocessor card*.

The embedding of a microprocessor in a smart card was first suggested by Jürgen Dethloff as an improved inhibitor to protect the memory access of memory cards. The microprocessor card, as it is known today, was patented [239] and developed by Michel Ugon with the company CII-Honeywell Bull. This ambitious project aimed to develop a smart card comparable to a miniature computer that could be programmed and, in particular, that could perform cryptographic computations. The first prototypes were made available in 1984 and the microprocessor card was subsequently adopted by French banks in 1986. Since then, the use of microprocessor cards has become widespread and these are today present in many daily life applications (some of them are presented in Section 1.4.3). The next section gives a brief survey of what a microprocessor card is.

1.4.2.2 Microprocessor cards

A microprocessor card is a pocket-sized plastic card (usually $85.60 \times 53.98 \times 0.76$ millimeters) in which a *microchip* (or *chip* for short) is embedded. A gold-platted area of approximately one square centimeter is usually observed on the card surface. This area is a printed circuit board, usually called the *micromodule*, which acts as the interface between the chip and the card reader. Figure 1.2 shows the picture of a typical microprocessor card.

Figure 1.2: Picture of a French social security smart card.

The chip is composed of a microprocessor also called *central processing unit (CPU)*, and of several memories. The CPU is similar (although more elementary) to traditional computer processors. The register file is often smaller and optimization mechanisms (such as cache) are not always included. Typical bit-sizes for cards microprocessors are 8, 16 and 32, while modern computer processors bit-sizes are usually 32 or 64. The chip contains three different kinds of memories:

— The *read-only memory (ROM)* which cannot be written nor erased. The ROM is set during the manufacturing of the chip and merely contains the *mask* that is the program running on the chip.

— The *electrically-erasable programmable read-only memory (EEPROM)* which can be written and erased but that is also non-volatile (its content is preserved while the chip is not powered). EEPROM typically contains user specific information (*e.g.* name, bank account, client number, *etc.*) and, in particular, the cryptographic keys. It is worth noting that a specific kind of EEPROM, known as *flash*, is commonly used. It is faster, cheaper but also has a shorter lifetime than a traditional EEPROM.

— The *random-access memory (RAM)* which is a fast read-write access volatile memory. As in traditional computers, RAM is the working space of programs during their execution.

The chip may further include additional modules such as (i) *random number generators* used in particular for the random generation of secret keys, (ii) *cryptographic coprocessors* used to speed up the implementation of cryptographic operations and (iii) *checksum mechanisms* used to check the data integrity.

The micromodule is composed of several contact pads that are used for the power supply input (Vcc) and ground (Gnd), the external clocking signal (CLK), the input/output data transmission (I/O) and the reset of the chip (RST). Some modern smart cards, known as *contactless* cards, communicate with readers *via* radio frequencies. Contactless cards do not include a micromodule but a radio-frequency antenna that is printed inside the card plastic all around. There also exist some dual-interface smart cards that include both a micromodule and an radio-frequency antenna allowing both contact and contactless communications.

1.4.3 Embedded cryptography in everyday life

Cryptographic smart cards are currently used in different applications of everyday life. We review several of these applications hereafter.

1.4.3.1 Mobile telephony

In mobile telephony, the GSM (*global system for mobile communication*) and UMTS (*universal mobile telecommunication system*) standard technologies make use of smart cards known as *SIM cards* (where SIM stands for *subscriber identity module*). The SIM card contains the information specific to the subscriber as well as the secret key enabling its authentication on the provider's network. This authentication is performed by a secret key challenge-response protocol. In case of successful authentication, secret key ciphering and message authentication codes are then used to secure the communication between mobile phones and the provider's antennas.

As in many smart card applications, the user authenticates to the card by means of a secret code: the *personal identification number (PIN)*. When the phone is turned on, the user is asked to enter the PIN. Without it, the card does not perform any operation which renders the phone useless.

1.4.3.2 Banking

Smart cards are also involved in banking applications such as electronic payments and cash withdrawals. In most countries, banking cards are magnetic stripe cards that do not include a microprocessor. The magnetic stripe contains the user and bank account information. Such cards can easily be cloned by copying the magnetic stripe content. Moreover, a lost (or stolen) card can be directly used by its finder (or its thief) since no user authentication mechanism (such as the PIN) is implemented, except a written signature on the payment receipts, which is only checked afterwards in case of a dispute. Banking smart cards with a microprocessor do not suffer such security weaknesses and make fraud much more difficult. First, the user is asked for a PIN code prior to each transaction. Moreover, as specified by the *Eurocard-Mastercard-Visa (EMV)* standard, some authentication mechanisms are employed to secure transactions. The card data (composed of the client name, the account number, the card validity period, *etc.*) is signed by the issuing bank. At the beginning of every transaction, the card transmits its signed data together with the issuing bank certificate. This enables the terminal to authenticate the card by verifying the bank certificate (which is signed by a certificate authority) and then by verifying the card data (using the bank certificate). This process, which is called *static data authentication (SDA)*, ensures that the card has indeed been issued by a legitimate bank. However it does not prevent the card from being cloned since the signed card data may be duplicated. That is why a *dynamic data authentication (DDA)* process is usually involved in large transaction. It starts as SDA but the card also provides its own certificate (that is signed by the issuing bank). The terminal then checks the validity of the card certificate and, if valid, sends a challenge to the card. The card signs the challenge and sends it back to the terminal, which then verifies the signature to authenticate the card. The DDA prevents any simple cloning of the card since the card's private key is required. In addition, every transaction (date, place, total amount, *etc.*) is signed by the card which authenticates the transaction and ensures its non-repudiation.

1.4.3.3 E-passport

An electronic passport (or *e-passport*) is not a plastic card as usual smart cards, but rather a classical passport embedding a chip that is similar to a smart card chip. The chip is contactless and uses a

radio-frequency technology to communicate with the readers. The chip contains the different identity information about the holder (that are also written on the passport), a photograph of the holder in digital format as well as fingerprints and iris data as optional biometrics. In order to attest the passport's authenticity, these data are signed by the issuer country by means of an asymmetric signature scheme (RSA, DSA or ECDSA). Airport identity checkpoints are provided with the public keys of the different passport issuers, thereby allowing them to verify signature validity. In order to protect the user data stored on the chip from radio-frequency eavesdropping, the reader must authenticate to the chip. For such purposes, two different protocols have been defined: the *basic access control (BAC)* and the *extended access control (EAC)*. The BAC authentication consists of a challenge-response secret key authentication. The secret key is derived from optically scannable data printed on the passport so that the physical access to the passport is required (hence preventing radio-frequency eavesdropping). Furthermore, the communication between the chip and the reader is secured by secret key ciphering and message authentication codes using different temporary keys for every session. The EAC authentication is much stronger than the BAC authentication since it is based on a PKI and avoids the use of optically scannable information as a secret key. The reader has a certified public key that is transmitted to the chip. The chip can then check the public key's validity and hence verify the user's legitimacy.

1.4.3.4 Pay-TV

A typical pay-TV mechanism makes use of a *set-top box* that receives digital video content under an encrypted form, decrypts it and displays it, depending on the subscriber's rights (*i.e.* depending on whether or not the subscriber paid for the video content). For such purposes the provider often uses a smart card included in the set-top box that stores the subscriber rights as well as the secret keys used for the video decryption. When a customer buys a movie, the provider sends an authenticated and encrypted order to the smart card specifying the movie rights and the decrypting key. Such a system prevents the exposure of the decrypting keys that would enable broad-based piracy.

1.4.3.5 Other applications

Smart cards are also widely used for access control applications. Most companies provide their employees with *access badges* that are necessary for access and movement into the company offices. Computer access is also more and more secured *via* the usage of access badges. Depending on the security required, the access badges may or may not implement cryptographic/public key authentications.

Some public transportation networks also employ access smart cards as a replacement for the traditional cardboard ticket. Such *e-ticket* systems are mostly deployed in megalopolises such as Hong Kong (since 1997), Paris (since 2001), Singapore (since 2001) and London (since 2003), but are also more and more common in smaller towns such as Rennes (France, since 2006).

Another important application is the social security card used in some countries. Its role is to store and protect the medical information about the holder as well as to authenticate the holder to medical institutions. Such a card, known as *Carte Vitale*, has been used in France since 1998 (see the picture in Figure 1.2). Other European countries (*e.g.* Germany, Belgium, Austria) also have their own social security cards. The *e-CEAM* project was launched in 2002 to merge the different solutions into a single European social security card.

To summarize, embedded cryptography is present all around us in our everyday lives to secure our communications. A legitimate question then arises: is this security effective? Do smart cards really protect our secret keys? As explained in the next section and as developed in the rest of this thesis, an affirmative answer to these questions represents a real challenge.

1.4.4 Physical cryptanalysis

In modern cryptography, a cryptosystem is usually studied in the so-called *black-box model*. In this model, a cryptosystem is seen as an oracle replying to some encryption and/or decryption queries according to a

secret key. The attacker is assumed to have access to this oracle, especially as he can ask for the encryption (or the decryption) of the messages of its choice. However he cannot get any further information about the secret key nor about the intermediate results of the cryptographic computation. The security of the cryptosystem is then defined according to the capabilities of an attacker having an unbounded access to encryption (or decryption) queries. Namely, one investigates whether based on such queries, an attacker may retrieve some information about the secret key (or guess the encryption/decryption of a non-queried message). If the number of queries and the computation time required are beyond computational feasibility, the cryptosystem can be considered as secure. The motivation for such a model relies on the principle that the encryption of a plaintext should not provide any exploitable information about the secret key. It may happen in practice that an attacker obtain a high number of ciphertexts corresponding to plaintexts that he knows or even that he has chosen. This information should not allow him to deduce any information about the secret key, nor should it facilitate the encryption or decryption of any other message.

The black-box model is well suited to characterize the intrinsic security of a cryptosystem, namely its effectiveness in providing secrecy. However, it does not enable to ensure its security in the physical world, where the mathematical description of the cryptosystem is replaced by a *physical implementation i.e.* by a device performing the cryptographic computations (*e.g.* a PC, a smart card). In fact, a physical implementation running a cryptographic computation leaks some physically observable information about the intermediate results of the computation. Examples of such leakages are the execution time, the power consumption of the device and the electromagnetic radiation produced during the computation. Moreover, a physical implementation is not tamper-proof: it may be altered and its computations may be disrupted. For instance, inducing a voltage spike to the device power supply may disrupt its execution.

Of course disrupting a computation or observing its physical behavior requires a physical access to the implementation[3]. This happen in practice for embedded cryptographic implementations. Let us illustrate our point with two examples from everyday life. When a banking smart card is involved in performing a payment, the client inserts the card in the shopkeeper's terminal and enters his PIN code. The terminal is then able to query the card for different cryptographic computations. In addition, the terminal has a physical access to the card which allows it to measure the card's physical leakage or to disrupt its execution. Another example is for pay-TV applications. This time, the client is the potential attacker: he has a physical access to the issuer's smart card that contains the secret keys protecting the video content. Many other scenarios could be described where the physical access to an embedded cryptographic implementation is granted to potential attackers.

In this context, new cryptanalytic attacks become possible which are known as *physical cryptanalysis*. Physical cryptanalysis includes two main families of attacks: *side channel analysis* and *fault analysis*. Side channel attacks exploit the physical leakage of the cryptographic computation. This leakage provides sensitive information that often makes it possible to recover the secret key even though the cryptosystem is secure in the black-box model. Regarding fault attacks, they consist in disrupting the cryptographic computation so that it produces erroneous results. These erroneous results are then analyzed in order to deduce information about the secret key.

Physical attacks performed against smart cards can further be divided into different categories depending on how they affect the physical integrity of the card:

— *Invasive attacks*: as explained in [149], a smart card can be *depackaged* in order to extract its microchip. This enables *probing attacks* that directly examine the content of ROM or EEPROM or that spy the memory bus during a computation as well as *destructive attacks* that remove or modify some elements of the chip. Such attacks are complicated to mount in practice and require high-tech microelectronic equipment.

— *Semi-invasive attacks*: the depackaging of the card can be used to facilitate physical cryptanalysis. It is in particular necessary to measure the electromagnetic radiations produced by the chip [99,194] as well as to induce errors *via* light pulses [211]. For such a purpose, a partial depackaging may suffice to expose the chip without altering its integrity. Figure 1.3 shows the results of such depackaging that was performed in the security lab of Oberthur Technologies.

[3]Except for the time that can be measured in any communication from a distance.

— *Non-invasive attacks*: altering the physical integrity of the card is not mandatory for physical cryptanalysis. In particular, measuring the power consumption [148] and injecting faults using power and clock glitches [30, 32] are non-invasive attacks.

Figure 1.3: Smart card depackaging on the back side (on the right) and on the front side (on the left).

Chapter 2

Technical background

Contents

2.1 Basics on probability theory

The calligraphic letters, like $\mathcal{X}$, are used to denote sets. The corresponding large letter X denotes a random variable over $\mathcal{X}$, while the lowercase letter x denotes a particular realization of X. The probability of an event A is denoted by $\mathrm{P}[A]$. The conditional probability of an event A given the occurrence of an event B is denoted $\mathrm{P}[A|B]$. Every discrete random variable X is associated with a *probability mass function* (pmf) $\mathrm{P}_X : x \mapsto \mathrm{P}[X = x]$. If X is continuous, it is associated with a *probability density function* (pdf), denoted by g_X and that satisfies, for every $x \in \mathcal{X}$:

$$\mathrm{P}[X \leqslant x] = \int_{-\infty}^{x} g_X(t)dt \ .$$

The function $x \mapsto \mathrm{P}[X \leqslant x]$ is further called *cumulative distribution function* (cdf) of X. The notation $x \mapsto \mathrm{P}[X = x]$ for a continuous random variable X, may be further used without ambiguity to denote the pdf of X.

The *expectation* $\mathrm{E}[X]$ and the *variance* $\mathrm{Var}[X]$ of a random variable X are respectively defined by:

$$\mathrm{E}[X] = \sum_{x \in \mathcal{X}} \mathrm{P}_X[x] \cdot x \tag{2.1}$$

if X is dicrete,

$$\mathrm{E}[X] = \int_{\mathcal{X}} g_X[x] \cdot x \ dx \tag{2.2}$$

if X is continuous, and:

$$\mathrm{Var}[X] = \mathrm{E}\left[(X - \mathrm{E}[X])^2\right] \ . \tag{2.3}$$

Since the expectation is a linear function of random variables, the variance of X satisfies: $\mathrm{Var}[X] = \mathrm{E}\left[X^2\right] - \mathrm{E}[X]$. The *standard deviation* $\sigma[X]$ of a random variable X is defined as the square root of its variance:

$$\sigma[X] = \sqrt{\mathrm{Var}[X]} \ . \tag{2.4}$$

The *covariance* $\mathrm{Cov}\,[X,Y]$ between two random variables X and Y is defined by:

$$\mathrm{Cov}\,[X,Y] = \mathrm{E}\,[(X - \mathrm{E}\,[X])(Y - \mathrm{E}\,[Y])]\ . \tag{2.5}$$

This definition implies $\mathrm{Cov}\,[X,X] = \mathrm{Var}\,[X]$ and $\mathrm{Cov}\,[X,Y] = \mathrm{E}\,[XY] - \mathrm{E}\,[X]\,\mathrm{E}\,[Y]$.

A n-dimensional random variable $\mathbf{X} = (X_1, \ldots, X_n)$ is called a *random vector* (or a *multivariate random variable*). Its expectation is defined as the vector composed of its coordinates expectations. The *covariance matrix* $\Sigma_{\mathbf{X}}$ of a random vector $\mathbf{X}$ is the analog for the variance for a univariate random variable that is defined as:

$$\Sigma_{\mathbf{X}} = (\mathrm{Cov}\,[X_i, X_j])_{1 \leqslant i,j \leqslant n}\ . \tag{2.6}$$

Two random variables X and Y are said to be *independent* if, for every $(x,y) \in \mathcal{X} \times \mathcal{Y}$, they satisfy:

$$\mathrm{P}\,[X \leqslant x, Y \leqslant y] = \mathrm{P}\,[X \leqslant x] \cdot \mathrm{P}\,[Y \leqslant y]\ . \tag{2.7}$$

For discrete random variables, the previous condition is equivalent to $\mathrm{P}\,[X = x, Y = y] = \mathrm{P}\,[X = x] \cdot \mathrm{P}\,[Y = y]$, for every $(x,y) \in \mathcal{X} \times \mathcal{Y}$.

Finally, the conditional random variable $(X|Y = y)$ is the random variable with pmf/pdf $x \mapsto \mathrm{P}\,[X = x|Y = y]$. If X and Y are independent then we have $(X|Y = y) = X$.

2.2 Pearson correlation coefficient

The Pearson correlation coefficient of two real-valued random variables X and Y is denoted by $\rho\,[X,Y]$. It measures the linear correlation between X and Y and is defined by:

$$\rho\,[X,Y] = \frac{\mathrm{Cov}\,[X,Y]}{\sigma\,[X]\,\sigma\,[Y]}\ . \tag{2.8}$$

For every pair of random variables (X,Y), the correlation coefficient satisfies:

$$-1 \leqslant \rho\,[X,Y] \leqslant 1\ . \tag{2.9}$$

We have $\rho\,[X,Y] = 1$ (resp. $\rho\,[X,Y] = -1$) if and only if X is an increasing (resp. decreasing) affine function of Y (and *vice versa*). On the other hand, if X and Y are independent then $\rho\,[X,Y] = 0$, the converse being false.

We recall hereafter a well-known property of the correlation coefficient.

Property 2.1. *For every $a_1, a_2 > 0$ and for every b_1, b_2:*

$$\rho\,[a_1 \cdot X + b_1, a_2 \cdot Y + b_2] = \rho\,[X,Y]\ . \tag{2.10}$$

2.3 Entropy and mutual information

In information theory, the *entropy* (or *Shannon entropy*) $\mathrm{H}[X]$ of a discrete random variable X aims at measuring the amount of information contained by a realization of X. It is defined by:

$$\mathrm{H}[X] = -\sum_{x \in \mathcal{X}} \mathrm{P}_X[x] \log_2(\mathrm{P}_X[x])\ . \tag{2.11}$$

The *differential entropy* extends the notion of entropy to continuous random variables. It is defined by:

$$\mathrm{H}[X] = -\int_{\mathcal{X}} g_X(x) \log_2(g_X(x))dx\ . \tag{2.12}$$

It is worth noting that, contrary to the entropy, the differential entropy can be negative.

To quantify the amount of information that a second random variable Y reveals about X, the notion of *mutual information* is usually involved. It is the value $\mathrm{I}(X;Y)$ defined by:

$$\mathrm{I}(X;Y) = \mathrm{H}[X] - \mathrm{H}[X|Y] \ , \tag{2.13}$$

where $\mathrm{H}[X|Y]$ is called the *conditional entropy of X given Y* which is defined by:

$$\mathrm{H}[X|Y] = \sum_{y \in \mathcal{Y}} \mathrm{P}_Y[y] \ \mathrm{H}[(X|Y=y)] \ , \tag{2.14}$$

or by:

$$\int_{\mathcal{Y}} g_Y(y) \ \mathrm{H}[(X|Y=y)] \ dy \ , \tag{2.15}$$

depending on whether Y is discrete or continuous.

The mutual information can also be seen as a measure of dependence between two random variables. In particular, X and Y are *independent* iff $\mathrm{I}(X;Y)$ equals 0 and X is a deterministic function of Y iff $\mathrm{I}(X;Y) = \mathrm{H}[X]$.

2.4 Gaussian distribution and Gaussian mixture

The *Gaussian distribution* of dimension n with expectation vector $m \in \mathbb{R}^n$ and covariance matrix $\Sigma \in \mathcal{M}_{n,n}(\mathbb{R})$ is denoted by $\mathcal{N}(m,\Sigma)$, and the corresponding pdf is denoted by $\phi_{m,\Sigma}$. This pdf is defined for every $x \in \mathbb{R}^n$ as:

$$\phi_{m,\Sigma}(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp\left(-\frac{1}{2}(x-m)' \, \Sigma^{-1} \, (x-m)\right) \ ,$$

where $(x-m)'$ denotes the transpose of the vector $(x-m)$ and $|\Sigma|$ denotes the determinant of the matrix Σ. If the dimension n equals 1, then the Gaussian distribution is said to be *univariate* and the single element of the covariance matrix is the variance that is usually denoted σ^2. If $n > 1$, the Gaussian distribution is said to be *multivariate*.

A *Gaussian mixture* is a distribution whose pdf is a finite linear combination of Gaussian pdfs. A Gaussian mixture pdf is denoted ϕ_θ and is defined, for every $x \in \mathbb{R}^n$, by:

$$\phi_\theta(x) = \sum_{t=1}^{T} w_t \cdot \phi_{m_t,\Sigma_t}(x) \ , \tag{2.16}$$

where $\theta = ((w_t, m_t, \Sigma_t))_{1 \leqslant t \leqslant T}$ is a $3T$-dimensional vector containing the so-called *mixing probabilities* w_t's (that satisfy $\sum_t w_t = 1$), as well as the means m_t and the covariance matrices Σ_t of the n Gaussian pdfs in the mixture. These Gaussian pdfs are called the *components* of the Gaussian mixture.

The differential entropy of a Gaussian random variable X with expectation m and covariance matrix Σ satisfies:

$$\mathrm{H}[X] = \frac{1}{2} \log((2\pi e)^n |\Sigma|) \ . \tag{2.17}$$

There is no analytical expression for the entropy of a Gaussian mixture with more that one component. However, upper and lower bounds can be derived. We give hereafter a lower bound.

Proposition 2.1. *[64] Let X be a random variable having a Gaussian mixture distribution with parameter $\theta = ((w_t, m_t, \Sigma_t))_{t=1,\ldots,T}$. Then, its differential entropy satisfies:*

$$\frac{1}{2} \log\left((2\pi e)^n \prod_{t=1}^{T} |\Sigma_t|^{w_t}\right) \leqslant \mathrm{H}[X] \ . \tag{2.18}$$

2.5 Basics on Boolean algebra

The exclusive or (XOR) operator is denoted $\oplus$ and is defined for every $(x, y) \in \{0, 1\}$ as:

$$x \oplus y = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y \end{cases} \tag{2.19}$$

The XOR is also called *binary addition* since it satisfies: $x \oplus y = x + y \bmod 2$. The AND operator is denoted $\wedge$ and is defined for every $(x, y) \in \{0, 1\}$ as:

$$x \wedge y = \begin{cases} 1 & \text{if } x = 1 \text{ and } y = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

The AND is also called *binary multiplication* since it satisfies: $x \wedge y = xy$. The OR operator is denoted $\vee$ and is defined for every $(x, y) \in \{0, 1\}$ as:

$$x \vee y = \begin{cases} 1 & \text{if } x = 1 \text{ or } y = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

For every vectors $x = (x_1, \ldots, x_n) \in \{0, 1\}^n$ and $y = (y_1, \ldots, y_n) \in \{0, 1\}^n$, the previous operators are extended as:

$$x \star y = (x_1 \star y_1, \ldots, x_n \star y_n) \tag{2.22}$$

for $\star$ being $\oplus$, $\wedge$ or $\vee$. The vectorial XOR and the vectorial AND are often called *bitwise addition* and *bitwise multiplication*.

The set $\{0, 1\}$ with the XOR as addition and the AND as multiplication defines a field that is usually denoted $\mathbb{F}_2$. The vectorial space $\mathbb{F}_2^n$ is then provided with a scalar product denoted $\cdot$ and defined for every $x, y \in \mathbb{F}_2^n$ as:

$$x \cdot y = x_1 y_1 \oplus \cdots \oplus x_n y_n \ . \tag{2.23}$$

The *Hamming weight* function $\mathrm{HW}(\cdot)$ is defined over $\mathbb{F}_2^n$ as the number of 1-coordinates of a vector. Namely, $\mathrm{HW}(x)$ is defined for every $x \in \mathbb{F}_2^n$ by:

$$\mathrm{HW}(x) = \sum_{i=1}^{n} x_i \ . \tag{2.24}$$

The following property of the Hamming weight function shall be useful for some of our analyses.

Property 2.2. *Let $x, m \in \mathbb{F}_2^n$. The Hamming weight of $x \oplus m$ satisfies:*

$$\mathrm{HW}(x \oplus m) = \mathrm{HW}(x) + \mathrm{HW}(m) - 2\ \mathrm{HW}(x \wedge m) \ . \tag{2.25}$$

Part I

Side channel analysis

Chapter 3

Introduction to side channel analysis

Contents

3.1 Introduction

Side channel analysis (SCA) is a cryptanalytic technique that takes advantage of information leaking from the physical implementation of cryptosystems. Classical side channels are the timing, the power consumption and the electromagnetic radiations emanating from a device during a cryptographic computation. These *physical leakages* directly depend on the operations performed and on the processed data such as the secret key and the intermediate results of the cryptographic computations. Monitoring these

leakages can hence provide sensitive information that allows an attacker to efficiently recover the secret key.

We give in this chapter a general introduction to side channel analysis. After a brief history, we present the basic principle of these attacks. We then introduce a theoretical model for their analysis. Afterwards, we present the main attack techniques as well as common countermeasures to these attacks. Finally, we address extension of classical side channel analysis to higher-order attacks targeting protected implementations.

3.2 Brief history

The first historical example of side channel analysis appeared during the 1950's when the US government became concerned that electromagnetic radiations emanating from electronic devices could be captured and interpreted to spy confidential communications. Subsequently the *TEMPEST* program was started by the US National Security Agency in order to study the so-called compromising emanations and to define some classified standards to prevent them. In 1985, the first public technical paper was published by Wim Van Eck [240] which explained how to eavesdrop on video display units by monitoring and decoding electromagnetic emanations. The first side channel attack that broke a cryptographic implementation was described by Paul Kocher in 1996 [146]. The author showed how classical implementations of RSA and other public key cryptosystems could be efficiently broken by exploiting the computation times of a few executions. Two years later, Paul Kocher, Joshua Jaffe, and Benjamin Jun described an attack exploiting the power consumed during a cryptographic computation [147, 148]. It was subsequently showed that a similar attack could be mounted based on the electromagnetic radiations emanating from the device [99, 194]. These side channel attacks turned out to be very efficient in practice to break a large range of cryptosystems including DES and RSA.

These publications provoked the great concern of the research community. Several researchers started to investigate these new kinds of physical attacks and ways to prevent them. Today, side channel analysis is a full branch of research in cryptography and embedded systems security. In particular, the international workshop on *Cryptographic Hardware and Embedded Systems (CHES)* has presented several innovative works in this field every year since 1999.

Side channel analysis has also had a great impact on the smart card industry. The design of smart cards had to be rethought in order to take this threat into account. Nowadays, rigorous security evaluations are performed by independent laboratories in order to certify the reliability of smart card products with respect to side channel analysis.

3.3 Basic principle

Figure 3.1 represents the typical equipment involved in a side channel attack (based on the power leakage). A PC sends some encryption commands to the card *via* a smart card reader and obtains the resulting ciphertexts. An oscilloscope is connected to a small resistor in series with the power supply (or ground). For each encryption, the oscilloscope measures the voltage difference across the resistor which yields the power consumption (by $I = U/R$). Each power trace (consumption over time) is sent to the PC that either processes it on-the-fly or stores it for post-treatment.

Electromagnetic analysis requires quite similar equipment, but the oscilloscope is connected to an electromagnetic radiation probe that is placed close to the chip. For such a purpose, the smart card is usually depackaged such as illustrated in Section 1.4.4. Most modern oscilloscopes include several measurements channels which allow them to monitor power and electromagnetic radiation at the same time. As an illustration, Figure 3.2 represents the power consumption and electromagnetic radiation produced during an RSA-CRT computation (see Section 1.3.4.2) on a smart card. It is worth noting that two blocks are clearly observable that correspond to the two modular exponentiations.

Side channel analysis can be divided into two categories of attacks depending on the exploited information. Some attacks focus on the operation flow while others focus on the processed data.

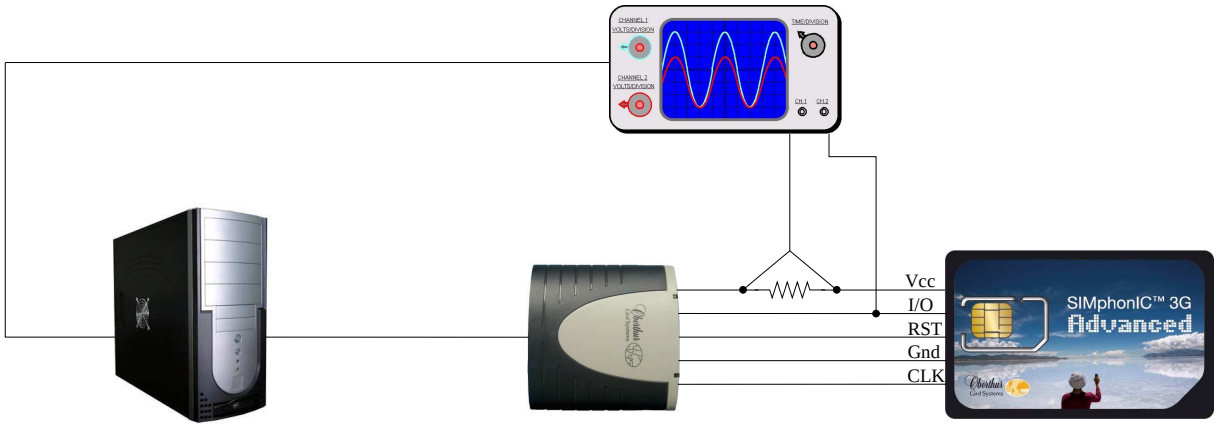


Figure 3.1: Typical equipment for a side channel analysis.

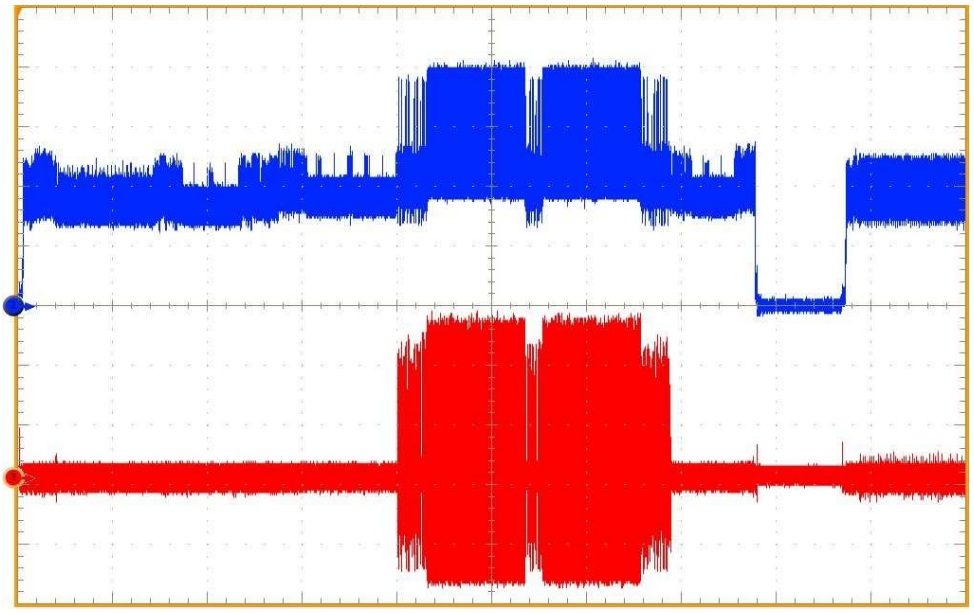


Figure 3.2: Power consumption (at the top) and electromagnetic radiation (at the bottom) produced during an RSA-CRT computation on a smart card.

3.3.1 Attacks on operation flow

The first type of attack are those which exploit the operation flow such as *timing attacks* [146] or *simple power analysis* (SPA) [148]. In fact, the operation flow of a cryptographic algorithm may depend on the secret key. Different operations may require different amounts of time and may induce different power consumption and/or electromagnetic radiation patterns. As a result, an attacker that is able to observe such differences may retrieve information about the secret key.

A typical example of an SPA attack is against an RSA implementation based on the *square-and-multiply* algorithm. In this algorithm (see for instance [161]), every loop iteration performs a modular square and a modular multiplication if the current exponent bit equals 1, otherwise it performs a single modular square. Modular multiplications and modular squares usually have different power consumption/electromagnetic radiation patterns. In that case, the leakage trace is composed of several square patterns interleaved by multiplication patterns only for the loop iterations where the exponent bit equals 1. Consequently, the leakage trace for a single RSA operation reveals the secret exponent. As an il-

lustration, Figure 3.3 shows a zoom on the electromagnetic leakage trace of the RSA-CRT computation given in Figure 3.2, with the corresponding exponent value. We clearly see that when the exponent bit equals 0, one single pattern is observed whereas when it equals 1, two different patterns are observed. In order to thwart SPA, an algorithm must be implemented in an *atomic* way, namely it must have a constant operation flow whatever its input. Some atomic exponentiation algorithms have for instance been proposed in [71, 76, 131, 135].

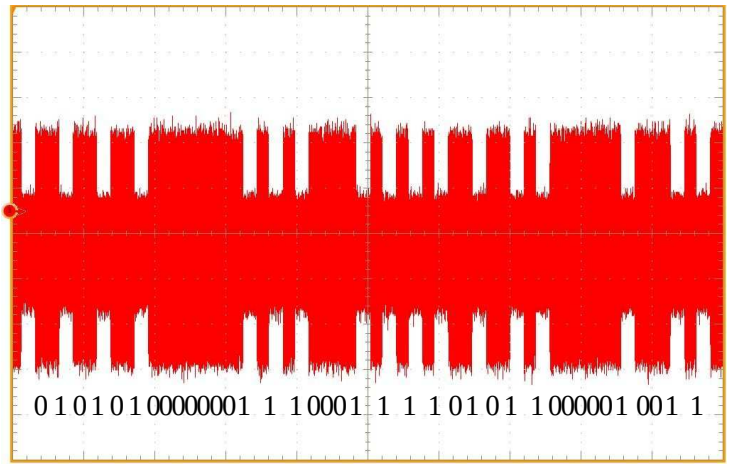


Figure 3.3: Electromagnetic radiation produced by a few loop iterations of a modular exponentiation on a smart card.

Timing attacks are more sophisticated since the adversary does not have access to several leakage patterns for the different operations but only to the overall time of the computation. Therefore the attacker has to perform a statistical analysis of the computation times obtained for several executions. A classical example of timing attack is against RSA using a Montgomery modular multiplication. In this modular multiplication algorithm (see [169]), a final subtraction may or may not be performed depending on the operands values. The overall time of the RSA computation hence depends on each intermediate result of the exponentiation which enables the recovery of the secret exponent [86]. Algorithmic atomicity also ensures security against timing attacks. An alternative solution is to randomize the intermediate results to render them unpredictable [146].

Timing attacks are also rendered possible by some optimization mechanisms of modern processors such as the cache [181, 236] or the branch prediction unit [17]. Here again, depending on the intermediate results, different operations are performed (*e.g.* some memory segments may or may not be loaded in cache) thereby enabling timing attacks. In order to overcome this kind of vulnerability, it is necessary to disable (or at least circumvent) such mechanisms. Otherwise, randomization techniques are still possible.

3.3.2 Attacks on processed data

At any time during the execution of an algorithm on an electronic device, the power consumption and the electromagnetic emanations depend on the intermediate variable being processed. In a cryptographic algorithm, some intermediate variables depend on small parts of the secret key; such variables are said to be *sensitive*. The leakage related to a sensitive variable can be exploited in order to recover the part of the secret key which is involved. For such purposes the adversary makes a guess about the value of the secret key part and, based on this guess, predicts the sensitive variable values for several computations with different known inputs. If the guess is correct then a statistical relation is observed between the predicted values and the leakage measurements, otherwise it is expected that no noticeable relation will be observed. This kind of attack is known as *differential power analysis* (DPA) [147, 148]. Since the first publication of DPA, several improvements have been proposed, in particular through the use of more powerful statistical tools (see for instance [58, 66]).

DPA-like side channel attacks are very powerful and intensive efforts have been made (and are still being made) by the research community to study them, to improve them and to find efficient ways to thwart them. The first part of this thesis is dedicated to the theoretical study and improvement of these attacks. In the following, these attacks will be referred to as *side channel key recovery attacks*. We introduce in the next section a theoretical model to formalize them.

3.4 Theoretical model

3.4.1 Attack model

Let K be a random variable representing a part of the secret key. Let X be a random variable representing a part of a public value such as an input (resp. output) of the target algorithm. Let Z be a random variable representing the result of an intermediate computation of the target algorithm that satisfies $Z = f(X, K)$ for a given function $f : \mathcal{X} \times \mathcal{K} \to \mathcal{Z}$. When K is a part of the secret key that can be exhaustively searched and when f is not constant with respect to k, the intermediate variable Z is said to be sensitive. We denote by L the random variable that represents the side channel leakage generated by the computation (and/or the handling) of Z on a physical implementation of the target algorithm. We shall further denote by $L(z)$ the random variable $(L|Z = z)$.

A side channel key recovery attack targeting the intermediate variable Z aims to recover the value k^* taken by K on a given physical implementation of the target algorithm. For such a purpose, the attacker collects several, say N, leakage measurements $(l_i)_i$ resulting from the computation of $f(x_i, k^*)$ for N inputs $(x_i)_i$. Namely, the l_i's are independent realizations of the random variables $L(f(x_i, k^*))$. Then, the attack makes use of a *distinguisher*, that is a function D which, from the leakage measurements vector $\mathbf{l} = (l_1, \ldots, l_N)$ and the corresponding inputs vector $\mathbf{x} = (x_1, \ldots, x_N)$, outputs a *distinguishing vector* $\mathbf{d} = (d_k)_{k \in \mathcal{K}}$. If the distinguisher is sound and if the leakage brings enough information on Z, then $k^* = \text{argmax}_{k \in \mathcal{K}}\ d_k$ holds with a high probability, namely, the attack is likely to recover the value k^* taken by K. Some metrics for the rigorous evaluation of side channel key recovery attacks are described in Section 3.4.4.

Remark 3.1. For the sake of simplicity, the model described above focuses on attacks targeting a single intermediate variable of the computation. However, an attack may exploit leakages on several intermediate variables (see for instance [49, 196, 201]). An example of particular interest is *higher-order side channel analysis* (see Section 3.7).

3.4.2 Attack classification

A side channel key recovery attack can be characterized according to several criteria. The most traditional are indicated below.

Known *vs.* **chosen plaintext attacks.** A side channel adversary is usually assumed to know the inputs and the outputs of the implementation under attack. That is, each leakage measurement is associated with a plaintext and a ciphertext. In a *known plaintext attack*, the plaintext is assumed to be uniformly distributed while in a *chosen plaintext attack*, it can be chosen by the adversary. In our model, the public variable X is a deterministic function of either the plaintext or the ciphertext (or both). In the first case and under a chosen plaintext attack scenario, the input vector $\mathbf{x}$ of the attack can be chosen by the attacker. Then, the adversary may use a specific strategy to select the x_i values in order to enhance the attack efficiency. Otherwise the x_i's are randomly drawn following the distribution of X (which results from the plaintext/ciphertext uniformity).

Profiled *vs.* **non-profiled attacks.** In a *profiled attack*, the adversary owns a *profile* of the leakage with respect to the processed data. In our model this means that he has a good estimation of the distribution of $L(z)$ for every $z \in \mathcal{Z}$. This makes it possible to use a likelihood distinguisher (see Section 3.5.3) which

is optimal provided that the distribution estimations are precise enough. Such a profile of the leakage is usually obtained through a *profiling stage* in which the attacker has a copy of the attacked device under his control. In a *non-profiled attack*, such a copy is not available and profiling is not possible. As a result, the attacker cannot get a profile of the leakage and hence cannot use a likelihood distinguisher. In that case, more generic distinguishers can be used, such as correlation or mutual information distinguishers (see Section 3.5).

Univariate *vs.* **multivariate attacks.** The power consumption and electromagnetic emanations of a device vary with respect to time. Consequently, the observed leakage is usually a vector whose coordinates correspond to several successive times in the computation. In a *univariate attack*, one single coordinate of this vector is used. That is, in our model, L corresponds to a univariate signal. If the attacker does not know which point in time corresponds to the processing of the target variable Z by the device, then he may apply the attack for each leakage coordinate independently. If the distinguisher is sound and if a sufficient number of leakage measurements is involved, then the correct key guess is clearly distinguished at the manipulation times of Z while no noticeable distinction occurs at other times. In a *multivariate attack*, several coordinates of the leakage are involved and L is a multivariate signal. The choice of the coordinates may either result from a profiling phase or may be defined according to a specific strategy.

Single-channel *vs.* **multi-channel attacks.** In a *single-channel attack*, the attacker has access to one and only one side channel leakage (*e.g.* power consumption or electromagnetic emanations), while in a *multi-channel attack* he has access to multiple side channel leakages (*e.g.* power consumption and electromagnetic emanations). Multi-channel attacks have been investigated in [18, 213].

3.4.3 Leakage models

Most integrated circuits are based on the *complementary metal oxide semiconductor (CMOS)* technology, which is in part due to its low power requirements. The power and electromagnetic leakage produced by a CMOS device results from its switching activity. When the output of a CMOS gate switches from 0 to 1, the corresponding capacitive load is charged from the power supply through the PMOS transistor and when the output value switches from 1 to 0, the capacitive load is discharged through the NMOS transistor. Additionally, each transition provokes a current from the power supply to the ground pin which also consumes some power. In addition, each of these transient currents produces electromagnetic radiation. In comparison, the static power consumption (and electromagnetic radiation) which does not result from the switching activity is negligible. As a result, there is a strong correlation between the data processed by the device and the power consumed (resp. the electromagnetic radiation emanated). Based on this fact and based on experimental studies, several models for the leakage of intermediate variables have been proposed in the literature; the most classical ones are presented below.

The leakage produced by the computation of an intermediate variable Z is modelled by a T-size random vector L over $\mathcal{L} = \mathbb{R}^T$. If $T = 1$ the leakage model is said to be *univariate*; otherwise if $T > 1$ the leakage model is said to be *multivariate*. The leakage $L(z)$ resulting from the computation of a particular value $z \in \mathcal{Z}$ can be seen as the composition of a deterministic part that we shall call *leakage function* and a random part that we shall call *leakage noise*. We formally define these two notions hereafter.

Definition 3.1. *The* leakage function *is the function $\varphi : \mathcal{Z} \mapsto \mathcal{L}$ defined, for every $z \in \mathcal{Z}$, by:*

$$\varphi(z) = \mathrm{E}\left[L | Z = z\right] \ . \tag{3.1}$$

Definition 3.2. *The* leakage noise *is the random variable B defined over $\mathcal{L}$ by:*

$$B = L - \varphi(Z) \ . \tag{3.2}$$

According to the two previous definitions, the leakage related to the computation of the intermediate result Z can be written as:

$$L = \varphi(Z) + B \ . \tag{3.3}$$

Several assumptions are usually made about the leakage model. Some are related to the noise and others are related to the leakage function. We give hereafter the most common assumptions of the side channel literature.

Assumption 3.1 (independent noise assumption)**.** *The noise B is independent of the intermediate variable Z.*

This assumption is frequently made in the literature and it has been practically validated numerous times (see for instance [106, 201, 216]). The noise in the leakage is indeed often independent of the target signal. This is especially true if most of the noise amount is produced by a noise generator (independent of the target algorithm) as a countermeasure to side channel analysis.

Assumption 3.2 (Gaussian noise assumption)**.** *The noise B has a Gaussian distribution.*

The Gaussian noise assumption is fairly realistic in practice since the noise can be modelled as the sum of several contributions of uncorrelated events. In addition, this assumption is very common in the literature (see for instance [66, 157, 201, 217]). It is worth noting that the independent noise assumption and the Gaussian noise assumption are not related and can be made (or not) independently of each other.

The next three assumptions concern the leakage function. As mentioned above, the leakage results from logical transitions occurring on the circuit wires. It is therefore reasonable to assume that every bit of a processed variable contributes independently to the overall instantaneous leakage [201]. This assumption that we shall call *independent bit leakage (IBL) assumption* is formalized hereafter.

Assumption 3.3 (IBL assumption)**.** *Let $\mathcal{Z} = \{0,1\}^n$ for some $n \in \mathbb{N}$. For every $z \in \mathcal{Z}$ and for every $t \in \{1, \ldots, T\}$, the coordinate φ_t of the leakage function $\varphi = (\varphi_1, \ldots, \varphi_T)$ can be expressed as:*

$$\varphi_t(z) = c_{t,0} + \sum_{i=1}^{n} c_{t,i} \cdot z[i] \ , \tag{3.4}$$

where $c_{t,i}$ are constants and $z[i]$ denotes the i^{th} binary coordinate of z.

Making the additional assumption that every transition equivalently contributes in the leakage, we obtain the so-called *Hamming distance model*.

Assumption 3.4 (Hamming distance model)**.** *For every $t \in \{1, \ldots, T\}$, there exists a constant $r_t \in \mathcal{Z}$ such that the coordinate φ_t of the leakage function can be expressed, for every $z \in \mathcal{Z}$, as:*

$$\varphi_t(z) = \delta_t + \varepsilon_t \cdot \mathrm{HW}(z \oplus r_t) \ , \tag{3.5}$$

where ε_t and δ_t are constants.

The Hamming distance model is a particular case of the IBL assumptions where the weights of the different bits satisfy $c_{t,i} = \pm\varepsilon_t/2$. The value r_t is usually called *reference state* [58]. An interesting particular case is $r_t = 0$. This typically occurs with pre-charged logic for which loads are charged or discharged at each clock cycle before the evaluation of the logic functions. This particular case is known as *Hamming weight model*.

Assumption 3.5 (Hamming weight model)**.** *For every $z \in \mathcal{Z}$ and for every $t \in \{1, \ldots, T\}$, the coordinate φ_t of the leakage function can be expressed as:*

$$\varphi_t(z) = \delta_t + \varepsilon_t \cdot \mathrm{HW}(z) \ , \tag{3.6}$$

where ε_t and δ_t are constants.

Remark 3.2. It is worth noting that the reference state may not be constant. A typical example is when an intermediate variable Z_1 overwrites a previously processed variable Z_0. In that case, the target variable should be the virtual variable $Z = Z_0 \oplus Z_1$.

3.4.4 Side channel metrics

We review hereafter classical metrics for side channel analysis. Some of them are leakage metrics and others are attack metrics. Their purposes are different: the first ones aim to measure the information contained in a side channel leakage L about the processed variable Z (or the secret key K) independently of any distinguisher and the latter aim to quantify the effectiveness of a side channel attack in a given context (leakage model, distinguisher, *etc.*).

3.4.4.1 Leakage metrics

Signal-to-noise ratio. A classical metric of signal processing is the *signal-to-noise ratio* (SNR). This notion is usually defined as the ratio of the *significant signal power* to the *noise signal power*. This definition is ambiguous since the notion of *signal power* is not clearly defined. As suggested in [156], a sound definition for this power in our context is the signal variance. In that case, the leakage SNR is defined as:

$$\mathrm{SNR} = \frac{\mathrm{Var}\left[\varphi(Z)\right]}{\mathrm{Var}\left[B\right]} \ . \tag{3.7}$$

Correlation. Another sound leakage metric is the correlation coefficient between the significant signal and the leakage signal: $\rho\left[\varphi(Z), L\right]$. This metric was suggested by Stefan Mangard in [156] where it is shown to satisfy:

$$\rho\left[\varphi(Z), L\right] = \frac{1}{\sqrt{1 + \frac{1}{\mathrm{SNR}}}} \ . \tag{3.8}$$

This metric is related the effectiveness of side channel attacks using the correlation coefficient as distinguisher (see Section 3.5.1). Indeed, as argued in [157, 220], the number of leakage measurements required by such an attack can be approximated by $c/\rho\left[\varphi(Z), L\right]^2$ where c denotes a constant depending on the number of key guesses and of the desired success rate for the attack.

Mutual information. Another metric that can be used to evaluate the information contained in the side channel leakage is the mutual information between the leakage L and either the target variable Z or the secret key element K. This approach was suggested by François-Xavier Standaert, Tal Malkin and Moti Yung in [216, 217].

The mutual information $\mathrm{I}(K; L|X)$ can be used to quantify the information leakage on the key for a given implementation. The mutual information $\mathrm{I}(Z; L)$ quantifies the information about the processed data that is revealed by the side channel leakage. In particular, it can be used to compare several countermeasures or signal processing techniques independently of any algorithm [154, 213]. These two metrics are closely related since they satisfy:

$$\mathrm{I}(K; L|X) = \mathrm{I}(Z; L) - \mathrm{I}(X; L) \ . \tag{3.9}$$

Interestingly, if Z is statistically independent of X (*e.g.* $Z = f'(X \oplus K)$ with K uniformly distributed) then the two metrics are equal.

3.4.4.2 Attacks metrics

Attacks success rate. The success rate is a classical metric in side channel analysis. Usually, a key recovery attack is considered successful if the distinguishing vector satisfies $k^* = \mathrm{argmax}_{k \in \mathcal{K}}\ d_k$. In [217], the authors propose to extend the notion of success rate to different orders. The o^{th}-order success rate of a side channel attack using a distinguisher D and a public vector $\mathbf{x}$, and targeting a secret key k^* is defined as:

$$\text{Succ-}o^{\mathsf{D}}_{\mathbf{x},k^*} = \mathrm{P}\left[\left(l_i \leftarrow L\left(f(k^*, x_i)\right)\right)_i \ ; \ \mathbf{d} \leftarrow \mathsf{D}(\mathbf{x}, \mathbf{l}) \ : \ k^* \in \underset{k \in \mathcal{K}}{\text{argmax-}o}\ d_k\right] \ ,$$

where argmax-$o_{k\in\mathcal{K}}$ d_k denotes the set of the o elements $k \in \mathcal{K}$ that maximize d_k. The notion of order is motivated by the fact that an attacker may perform an off-line exhaustive search after the side channel analysis. A o^{th}-order success means that the attacker has at the most o key candidates to test after the attack in order to recover the correct one.

Guessing entropy. The guessing entropy [60, 160] is defined as the expected number of key guesses to test before recovering a target key value. As pointed out in [217], the guessing entropy is relevant in the context of side channel analysis since it indicates the average workload to perform after side channel analysis. Let $\text{rank}_k(\mathbf{d})$ denote the index $i \in \{1, \ldots, |\mathcal{K}|\}$ such that d_k is the i^{th} higher element of $\mathbf{d}$. The guessing entropy of a side channel attack using a distinguisher D and a public vector $\mathbf{x}$, and targeting a secret key k^* is formally defined as:

$$\text{GE}^{\mathsf{D}}_{\mathbf{x},k^*} = \text{E}\left[\left(l_i \leftarrow L\left(f(k^*, x_i)\right)\right)_i \; ; \; \mathbf{d} \leftarrow \mathsf{D}(\mathbf{x}, \mathbf{l}) \;:\; \text{rank}_{k^*}(\mathbf{d})\right] \; . \tag{3.10}$$

Relation between the two metrics. We show hereafter that the guessing entropy is related to the success rate of every order. In fact, the correct key guess is rated at the o^{th} rank in the distinguishing vector if and only it is rated among the o first candidates but it is not rated among the $o-1$ first candidates. As a result, the probability that the correct key guess be rated at the o^{th} rank satisfies for every o:

$$\text{P}\left[\text{rank}_{k^*}(\mathbf{d}) = o\right] = \text{Succ-}o^{\mathsf{D}}_{\mathbf{x},k^*} - \text{Succ-}(o-1)^{\mathsf{D}}_{\mathbf{x},k^*} \; , \tag{3.11}$$

where Succ-0 is naturally defined at zero. This leads to the following relation:

$$\text{GE}^{\mathsf{D}}_{\mathbf{x},k^*} \;=\; \sum_{o=1}^{|\mathcal{K}|} o \cdot \text{P}\left[\text{rank}_{k^*}(\mathbf{d}) = o\right] \;=\; |\mathcal{K}| - \sum_{o=1}^{|\mathcal{K}|-1} \text{Succ-}o^{\mathsf{D}}_{\mathbf{x},k^*} \; . \tag{3.12}$$

Evaluation of the metrics. Most of the time, the success rate of an attack is empirically evaluated by performing the attack several times. This approach is not very satisfactory since it implies an important time complexity and it may even become impossible for attacks with medium or high complexity. It is therefore not suitable to efficiently and precisely determine the resistance of an embedded device if this one is not quite weak. To tackle this issue, it is of particular interest to investigate efficient ways to compute (or at least to precisely estimate) the success rate of an attack without having to perform it many times.

Issue 3.1. *Given an order o, a distinguisher D, an input vector $\mathbf{x}$ and a secret key element k^*, how to efficiently compute the success rate* $\text{Succ-}o^{\mathsf{D}}_{\mathbf{x},k^*}$ *?*

This issue is addressed in Chapter 4 under the Gaussian noise assumption for the two most widely used side channel distinguishers: the correlation and the likelihood (see Section 3.5).

3.5 Classical side channel distinguishers

This section presents some of the most usual distinguishers to perform a side channel key recovery attack.

3.5.1 Correlation

Correlation-based distinguishers are probably the most widely used to perform side channel key recovery in practice. Moreover, a great portion of the side channel literature focuses on their analysis and their improvement (see for instance [20, 40, 58, 165]).

Remark 3.3. The correlation-based distinguishers presented hereafter are univariate, namely they take has input a set of 1-sized leakage measurements. When they must be applied to a leakage trace (corresponding to several times), they are applied to every coordinate independently.

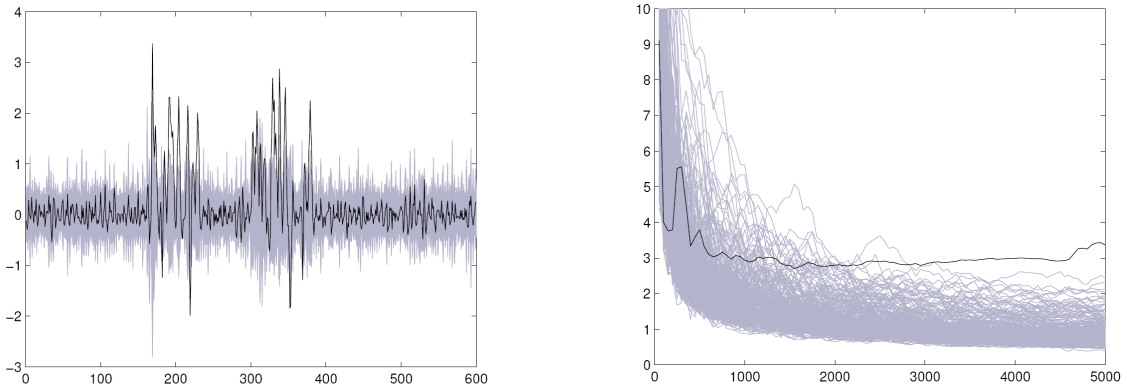
3.5.1.1 Difference of means

The first example of a correlation-based distinguisher is the *difference of means* used in the original DPA [148]. According to a guess k on the value of k^* and to a bit index j, the attacker predicts the value of the j^{th} bit of Z and he estimates the difference between the leakage expectation when the predicted bit equals 1 and the leakage expectation when the predicted bit equals 0. Namely, the attacker estimates the difference $\mathrm{E}\left[L|f_j(X,k)=1\right]-\mathrm{E}\left[L|f_j(X,k)=0\right]$ where f_j denotes the j^{th} binary coordinate of f. For such a purpose, the attacker first computes the *prediction vector* $(f_j(x_i,k))_{i\leqslant N}$ which is composed of the predicted values of the j^{th} bit of the target variable for the different executions. Then, the leakage measurements are separated in two categories: the ones for which the predicted bit $f_j(x_i,k)$ is equal to 1, and the ones for which it is equal to 0. Finally, the so-called differential Δ_k corresponding to the difference between the mean values of the two sets is computed:

$$\Delta_k = \frac{\sum_i f_j(x_i,k)\cdot l_i}{\sum_i f_j(x_i,k)} - \frac{\sum_i (1-f_j(x_i,k))\cdot l_i}{\sum_i (1-f_j(x_i,k))} \ . \tag{3.13}$$

Soundness. Let us denote by δ the difference $\mathrm{E}\left[L|Z[j]=1\right]-\mathrm{E}\left[L|Z[j]=0\right]$. For the correct key guess, the differential Δ_k tends towards δ as the number of leakage measurements grows. On the other hand, for a wrong key guess, Δ_k tends towards $(1-2\alpha)\delta$ where $\alpha\in[0,1]$ denotes the proportion of errors $f_j(x_i,k)\neq f_j(x_i,k^*)$. Since, for a wrong guess, α is usually close to $\frac{1}{2}$, Δ_k tends towards a value close to 0 which shows the soundness of the attack.

Remark 3.4. The previous assertion implicitly assumes that the coordinates $(f_i(X,k))_{k\in\mathcal{K},i\neq j}$, are independent of $f_j(X,k^*)$. This is not always strictly the case and this assumption strongly depends on the non-linearity of the function f (see [61, 191]).



(a) Difference-of-means for different key guesses over the time.

(b) Maximal difference-of-means for different key guesses over the number of leakage measurements.

Figure 3.4: DPA attack against an AES S-box based on the difference-of-means distinguisher.

Example. As an illustration, Figure 3.4 shows the results of a practical DPA attack based on the difference-of-means distinguisher. The leakage corresponds to the power consumption of a smart card performing an AES S-box computation (first round) in software. The targeted bit is the least significant bit of the (predicted) S-box output. Figure 3.4.(a) shows the difference-of-means values for the different key guesses over the time obtained based on 5000 power consumption measurements. Figure 3.4.(b) represents the convergence of the highest difference-of-means value for every key guess over the number of measurements. For both graphics, the curve corresponding to the correct key guess is plotted in black while others are plotted in gray. The attack clearly enable the recovery of the key value.

3.5.1.2 Pearson correlation coefficient

The use of the Pearson correlation coefficient (see Section 2.2) as a side channel distinguisher is a natural generalization of DPA. This approach was followed in several works a few years after the initial publication of DPA (see for instance [57, 80, 85, 151, 156, 218]). In particular, it has been deeply analyzed by Eric Brier, Christophe Clavier and Francis Olivier in [57, 58]. The authors named this attack *correlation power analysis* (CPA) but it is often referred to as DPA. It is nowadays the most widely used approach to perform DPA in the SCA literature and in the smart card industry.

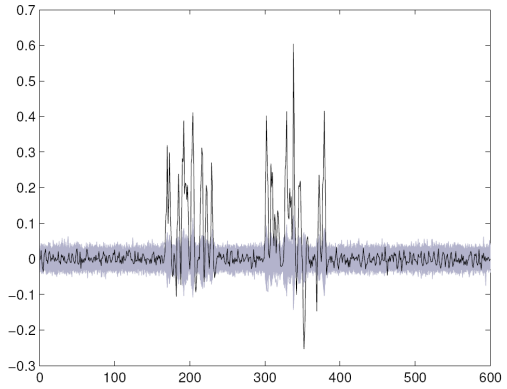
The adversary is assumed to own a model of the side channel leakage, that is a function $\mathsf{M} : \mathcal{X} \times \mathcal{K} \to \mathbb{R}$ such that $\mathsf{M}(x, k)$ is linearly related to the expectation of the leakage $L\left(f(x, k)\right)$, namely to $\varphi \circ f(x, k)$. In practice, the attacker estimates the leakage function φ (or an affine transformation of it) by a *prediction function* $\hat{\varphi}$ and set the model to $\mathsf{M}(x, k) = \hat{\varphi} \circ f(x, k)$. The attack consists in estimating the correlation coefficient $\rho\left[\mathsf{M}(X, k), L\right]$ for every key guess $k \in \mathcal{K}$. This correlation is estimated based on the prediction vector $\big(\mathsf{M}(x_1, k), \ldots, \mathsf{M}(x_N, k)\big)$ and the leakage measurements vector $\mathbf{l}$ by the following coefficient:

$$\rho_k = \frac{\frac{1}{N}\sum_i \Big(\mathsf{M}(x_i, k) - \frac{1}{N}\sum_j \mathsf{M}(x_j, k)\Big)\Big(l_i - \frac{1}{N}\sum_j l_j\Big)}{\sqrt{\frac{1}{N}\sum_i \Big(\mathsf{M}(x_i, k) - \frac{1}{N}\sum_j \mathsf{M}(x_j, k)\Big)^2}\sqrt{\frac{1}{N}\sum_i \Big(l_i - \frac{1}{N}\sum_j l_j\Big)^2}} \ . \tag{3.14}$$

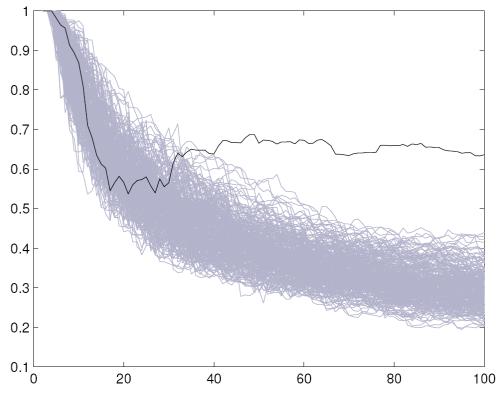
Soundness. The coefficient ρ_k tends towards $\rho\left[\mathsf{M}(X, k), L\right]$ as the number N of leakage measurements grows. If the leakage model is sound, namely if the model satisfies $\mathsf{M}(x, k) = \hat{\varphi} \circ f(x, k)$ with $\rho\left[\varphi, \hat{\varphi}\right] = 1$, then we have (see [156]):

$$\rho\left[\hat{\varphi} \circ f(X, k), L\right] = \rho\left[\varphi \circ f(X, k), L\right] = \rho\left[\varphi \circ f(X, k), \varphi \circ f(X, k^*)\right] \cdot \rho\left[\varphi(Z), L\right] \ . \tag{3.15}$$

From this equation, it is clear that the expected correlation is maximal for the good key guess. Indeed, $\rho\left[\varphi \circ f(X, k), \varphi \circ f(X, k^*)\right]$ equals 1 if $k = k^*$ and it is strictly lower than 1 otherwise (assuming $\varphi \circ f(\cdot, k) \neq \varphi \circ f(\cdot, k^*)$).



(a) Correlation for different key guesses over the time.



(b) Maximal correlation for different key guesses over the number of leakage measurements.

Figure 3.5: DPA attack against an AES S-box based on the Pearson correlation distinguisher.

Example. As an illustration, Figure 3.5 shows the results of a practical DPA attack based on the Pearson correlation coefficient. The leakage measurements used for this attack are the same than those used for the attack presented in the previous section. They correspond to the power consumption of a smart card performing an AES S-box computation (first round) in software. The model used for the attack is the Hamming weight of the predicted S-box output. Figure 3.5.(a) shows the correlation values for the

different key guesses over the time obtained based on 5000 power consumption measurements. Figure 3.5.(b) represents the convergence of the highest correlation value for every key guess over the number of measurements. For both graphics, the curve corresponding to the correct key guess is plotted in black while others are plotted in gray. It is worth noting that this attack requires significantly less power measurements to rank the correct key guess as first compared to the difference-of-means attack presented in the previous section. This comes from the fact that the correlation attack predicts all the computed bits whereas the difference-of-means attack only predicts one computed bit over eight. The contribution of the seven remaining bits to the power then acts as a noise (which is often referred to as *algorithmic noise* in the literature [165]).

3.5.1.3 Relation between the two distinguishers

We address here the relation between the difference-of-means and the Pearson correlation distinguishers in order to exhibit their similarities and disparities. It is a common belief that the Pearson correlation coefficient is a better distinguisher than the difference of means. As argued hereafter, it is actually mostly more generic due to the choice of the leakage model.

Under a single-bit leakage model (one single bit of Z is predicted), $\mathsf{M}(x,k) = f_j(x,k)$, it can be shown that the Pearson correlation distinguisher behave in a close way as the difference-of-means distinguisher. In particular, assuming that the sets $\{i; f_j(x_i,k) = b\}$ have the same cardinal for every $(b,k) \in \{0,1\} \times \mathcal{K}$ (which is a reasonable assumption), we have $\rho_k = c \cdot \Delta_k$ where c denotes a constant value with respect to k. In that case, we clearly have the equivalence of the two distinguishers.

Under a multi-bit leakage model, the difference-of-means distinguisher can be generalized using a weighted sum. Namely, $\Delta_k^{(j)}$ is computed for every j and the obtained values are summed after being weighted by some constants ω_j. This is almost equivalent to compute a Pearson correlation distinguisher ρ_k with model $\mathsf{M}(x,k) = \sum_j \omega_j f_j(x,k)$. Once again, assuming that the sets $\{i; f_j(x_i,k) = b\}$ have the same cardinal for every $(b,k) \in \{0,1\} \times \mathcal{K}$, we have $\rho_k = c \cdot \sum_j \omega_j \Delta_k^{(j)}$ where c denotes a constant value with respect to k.

To summarize, for a given adversary with a given knowledge about the leakage model, the difference-of-means distinguisher and the Pearson correlation distinguisher are almost equivalent. The main difference between them comes from the fact that the Pearson correlation is normalized (*i.e.* it ranges over $[-1,1]$). Although this normalization does not change the classification between the key guesses for a given time coordinate, it enables a better selection of the significant coordinates by eliminating noise spikes in the correlation trace.

3.5.2 Mutual information

Mutual information is a classical tool from information theory that measures the statistical dependence between two random variables (see Section 2.3). The use of the mutual information as side channel distinguisher was initially proposed by Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel in [104,105]. A similar approach was independently proposed by Sébatien Aumônier in [29]. Side channel attacks using mutual information as distinguisher are usually referred to as *mutual information analysis* (MIA).

In a MIA attack, the adversary uses a model M such that $\mathsf{M}(X,K)$ is statistically dependent on L. The attack then consists in estimating the mutual information $\mathrm{I}(\mathsf{M}(X,k);L)$ for every key guess $k \in \mathcal{K}$. This mutual information satisfies:

$$\mathrm{I}(\mathsf{M}(X,k);L) = \mathrm{H}[L] - \mathrm{H}[L|\mathsf{M}(X,k)] \ . \tag{3.16}$$

The key guess maximizing the mutual information $\mathrm{I}(\mathsf{M}(X,k);L)$ is hence the one minimizing the entropy $\mathrm{H}[L|\mathsf{M}(X,k)]$. Therefore, one only needs to estimate this conditional entropy to mount the attack. According to (2.14), it satisfies:

$$\mathrm{H}[L|\mathsf{M}(X,k)] = -\sum_{y \in \mathrm{Im}(\mathsf{M})} \mathrm{P}\left[\mathsf{M}(X,k) = y\right] \mathrm{H}[L|\mathsf{M}(X,k) = y] \ . \tag{3.17}$$

The entropies $\mathrm{H}[L|\mathsf{M}(X,k) = y]$ can be directly computed from the pdf of the conditional leakages $(L|\mathsf{M}(X,k) = y)$. Based on the pairs (x_i, l_i), the attacker derives some estimations of these pdfs for every $y \in \mathrm{Im}(\mathsf{M})$. This allows him to compute some estimations of the entropies $\mathrm{H}[L|\mathsf{M}(X,k) = y]$ and, according to (3.17), an estimation for the overall conditional entropy $\mathrm{H}[L|\mathsf{M}(X,k)]$. The key guess minimizing this estimation is selected as good key candidate.

Soundness. The soundness of MIA is related to two issues. A first issue is the choice of the model M. Unlike correlation attacks, the model does not need to be linearly related to the leakage function. Indeed, the mutual information detects every kind of statistical dependence, not only linear ones. As a result, the attacker can use $\mathsf{M} = f$ as a model, namely he can estimate the mutual information between the leakage and the target variable. This choice is of particular interest since the attack does not require any assumption about the leakage model. However this approach has an important drawback: if the function $f(\cdot, k)$ is injective for all k, then the mutual information $\mathrm{I}(f(X,k); L)$ is constant with respect to k which implies that the mutual information distinguisher is constant for every $k \in \mathcal{K}$ and no information is gained about the secret key. For the attack to succeed when $f(\cdot, k)$ is injective, the model must be defined as $\mathsf{M} = \hat{\varphi} \circ f$ where $\hat{\varphi}$ is a non-injective function that statistically depends on φ. For instance, in the Hamming weight model, one shall use $\mathsf{M} = \mathrm{HW} \circ f$.

Example. A classical target variable in side channel attack is an S-box output of the form $Z = S(X \oplus K)$. If the S-box S is non-injective (as in DES for instance) then one can use $\mathsf{M} = f$, otherwise if S is bijective (as in AES for instance) then one must use $\mathsf{M} = \hat{\varphi} \circ f$ with $\hat{\varphi}$ being non-injective.

A possibility to avoid this drawback is to attack a "non-injective" sensitive variable (*i.e.* such that the functions $f(\cdot, k)$ are non-injective). For instance, in AES, one could target the result of the bitwise addition between two S-box outputs occurring in the first MixColumns operation.

The second issue of MIA is the choice of a sound pdf estimation method, namely a method ensuring that the estimated pdf tends as fast as possible towards the effective pdf as the number of leakage measurements grows. It was initially suggested in [104, 105] to use a histogram estimation. This method is sound but it is very basic and better estimation methods may exist that improve MIA efficiency.

Issue 3.2. *Among the existing pdf estimation methods, which one optimizes the efficiency of MIA?*

We deal with this issue in Chapter 5 where we apply several classical pdf estimation methods and draw conclusions about their efficiencies in the context of MIA.

3.5.3 Likelihood

The likelihood distinguisher is used in the context of profiled attacks *i.e.* attacks where the adversary owns a profile of the side channel leakage. Such a profile consists of a set of estimations for the pdfs $(l \mapsto \mathrm{P}[L = l|Z = z])_{z \in \mathcal{Z}}$. In practice, these estimations are obtained through a profiling stage on a physical implementation identical to the targeted one (except the secret key) and that is under the attacker's control. This approach was first proposed by Suresh Chari, Josyula Rao and Pankaj Rohatgi who introduced the so-called *template attacks* [66]. Several variants and improvements followed (see for instance [18, 28, 201]); all these attacks are usually referred to as profiled attacks.

In a profiled attack, the attacker estimates the *likelihood* of a key guess k, *i.e.* the probability that K equals k, given the leakage measurements vector $\mathbf{l}$ and the inputs vector $\mathbf{x}$. Let $\mathbf{L}$ and $\mathbf{X}$ be the random variables representing the leakage measurements vector and the inputs vector. From Bayes' Theorem, we have:

$$\mathrm{P}[K = k|\mathbf{L} = \mathbf{l}, \mathbf{X} = \mathbf{x}] = \frac{\mathrm{P}[\mathbf{L} = \mathbf{l}|K = k, \mathbf{X} = \mathbf{x}]\,\mathrm{P}[K = k]}{\mathrm{P}[\mathbf{L} = \mathbf{l}|\mathbf{X} = \mathbf{x}]} \ . \tag{3.18}$$

Assuming that K is uniformly distributed, the previous probability satisfies:

$$\mathrm{P}[K = k|\mathbf{L} = \mathbf{l}, \mathbf{X} = \mathbf{x}] = \alpha\,\mathrm{P}[\mathbf{L} = \mathbf{l}|K = k, \mathbf{X} = \mathbf{x}] \ , \tag{3.19}$$

where α is constant towards k. Since the different leakage measurements l_i's are assumed to be independently drawn, we finally obtain:

$$\mathrm{P}\left[K = k | \mathbf{L} = \mathbf{l}, \mathbf{X} = \mathbf{x}\right] = \alpha \prod_{i=1}^{N} \mathrm{P}\left[L = l_i | Z = f(x_i, k)\right] \ . \tag{3.20}$$

For computational reasons, one usually processes the logarithm of the estimated likelihood and averages it on the number of leakage measurements. Moreover, since α is constant with respect to k, it is usually ignored. On the whole, one computes the *log-likelihood* λ_k defined by:

$$\lambda_k = \frac{1}{N} \sum_{i=1}^{N} \log \widehat{\mathrm{P}}\left[L = l_i | Z = f(x_i, k)\right] \ , \tag{3.21}$$

where $\widehat{\mathrm{P}}\left[\cdot\right]$ stands for the pdf estimations.

Profiling stage. The first profiling method was described in the original paper by Suresh Chari *et al.* [66]. The Gaussian noise assumption was introduced under which the pdfs $\mathrm{P}\left[L = l | Z = z\right]$ are Gaussian pdfs with parameters (m_z, Σ_z). Therefore the method consists in estimating the expectation m_z and covariance matrix Σ_z for every $z \in \mathcal{Z}$. The practical issue posed by this approach is the selection of the so-called *points of interest*. As explained above, the leakage corresponding to a cryptographic computation can be seen as a T-size vector corresponding to T points in time and depending more or less on the target variable Z. For typical values of T (*e.g.* $\sim 10^4$), the estimation of the $T \times T$ covariance matrix Σ_z is impractical. Therefore, one must restrict the leakage profile to some well chosen coordinates among $\{1, \ldots, T\}$. In [66], a heuristic is proposed that consists in selecting the points where the variance of $(\widehat{m}_z)_{z \in \mathcal{Z}}$ is maximal. In [28], this method is improved by using *principal component analysis* (see for instance [129]): a few linear combinations of the leakage points are selected that catch the main variability of $(\widehat{m}_z)_{z \in \mathcal{Z}}$. The limitation of these approaches is that the leakage noise is not taken into account. An alternative method is proposed in [213] that overcomes this problem for small-sized leakage. Independently of the points selection issue, an improved estimation method is proposed in [201] which is based on the IBL assumption and on the independent noise assumption (see Section 3.4.3). The IBL assumption enables the use of *linear regression* (see for instance [88]) for the means estimation which renders it more efficient. The independent noise assumption implies that a single covariance matrix needs to be estimated which also improves the profiling efficiency [106, 215].

Soundness. It can be checked that the expected value of the log-likelihood λ_k satisfies:

$$\mathrm{E}\left[\lambda_k\right] = \sum_{\substack{l \in \mathcal{L} \\ x \in \mathcal{X}}} \mathrm{P}\left[X = x\right] \mathrm{P}\left[L = l | Z = f(x, k^*)\right] \log \left(\widehat{\mathrm{P}}\left[L = l | Z = f(x, k)\right] \right) \ . \tag{3.22}$$

Assuming the pdf estimations are perfect (*i.e.* $\widehat{\mathrm{P}}\left[\cdot\right] = \mathrm{P}\left[\cdot\right]$), it can be checked that this expression is maximal for $k = k^*$ which demonstrates the soundness of the likelihood distinguisher. Moreover, for good pdf estimations, the likelihood distinguisher is not only sound but it is also optimal since it computes the exact probability that the secret variable K equals the different key guesses given the observed leakage samples.

Note that bad estimations may significantly decrease the attack efficiency. Although this issue did not receive a lot of attention so far, it would be interesting to study how the likelihood distinguisher tolerates bad estimations. Also, since the estimations are usually obtained from a copy device under the attacker control, a legitimate question to ask is: to what extent do different copies of the same cryptographic implementation have identical leakage distributions? Only the answers to these questions would give a clear view of the practicability of profiling side channel attacks.

3.5.4 Discussion

We have described above three distinguishers that can be used for side channel key recovery attacks. We will now argue that the choice of the distinguisher to use mainly depends on the attacker knowledge about the leakage, namely about the leakage function and the noise distribution.

The more generic distinguisher is the mutual information since it can be used without any knowledge about the leakage. Indeed, targeting a "non-injective" sensitive variable and using as model its predicted value is sufficient to obtain a sound key recovery attack (see Section 3.5.2). However, the genericity offered by MIA is not very useful in practice since the leakage often depends linearly of the Hamming weight of processed data or at least on the different bits (see the IBL assumption, Section 3.4.3). This implies that correlation-based distinguishers can be used since they only require a model that is linearly correlated to the leakage. As a matter of fact, in a common context, DPA attacks are more efficient than MIA attacks (see Chapter 5 for experimental results). On the other hand, likelihood-based attack required a high degree of knowledge about the leakage, since they need precise estimations of the leakage function and of the noise distribution. Besides, for sound estimations, they give the best results (see for instance [175, 214]).

3.6 Countermeasures to side channel analysis

We review hereafter the most common countermeasures to side channel key recovery attacks. Most of these countermeasures aim at removing (or at least lowering) the dependence between the leakage and the processed sensitive variables.

3.6.1 Hardware modules

A first classical countermeasure consists in adding some noise to the leakage. This can be ensured by the use of a *noise generator* that is a hardware module randomly consuming power. The main drawback of such an approach is that it implies a significant overhead of power consumption which is prohibitive in some situations (*e.g.* mobile telephony). Another approach is to use a *filter* to flatten the power consumption [80, 149, 162]. However such a module has no effect on the electromagnetic leakage. Another widely used solution is the insertion of *random process interrupts* that provoke random delays in the execution and spread the sensitive signal over several times [74]. All these approaches decrease the leakage signal-to-noise ratio. However they are usually not sufficient to completely counteract practical attacks.

3.6.2 Secure logic styles

Recently another approach has been intensively investigated that focuses on the design of secure logic styles. In this approach, every single gate constituting the circuit is secured. Two main approaches exist to secure a logic gate: *dual-rail* and *masking*.

Dual-rail logic [69, 121, 212, 231, 232] aims to render the power consumption constant at each clock cycle independently of the processed data. For such a purpose, each wire is doubled and the circuit is designed in such a way that, at each clock cycle, a transition occurs on one and only one wire of each pair. As a result, a constant number of transitions occur at each clock cycle. This is expected to render the power consumption constant. However, small variations of the power consumption are observed in practice due to small loading imbalances between associated wires [69]. In the electromagnetic leakage, this imbalance may even be induced by the attacker who can set the position of the measurement probe. Dual-rail may also be defeated when input signals of a gate arrive at different times [223].

Masking at the gate level consists in randomizing each logical signal in such a way that a transition always occur with a probability $\frac{1}{2}$ (see for instance [224, 233]). For such a purpose, each logical value v is represented by a masked value $v \oplus r$ for a randomly generated r called *the mask*. The masked values leaking by side channels are uncorrelated to the significant data which *a priori* prevents SCA. Unfortunately, this approach suffers multiple flaws in practice. First, the occurrence of *glitches*, which is inherent to electronic circuits, renders the average number of transitions dependent on the unmasked

data [158, 159]. To overcome this weakness a logic style combining dual-rail and masking was proposed in [190]. However, this approach has been shown to be ineffective due to the so-called *early propagation effect* [150, 189, 223]. A patch was proposed in [189] at the price of a significant overhead in silicium area and power consumption. Another approach to thwart glitches has been investigated in [174, 225] which is based on *secret sharing* [207]. However, even if glitches and early propagation are avoided, another problem remains: a mask r is used for several (usually all) logic gates within a clock cycle. This enables a so-called *pdf attack* as described in [200, 230]. Using a different mask for each logic gate at each clock cycle would require a huge generation of random bits which is clearly impractical. Finally, masking has an inherent weakness: since masks are generated and handled by the device, they also leak information. The leaked information about the masked data together with the leaked information about the masks jointly depend on the unmasked data which enables *higher-order side channel attacks* (see Section 3.7).

To summarize, the design of a perfectly secure logic style is still an open issue. In the meantime, the use of existing logic styles still improve SCA resistance of the circuit by reducing the amount of information leakage [154].

3.6.3 Algorithmic randomization techniques

Algorithmic randomization techniques are included in the design of algorithms. One of the main advantages of such an approach is that it works for both software and hardware implementations. As they are dedicated, these countermeasures concentrate on the sensitive parts of the computation rather than on the whole device such as secure logic styles. Depending on the situation, these techniques may also be called *blinding* or *masking*. The purpose of these countermeasures is to render the processed data independent of predictable intermediate variables.

3.6.3.1 Randomization techniques for block ciphers

For block ciphers implementations, one usually makes use of a *masking scheme*. The principle of such a scheme is to perform the computation is such a way that every sensitive intermediate variable is masked by the addition of a random value. When a d^{th}-order masking scheme is involved, every sensitive variable Z is randomly split into $d+1$ shares $M_0, \ldots, M_d$ in such a way that the relation:

$$M_0 \star \cdots \star M_d = Z \tag{3.23}$$

is satisfied for a group operation $\star$ (*e.g.* the XOR or the modular addition). Usually, d shares $M_1, \ldots, M_d$ (called *the masks*) are randomly picked up and the remaining share M_0 (called *the masked variable*) is computed according to (3.23). This ensures that every set of less than $d+1$ shares among the M_i's is independent of Z. The attacker must then perform a $(d+1)^{th}$*-order side channel attack* by exploiting simultaneously the leakage signals $L_0, \ldots, L_d$ resulting from the manipulation of the $d+1$ shares $M_0, \ldots, M_d$. According to the analysis in [65], the complexity of such attacks increases exponentially with d, which makes masking a sound countermeasure.

The design and cryptanalysis of masking schemes are studied in the second part of this thesis.

3.6.3.2 Randomization techniques for RSA

An RSA computation $s = m^d \bmod N$ can be randomized in various ways due to its strong mathematical structure. Let r and r' denote random numbers, we have:

$$\begin{aligned} s &= m^{d+r\cdot\varphi(N)} \bmod N \\ &= m^{d-r} \cdot m^r \bmod N \\ &= (m \cdot r^{-1})^d \cdot r^d \bmod N \\ &= \left(m^d + r' \cdot N \bmod (r \cdot N)\right) \bmod N \ . \end{aligned}$$

Each of these expressions gives rise to a possible randomization of an RSA computation. In addition, these techniques can be combined. Further details about SCA countermeasures for RSA can be found in [70, 76, 128, 166].

3.6.4 Software desynchronization

Desynchronization may also be included in software. An efficient way to do that is by using *operations shuffling* [124, 229]. The principle is to perform, as far as possible, the different operations in a random order. Shuffling can be enhanced by the addition of dummy operations in order to increase the degree of randomization. Another similar countermeasure consists in inserting random delays. This is usually done by dummy loops performing a random number of iterations. Recently, improved methods have been proposed for the insertion of random delays in software [79, 237]. All these countermeasures result in a decrease of the leakage signal-to-noise ratio. However using advanced DPA attacks, their effect may be mitigated [74].

3.6.5 Protocol level countermeasures

Countermeasures against SCA may also be included at the protocol level. As mentioned by Paul Kocher *et al.* in [148], frequent key update procedures may prevent side channel attacks. Related approaches have been followed in [184, 187]. To date, the protocols of most applications are designed without consideration for SCA. This is unfortunate since protocol level countermeasures seem promising and they would certainly lead to significant gains in efficiency.

3.7 Higher-order side channel analysis

Higher-order side channel analysis was first introduced by Paul Kocher, Joshua Jaffe, and Benjamin Jun in [148] as attacks "that combine multiple samples from within a trace". It was then more formally defined by Thomas Messerges in [164]: "A d^{th}-order DPA attack makes use of d different samples in the power consumption signal that correspond to d different intermediate values calculated during the execution of an algorithm". Today, higher-order SCA usually refer to side channel attacks targeting masked implementations [2, 23, 176, 183]. This terminology shall be used in the present thesis.

Let us consider a masking scheme in which a sensitive variable Z is split into $d+1$ shares M_0, ..., M_d satisfying (3.23) and which produce $d+1$ leakage signals L_0, ..., L_d. As explained hereafter, the classical side channel distinguishers presented in Section 3.5 can be extended to perform a $(d+1)^{\text{th}}$-order attack exploiting the L_i's.

3.7.1 Higher-order differential power analysis

We call higher-order differential power analysis (HO-DPA) any higher-order SCA using a correlation distinguisher. Since the correlation is a univariate operator, the attacker must combine the $d+1$ leakage signals L_0, ..., L_d in order to create a univariate signal that is correlated to the target variable. This is done by means of a *combining function* $\mathcal{C}$. The attack then consists in estimating the correlation between the *combined leakage* $\mathcal{C}(L_0, \ldots, L_d)$ and the model $\mathsf{M}(X, k) = \hat{\varphi} \circ f(X, k)$ for every key guesses $k \in \mathcal{K}$.

Several combining functions have been proposed in the literature. Two of them are commonly used: the *product combining* suggested by Suresh Chari, Charanjit Jutla, Josyula Rao, and Pankaj Rohatgi in [65] which consists in multiplying the different signals:

$$\mathcal{C}_{prod}(L_0, \ldots, L_d) = L_0 \times \cdots \times L_d \ , \tag{3.24}$$

and the *absolute difference combining* proposed by Thomas Messerges in [164] which computes the absolute value of the difference between two signals:

$$\mathcal{C}_{diff}(L_0, L_1) = |L_0 - L_1| \ . \tag{3.25}$$

The latter can be extended to higher orders by induction:

$$\mathcal{C}_{diff}(L_0, \ldots, L_d) = |\cdots||L_0 - L_1| - L_2|\cdots - L_d| \ . \tag{3.26}$$

Other combining functions have been proposed in [133, 175]. All these methods are sound to perform HO-DPA, however it is not clear which of them is the most efficient.

Issue 3.3. *Among the existing combining functions, which one leads to the most efficient HO-DPA?*

Once a combining function has been chosen, a remaining issue is the choice of the prediction function $\hat{\varphi}$ optimizing the attack.

Issue 3.4. *Given a combining function $\mathcal{C}$, which prediction function $\hat{\varphi}$ optimizes the attack efficiency?*

The two issues above are addressed in Chapter 6. First we give the optimal prediction function according to any combining function and according to the leakage model. Afterwards, we study existing combining functions for second-order DPA in the Hamming weight model. Our analysis allows us to exhibit an improved product combining that is shown to be better than the other existing combining functions. Afterwards we analyze the improved product combining at any order.

3.7.2 Higher-order mutual information analysis

Contrary to the correlation, the mutual information is a multivariate operator. As a result, it can be directly involved in performing a higher-order MIA (HO-MIA). One just estimates the mutual information between the prediction $\hat{\varphi} \circ f(X, k)$ and the $(d+1)$-tuple of leakages $(L_1, \ldots, L_d)$. The drawback of HO-MIA is that it relies on pdf estimations (or at least entropy estimations) whose complexity grows exponentially with the leakage dimension. As a result, it may quickly becomes impractical as d increases.

HO-MIA is discussed in more detail in Chapter 5: we address its theoretical and practical aspects and we provide several experimental results.

3.7.3 Higher-order profiled attacks

The likelihood distinguisher can also be straightforwardly extended to higher orders. The principle remains the same: estimate the probability $\mathrm{P}\left[K = k | (\mathbf{l}, \mathbf{x})\right]$. Here each leakage measurements l_i is composed of $d+1$ (possibly multivariate) leakages: $l_i = (l_{i,0}, \ldots, l_{i,d})$ such that $l_{i,j}$ corresponds to the leakage of the j^{th} share in the i^{th} leakage measurements. Similarly to the first-order case, we have:

$$\mathrm{P}\left[K = k | (\mathbf{l}, \mathbf{x})\right] = \alpha \prod_{i=1}^{N} \mathrm{P}\left[L_0 = l_{i,0}, \ldots, L_d = l_{i,d} | Z = f(x_i, k)\right] \ . \tag{3.27}$$

Assuming that the leakage noises of each share are mutually independent, we further have:

$$\mathrm{P}\left[K = k | (\mathbf{l}, \mathbf{x})\right] = \alpha \prod_{i=1}^{N} \left(\sum_{\mathbf{m} \in \mathcal{Z}^d} \mathrm{P}\Big[L_0 = l_{i,0} | f(x_i, k) = M_0 \star m_1 \star \cdots \star m_d\Big] \cdot \prod_{j=1}^{d} \mathrm{P}\left[L_j = l_{i,j} | M_j = m_j\right] \right) \ .$$

Some works have described higher-order profiled attacks with an *a priori* known leakage model [175, 183]. However in the general case the pdfs $\mathrm{P}\left[L_i | M_i\right]$ must be estimated as for first order profiled attacks. The profiling efficiency then depends on whether or not the adversary controls the masks values on the profiled device. If the adversary controls the M_i values (or at least knows them), he can apply an estimation method for each $\mathrm{P}\left[L_i | M_i\right]$ just as in the first order case (see Section 3.5.3). Without controlling the M_i values, the profiling is much harder. In the Gaussian model, this amounts to estimating Gaussian mixture pdfs which is usually done *via* an *expectation maximization algorithm* [152].

Chapter 4

Success rate of side channel analysis in the Gaussian model

Contents

4.1 Introduction

In this chapter, we investigate the issue of evaluating the success rate of side channel analysis in the widely admitted Gaussian leakage model, *i.e.* under the Gaussian noise assumption. We introduce a new approach that allows us to efficiently compute the success rate of an attack using either the correlation or the likelihood as distinguisher.

The results presented in this chapter have been published in the international workshop on *Selected Areas in Cryptography (SAC 2008)* [7].

4.2 Approach

Under the Gaussian noise assumption, the leakage related to the computation of a given value $z \in \mathcal{Z}$ has a Gaussian distribution $\mathcal{N}(m_z, \Sigma_z)$. That is, the leakage function and the noise are such that $\varphi(z) = m_z$ and $(B|Z = z) \sim \mathcal{N}(0, \Sigma_z)$. In order to determine the exact success rate of an attack,

we must investigate the multivariate probability distribution of the distinguishing vector $(d_k)_{k\in\mathcal{K}}$. This distribution can be expressed with respect to the inputs vector $\mathbf{x}$, the secret key k^* and the leakage distribution parameters $(m_z, \Sigma_z)_{z\in\mathcal{Z}}$. We will show that under the Gaussian assumption, the multivariate distribution of the distinguishing vector is (or at least can be precisely approximated by) a multivariate Gaussian distribution. This will enable us to show how the success rate of such attacks can be efficiently computed.

For clarity and without ambiguity, we shall respectively denote by m_{x,k^*} and Σ_{x,k^*} the mean vector $m_{f(x,k^*)}$ and the covariance matrix $\Sigma_{f(x,k^*)}$. We will further denote by τ_x the occurrence ratio of an element $x \in \mathcal{X}$ through the inputs vector $\mathbf{x}$, *i.e.* :

$$\tau_x = \frac{|\{i; x_i = x\}|}{N} \ . \tag{4.1}$$

4.3 Correlation distinguisher

4.3.1 Distribution

We investigate hereafter the distribution of the correlation distinguishing vector $(\rho_k)_{k\in\mathcal{K}}$ such as defined by (3.14). Since the correlation is an univariate distinguisher, we investigate the distribution of this distinguisher with respect to univariate leakage measurements $l_i \sim \mathcal{N}\left(m_{x_i,k^*}, \sigma^2_{x_i,k^*}\right)$.

Let us first denote by $\overline{\mathsf{M}}_k$ and $\widehat{\sigma}_k$ the mean and the standard deviation of the prediction vector $\big(\mathsf{M}(x_i,k)\big)_i$, namely:

$$\overline{\mathsf{M}}_k = \sum_{x\in\mathcal{X}} \tau_x \mathsf{M}(x,k) \quad \text{and} \quad \widehat{\sigma}_k^2 = \sum_{x\in\mathcal{X}} \tau_x \left(\mathsf{M}(x,k) - \overline{\mathsf{M}}_k\right)^2 \ .$$

Instead of focusing on ρ_k, we focus in the sequel on the following coefficient:

$$\dot{\rho}_k = \frac{1}{\widehat{\sigma}_k N} \sum_{i=1}^{N} \left(\mathsf{M}(x_i,k) - \overline{\mathsf{M}}_k\right) l_i \ . \tag{4.2}$$

The distribution of $(\dot{\rho}_k)_{k\in\mathcal{K}}$ is indeed more convenient to analyze than the one of $(\rho_k)_{k\in\mathcal{K}}$. Moreover, one can verify that the ratio $\dot{\rho}_k/\rho_k$ equals the standard deviation of the leakage measurement vector $\mathbf{l}$. Consequently, $\dot{\rho}_k/\rho_k$ is positive and constant with respect to the key guess k. As a result,

$$\underset{k\in\mathcal{K}}{\text{argmax-}o}\ \rho_k = \underset{k\in\mathcal{K}}{\text{argmax-}o}\ \dot{\rho}_k$$

holds for every k and hence, the success rate of the attack is fully determined by the distribution of the vector $(\dot{\rho}_k)_{k\in\mathcal{K}}$. The next proposition provides us with the exact distribution of this vector.

Proposition 4.1. *The vector $(\dot{\rho}_k)_{k\in\mathcal{K}}$ has a multivariate Gaussian distribution whose expectation satisfies for every $k \in \mathcal{K}$:*

$$\mathrm{E}\left[\dot{\rho}_k\right] = \frac{1}{\widehat{\sigma}_k} \sum_{x\in\mathcal{X}} \tau_x \left(\mathsf{M}(x,k) - \overline{\mathsf{M}}_k\right) m_{x,k^*} \ , \tag{4.3}$$

and whose covariance satisfies for every $(k_1,k_2) \in \mathcal{K}^2$:

$$\mathrm{Cov}\left[\dot{\rho}_{k_1}, \dot{\rho}_{k_2}\right] = \frac{1}{N\widehat{\sigma}_{k_1}\widehat{\sigma}_{k_2}} \sum_{x\in\mathcal{X}} \tau_x \left(\mathsf{M}(x,k_1) - \overline{\mathsf{M}}_{k_1}\right)\left(\mathsf{M}(x,k_2) - \overline{\mathsf{M}}_{k_2}\right)\sigma^2_{x,k^*} \ . \tag{4.4}$$

Proof. Since the l_i's are drawn from Gaussian distributions $\mathcal{N}\left(m_{x_i,k^*}, \sigma_{x_i,k^*}\right)$ and since the vector $(\dot{\rho}_k)_{k\in\mathcal{K}}$ is a linear transformation of $\mathbf{l}$, one deduces that $(\dot{\rho}_k)_{k\in\mathcal{K}}$ has a multivariate Gaussian distribution.

Now, for every $x \in \mathcal{X}$, we have $N\tau_x$ elements among the x_i's that are equal to x. This, together with (4.2) immediately leads to (4.3). Then, the mutual independence of the l_i's and the bilinearity of the covariance imply (4.4). □

Proposition 4.1 gives the exact distribution of the distinguishing vector $(\dot{\rho}_k)_{k\in\mathcal{K}}$. This makes it possible to precisely compute the success rate of a side channel attack that involves the Pearson correlation coefficient (see Section 4.5).

It is worth noting that the distribution of $(\dot{\rho}_k)_{k\in\mathcal{K}}$ does not fully depend on the inputs vector $\mathbf{x}$ but only on the different ratios τ_x's. A common choice, for a chosen plaintext SCA, is to set these ratios at $\tau_x = 1/|\mathcal{X}|$. For a known plaintext/ciphertext SCA, assuming that the x_i's are uniformly drawn, we further have $\tau_x \approx 1/|\mathcal{X}|$ for N large enough. We investigate this particular setting hereafter.

4.3.2 Distribution in the uniform setting

We investigate here the setting where the x_i's are chosen such that $\tau_x = 1/|\mathcal{X}|$ holds for every x. We further assume that the target variable Z can be expressed as $Z = f'(X \oplus K)$ where f' is a balanced function (*i.e.* the cardinal of $f'^{-1}(z)$ is constant for every $z \in \mathcal{Z}$).

In the uniform setting, the previous study can be simplified. In this setting, the mean and the standard deviation of the prediction vector are constant with respect to k^*. Indeed, for every $k \in \mathcal{K}$, we have:

$$\overline{\mathsf{M}}_k = \frac{1}{|\mathcal{Z}|}\sum_{z\in\mathcal{Z}} \mathsf{M}(z) \quad \text{and} \quad \widehat{\sigma}_k = \sqrt{\frac{1}{|\mathcal{Z}|}\sum_{z\in\mathcal{Z}} \left(\mathsf{M}(z) - \overline{\mathsf{M}}\right)^2} \ .$$

Hence, we can focus on the following coefficient:

$$\ddot{\rho}_k = \frac{1}{N}\sum_{i=1}^{N} \mathsf{M}(x_i, k) l_i \ . \tag{4.5}$$

Once again $\ddot{\rho}_k/\rho_k$ is positive and constant with respect to k which implies that focusing on ρ_k instead of $\ddot{\rho}_k$ does not affect the success rate of the attack. The following corollary gives the distribution of $(\ddot{\rho}_k)_{k\in\mathcal{K}}$.

Corollary 4.1. *The vector $(\ddot{\rho}_k)_{k\in\mathcal{K}}$ has a multivariate Gaussian distribution whose expectation satisfies for every $k \in \mathcal{K}$:*

$$\mathrm{E}\left[\ddot{\rho}_k\right] = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} \mathsf{M}(x,k) m_{x,k^*} \ , \tag{4.6}$$

and whose covariance satisfies for every $(k_1, k_2) \in \mathcal{K}^2$:

$$\mathrm{Cov}\left[\ddot{\rho}_{k_1}, \ddot{\rho}_{k_2}\right] = \frac{1}{N|\mathcal{X}|}\sum_{x\in\mathcal{X}} \mathsf{M}(x,k_1)\mathsf{M}(x,k_2)\sigma^2_{x,k^*} \ . \tag{4.7}$$

Proof. Corollary 4.1 straightforwardly holds from Proposition 4.1 by setting $\overline{\mathsf{M}}_k$ to 0 and $\widehat{\sigma}_k$ to 1. □

An interesting property of the uniform setting is stressed in the following proposition.

Proposition 4.2. *Let $(d_k)_{k\in\mathcal{K}}$ and $(d'_k)_{k\in\mathcal{K}}$ be the distributions of the vector $(\ddot{\rho}_k)_{k\in\mathcal{K}}$ for two secret keys $k_1^* \in \mathcal{K}$ and $k_2^* \in \mathcal{K}$ respectively. In the uniform setting, the distributions $(d_{k\oplus k_1^*})_{k\in\mathcal{K}}$ and $(d'_{k\oplus k_2^*})_{k\in\mathcal{K}}$ are indentical.*

Proof. In the uniform setting, we have $\mathsf{M}(x,k) = \mathsf{M}(f'(x\oplus k))$ and $m_{x,k^*} = m_{f'(x\oplus k^*)}$. Hence, from (4.6) we get $\mathrm{E}\left[d_{k\oplus k_1^*}\right] = \mathrm{E}\left[d'_{k\oplus k_2^*}\right]$ for every $k \in \mathcal{K}$ and from (4.7) we get $\mathrm{Cov}\left[d_{k_1\oplus k_1^*}, d_{k_2\oplus k_1^*}\right] = \mathrm{Cov}\left[d'_{k_1\oplus k_2^*}, d'_{k_2\oplus k_2^*}\right]$ for every $(k_1, k_2) \in \mathcal{K}^2$. Finally, since $(d_k)_{k\in\mathcal{K}}$ and $(d'_k)_{k\in\mathcal{K}}$ are both Gaussian then they are identical. □

Proposition 4.2 shows that the vector $(\ddot{\rho}_{k\oplus k^*})_{k\in\mathcal{K}}$ has the same distribution for every k^*. Moreover, the event $k^* \in \text{argmax-}o_{k\in\mathcal{K}}\ \ddot{\rho}_k$ can be rewritten as $0 \in \text{argmax-}o_{k\in\mathcal{K}}\ \ddot{\rho}_{k\oplus k^*}$. Since the distribution of $(\ddot{\rho}_{k\oplus k^*})_{k\in\mathcal{K}}$ is independent of k^*, we get that, in the uniform setting, the success rate is constant with respect to k^*. Therefore, one only needs to analyze the distribution of $(\ddot{\rho}_{k\oplus k^*})_{k\in\mathcal{K}}$ for a given secret key (*e.g.* for $k^* = 0$) to get the distribution and the success rate of $(\ddot{\rho}_k)_{k\in\mathcal{K}}$ for any secret key k^*.

4.4 Likelihood distinguisher

4.4.1 Likelihood in the Gaussian model

Under the Gaussian noise assumption, every leakage pdf $l \mapsto \mathrm{P}[L = l|Z = z]$ is assumed to be the Gaussian pdf ϕ_{m_z,Σ_z}. Estimating such a pdf amounts to estimating the parameters (m_z, Σ_z) for every $z \in \mathcal{Z}$. In the following, we shall denote the attacker estimations of (m_z, Σ_z) by $(\widehat{m}_z, \widehat{\Sigma}_z)$. For clarity and without ambiguity, the estimated parameters $\widehat{m}_{f(x,k)}$ and $\widehat{\Sigma}_{f(x,k)}$ are further denoted by $\widehat{m}_{x,k}$ and by $\widehat{\Sigma}_{x,k}$. That is, the estimated pdf $\widehat{\mathrm{P}}[L = \cdot|Z = f(x,k)]$ satisfies for every $l \in \mathcal{L}$:

$$\widehat{\mathrm{P}}[L = l|Z = f(x,k)] = \frac{1}{\sqrt{(2\pi)^T|\widehat{\Sigma}_{x,k}|}} \exp\left(-\frac{1}{2}(x - \widehat{m}_{x,k})' \widehat{\Sigma}_{x,k}^{-1} (x - \widehat{m}_{x,k})\right) ,$$

According to (3.21), the likelihood λ_k then satisfies:

$$\lambda_k = \frac{1}{2N} \sum_{i=1}^{N} \left(\log\left((2\pi)^T|\widehat{\Sigma}_{x_i,k}|\right) - (l_i - \widehat{m}_{x_i,k})' \widehat{\Sigma}_{x_i,k}^{-1} (l_i - \widehat{m}_{x_i,k})\right) . \tag{4.8}$$

4.4.2 Distribution

We investigate hereafter the distribution of the likelihood distinguishing vector $(\lambda_k)_{k\in\mathcal{K}}$ such as defined by (4.8).

Let us first introduce a few notations. The element of the i^{th} row and of the j^{th} column of a matrix A is denoted by $A[i,j]$ while the i^{th} element of a vector V is denoted by $V[i]$. A' denotes the transpose of a matrix (or a vector) A. The notation $\|\cdot\|$ is used to denote the Euclidian norm while the notation $\|\cdot\|_{hs}$ refers to the Hilbert-Schmidt matrix norm defined by $\|A\|_{hs} = \sqrt{\sum_{i,j} A[i,j]^2}$. We shall further denote by A^2 the product $A'A$ and by $A^{-1/2}$ any matrix satisfying $(A^{-1/2})'A^{-1/2} = A$ (*e.g.* the Cholesky decomposition matrix). Finally the trace of A is denoted by $\mathrm{Tr}(A)$.

The next proposition provides a precise approximation of the distribution of the likelihood vector $(\lambda_k)_{k\in\mathcal{K}}$.

Proposition 4.3. *The distribution of the vector $(\lambda_k)_{k\in\mathcal{K}}$ tends toward a multivariate Gaussian distribution as N grows. Moreover, for every $k \in \mathcal{K}$, the expectation of λ_k satisfies:*

$$\mathrm{E}[\lambda_k] = \frac{1}{2} \sum_{x\in\mathcal{X}} \tau_x \Big(\log\left(2\pi|\widehat{\Sigma}_{x,k}|\right) - \left\|\widehat{\Sigma}_{x,k}^{-1/2} (m_{x,k^*} - \widehat{m}_{x,k})\right\|^2 - \mathrm{Tr}\Big(\widehat{\Sigma}_{x,k}^{-1/2} \Sigma_{x,k^*} \left(\widehat{\Sigma}_{x,k}^{-1/2}\right)'\Big)\Big) , \tag{4.9}$$

and for every $(k_1, k_2) \in \mathcal{K}^2$, the covariance between λ_{k_1} and λ_{k_2} satisfies:

$$\mathrm{Cov}[\lambda_{k_1}, \lambda_{k_2}] = \frac{1}{N} \sum_{x\in\mathcal{X}} \tau_x \Big(\frac{1}{2} \left\|\widehat{\Sigma}_{x,k_1}^{-1/2} \Sigma_{x,k^*} (\widehat{\Sigma}_{x,k_2}^{-1/2})'\right\|_{hs}^2 + (m_{x,k^*} - \widehat{m}_{x,k_1})' \widehat{\Sigma}_{x,k_1}^{-1} \Sigma_{x,k^*} \widehat{\Sigma}_{x,k_2}^{-1} (m_{x,k^*} - \widehat{m}_{x,k_2}) \Big) . \tag{4.10}$$

The proof of Proposition 4.3 makes use of the following lemma.

Lemma 4.1. *Let X be a T-size random vector having a Gaussian distribution $\mathcal{N}(0, \Sigma)$. Let A_1 and A_2 be two $(T \times T)$-matrices and let m_1 and m_2 be two T-size vectors. Let Q_1 and Q_2 be two quadratic forms defined, for $j = 1, 2$, by $Q_j = (X + m_j)' A_j^2 (X + m_j)$. For $j = 1, 2$, the expectation of Q_j satisfies:*

$$\mathrm{E}[Q_j] = \|A_j m_j\|^2 + \mathrm{Tr}(A_j \Sigma A_j') . \tag{4.11}$$

And the covariance of Q_1 and Q_2 satisfies:

$$\mathrm{Cov}[Q_1, Q_2] = 2\|A_1 \Sigma A_2'\|_{hs}^2 + 4\, m_1' A_1^2 \Sigma A_2^2 m_2 . \tag{4.12}$$

Proof. We have $Q_j = \sum_{i=1}^{T} (A_j\,(X + m_j))[i]^2$ which leads to:

$$\begin{aligned} \mathrm{E}\,[Q_j] &= \sum_{i=1}^{T} \mathrm{E}\left[(A_j\,(X+m_j))[i]^2\right] && (4.13)\\ &= \sum_{i=1}^{T} \mathrm{E}\,[(A_j\,(X+m_j))[i]]^2 + \sum_{i=1}^{T} \mathrm{Var}\,[(A_j\,(X+m_j))[i]]\ , && (4.14)\end{aligned}$$

since $\mathrm{E}\left[Y^2\right] = \mathrm{Var}\,[Y] + \mathrm{E}\,[Y]^2$ holds for every random variable Y. From $(X + m_j) \sim \mathcal{N}\,(m_j, \Sigma)$ we have $A_j\,(X + m_j) \sim \mathcal{N}\left(A_j\,m_j, A_j\,\Sigma\,A_j'\right)$ which directly yields (4.11).

The quadratic form Q_j can be rewritten as $Q_j = (A_j\,X)^2 + (A_j\,m_j)^2 + 2m_j'\,A_j^2\,X$ for $j = 1, 2$. By bilinearity, $\mathrm{Cov}\,[Q_1, Q_2]$ satisfies:

$$\begin{aligned}\mathrm{Cov}\,[Q_1, Q_2] = \mathrm{Cov}\left[(A_1\,X)^2,\ (A_2\,X)^2\right] + 2\ \mathrm{Cov}\left[(A_1\,X)^2,\ m_2'\,A_2\,X\right]\\ + 2\ \mathrm{Cov}\left[(A_2\,X)^2,\ m_1'\,A_1\,X\right] + 4\ \mathrm{Cov}\,[m_1'\,A\,X,\ m_2'\,A\,X]\ . \quad (4.15)\end{aligned}$$

We claim the three following relations:

$$\mathrm{Cov}\left[(A_1\,X)^2, (A_2\,X)^2\right] = 2\,\|A_1\,\Sigma\,A_2'\|_{hs}^2\ , \quad (4.16)$$

$$\mathrm{Cov}\left[(A_1\,X)^2, m_2'\,A_2^2\,X\right] = \mathrm{Cov}\left[(A_2\,X)^2, m_1'\,A_1^2\,X\right] = 0\ , \quad (4.17)$$

$$\mathrm{Cov}\left[m_1'\,A_1^2\,X,\ m_2'\,A_2^2\,X\right] = m_1'\,A_1^2\,\Sigma\,A_2^2\,m_2\ . \quad (4.18)$$

These relations together with (4.15) result in (4.12) and state the correctness of Lemma 4.1. Relation (4.18) straightforwardly holds from the bilinearity of the covariance and by symmetry of A_1^2 (*i.e.* $(A_1^2)' = A_1^2$). Relations (4.16) and (4.17) are stated hereafter.

First, let us show (4.16). The covariance between $(A_1\,X)^2$ and $(A_2\,X)^2$ can be rewritten as:

$$\begin{aligned}\mathrm{Cov}\left[(A_1\,X)^2, (A_2\,X)^2\right] &= \sum_{i,j} \mathrm{Cov}\left[(A_1\,X)[i]^2, (A_2\,X)[j]^2\right] && (4.19)\\ &= \sum_{i,j} \left(\mathrm{E}\left[(A_1\,X)[i]^2 (A_2\,X)[j]^2\right] - \mathrm{E}\left[(A_1\,X)[i]^2\right] \mathrm{E}\left[(A_2\,X)[j]^2\right]\right) && (4.20)\end{aligned}$$

Since the expectations of $A_1\,X$ and $A_2\,X$ equal zero, the expectation of the product $(A_1\,X)[i]^2 (A_2\,X)[j]^2$ is the Gaussian forth order moment that is known to satisfy:

$$\mathrm{E}\left[(A_1\,X)[i]^2 (A_2\,X)[j]^2\right] = \mathrm{E}\left[(A_1\,X)[i]^2\right] \mathrm{E}\left[(A_2\,X)[j]^2\right] + 2\ \mathrm{Cov}\,[(A_1\,X)[i], (A_2\,X)[j]]^2\ . \quad (4.21)$$

Hence, (4.20) gives:

$$\mathrm{Cov}\left[(A_1\,X)^2, (A_2\,X)^2\right] = 2 \sum_{i,j} \mathrm{Cov}\,[(A_1\,X)[i], (A_2\,X)[j]]^2\ . \quad (4.22)$$

Since we have $\mathrm{Cov}\,[(A_1\,X)[i], (A_2\,X)[j]] = (A_1\,\Sigma\,A_2')[i, j]$, one deduces that (4.22) finally results in (4.16).

We now show the correctness of (4.17). We have:

$$\mathrm{Cov}\left[(A_1\,X)^2,\ m_2'\,A_2^2\,X\right] = \sum_{i} \mathrm{Cov}\left[(A_1\,X)[i]^2,\ m_2'\,A_2^2\,X\right]\ . \quad (4.23)$$

Since X has a zero mean, every term of the previous sum is a Gaussian third order moment and is hence equal to zero. This way, we get (4.17). □

We give hereafter the proof of Proposition 4.3.

Proof. (Proposition 4.3) Since the l_i's are independently drawn from distributions $\mathcal{N}(m_{x_i,k^*}, \Sigma_{x_i,k^*})$ and since, for every x, there is a ratio τ_x of the x_i's that equal x, Relation (4.8) and Lemma 4.1 directly lead to (4.9) and (4.10).

Now, $(\lambda_k)_{k\in\mathcal{K}}$ can be expressed as a linear transformation of the vector $\sum_{i=1}^{N} l_i$ and of the vector $\left(\sum_{i=1}^{N} l_i[j_1]l_i[j_2]\right)_{1\leqslant j_1,j_2\leqslant T}$. The first one has a multivariate Gaussian distribution and, from the multivariate central limit theorem, the second one tends toward a multivariate Gaussian distribution as N grows. Hence $(\lambda_k)_{k\in\mathcal{K}}$ tends toward a multivariate Gaussian distribution as N grows. □

4.4.3 Convergence of the distribution

Proposition 4.3 shows that the distribution of the log-likelihood vector $(\lambda_k)_{k\in\mathcal{K}}$ tends towards a multivariate Gaussian distribution as the number N of leakage measurements grows. We argue hereafter that this convergence is quick meaning that approximating the distribution of $(\lambda_k)_{k\in\mathcal{K}}$ by a multivariate Gaussian is sound.

According to (4.8), the log-likelihood λ_k can be expressed as the sum of $|\mathcal{X}|$ values $\lambda_{k,x}$ that are defined by:

$$\lambda_{k,x} = \frac{\tau_x}{2}\log\left((2\pi)^T|\widehat{\Sigma}_{x,k}|\right) - \frac{1}{2N}\sum_{\substack{i=1\\x_i=x}}^{N}(l_i-\widehat{m}_{x,k})'\,\widehat{\Sigma}_{x,k}^{-1}\,(l_i-\widehat{m}_{x,k})\ . \tag{4.24}$$

The first term is constant and the second term is a sum of $N\tau_x$ elements of the form $X'\,A^2\,X$ where A is the matrix $\widehat{\Sigma}_{x,k}^{-1/2}$ and X is a Gaussian random variable $\mathcal{N}(m_{x,k^*}-\widehat{m}_{x,k}, \Sigma_{x,k^*})$. The distribution of such a sum is given in the following lemma. At first, let us recall that the chi-square distribution with n degrees of freedom $\chi^2(n)$ is the distribution obtained by summing n independent $\mathcal{N}(0,1)$-distributed random variables.

Lemma 4.2. *Let $(X_j)_j$ be n independent T-size random vectors having a Gaussian distribution $\mathcal{N}(m,\Sigma)$, let A be a $(T\times T)$-matrix and let $(Q_j)_j$ be the quadratic forms defined as $Q_j = X_j'\,A^2\,X_j$. The sum of the Q_j satisfies:*

$$\sum_{j=1}^{n} Q_j = \beta + G + \sum_{i=1}^{T}\alpha_i C_i\ , \tag{4.25}$$

where $\beta = n(A\cdot m)^2$, $\alpha_i = (A\,\Sigma\,A')[i,i]$, G is an univariate Gaussian random variable, C_i are T chi-square random variables with n degrees of freedom.

Proof. For $j=1,2$, we have $Q_j = (A\,X_j)^2$. Denoting by $\overline{X}_j$ the centered random variable $X_j - m$, we get $Q_j = (A\,m)^2 + 2\ m'\,A^2\,\overline{X}_j + (A\,\overline{X}_j)^2$ and hence, $\sum_j Q_j = \beta + 2\sum_j m'\,A^2\,\overline{X}_j + \sum_j\sum_i (A\,\overline{X}_j)[i]^2$.

After denoting $2\sum_j m'\,A^2\,\overline{X}_j$ by G and $\frac{1}{\alpha_i}\sum_j (A\,\overline{X}_j)[i]^2$ by C_i, we get (4.25). Now, G is Gaussian since it is defined as a sum of Gaussian random variables. Moreover, the covariance matrix of $A\,X_j$ being equal to $A\,\Sigma\,A'$, we have, for every j: $\alpha_i = \mathrm{Var}\left[(A\,\overline{X}_j)[i]\right]$. This implies that $\frac{1}{\sqrt{\alpha_i}}(A\,\overline{X}_j)[i]$ is $\mathcal{N}(0,1)$-distributed for every j, hence by definition C_i is $\chi^2(n)$-distributed. □

A chi-square distribution with n degrees of freedom quickly tends towards a Gaussian distribution as n grows. A rule of thumb in probability theory is to consider the approximation $\chi^2(n) \approx \mathcal{N}(n,2n)$ quite reasonable for $n \geqslant 30$. From Lemma 4.2, $\lambda_{k,x}$ is a sum of a constant, a Gaussian random variable and T chi-square random variables with $N\tau_x$ degrees of freedom. Therefore, for $N\tau_x$ large enough, we can consider that $\lambda_{k,x}$ has a Gaussian distribution. If this holds for every $x\in\mathcal{X}$ then the distribution of λ_k can fairly be approximated by a Gaussian.

As shown in Section 4.5, approximating the distribution of the likelihood vector by a multivariate Gaussian is sound to estimate the success rate of a likelihood attack. The computational cost of (4.9) and

(4.10) is $O(|\mathcal{X}|T^3)$ where T denotes the leakage dimension. The total cost of computing the distribution parameters is hence $O(|\mathcal{K}|^2|\mathcal{X}|T^3)$. This may be prohibitive if the leakage dimension is high. However, the leakage dimension can be reduced by pre-processing the leakage measurements [28, 213]. In practice, $T = 3$ is often sufficient to catch most of the side channel information [28, 213].

4.4.4 Distribution in the uniform setting

Proposition 4.2 also applies to the log-likelihood vector $(\lambda_k)_{k\in\mathcal{K}}$. Besides, in the uniform setting (see Section 4.3.2), the success rate of a likelihood attack is also constant with respect to k^*.

4.5 Success rate evaluation

In accordance with the analyses of Sections 4.3 and 4.4, we assume that the distribution of the distinguishing vector $\mathbf{d} = (d_k)_{k\in\mathcal{K}}$ is a multivariate Gaussian $\mathcal{N}(m_{\mathbf{d}}, \Sigma_{\mathbf{d}})$. In this section we present two approaches to compute the success rate of a side channel key recovery attack, once the parameter of this distribution have been determined.

In the first approach, we show that the success rate can be expressed as a sum of Gaussian cumulative distribution functions (cdf). It can hence be estimated by numerically computing these cdf. The second approach consists in simulating the multivariate Gaussian vector $\mathbf{d}$ several times in order to get a precise estimation of the success rate.

4.5.1 Numerical computation

We show hereafter that the success rate can be expressed as a sum of Gaussian cdf. For this purpose, we need to introduce the *comparison vector* that is the $(|\mathcal{K}|-1)$-size vector $\mathbf{c} = (c_k)_{k\in\mathcal{K}/\{k^*\}}$ defined for every $k \in \mathcal{K}/\{k^*\}$ by:

$$c_k = d_{k^*} - d_k \ . \tag{4.26}$$

If all the coordinates of this vector are positive then the attack succeeds in isolating the good key guess as first candidate. If n coordinates are negative then the attack rates the good key guess as the $(n+1)^{\text{th}}$ candidate; in other words, it succeeds at the $(n+1)^{\text{th}}$ order. The comparison vector is a linear transformation of the distinguishing vector by a $((|\mathcal{K}|-1)\times|\mathcal{K}|)$-matrix P whose expression straightforwardly follows from (4.26). This implies that the comparison vector has a multivariate Gaussian distribution $\mathcal{N}(m_{\mathbf{c}}, \Sigma_{\mathbf{c}})$ where $m_{\mathbf{c}} = Pm_{\mathbf{d}}$ and $\Sigma_{\mathbf{c}} = P\Sigma_{\mathbf{d}}P'$.

Let $\alpha \subseteq \{1, \ldots, |\mathcal{K}|-1\}$ be a set of indices and let I_α and S_α be the $(|\mathcal{K}|-1)$-size vectors defined by:

$$I_\alpha[i] = \begin{cases} -\infty & \text{if } i \in \alpha \\ 0 & \text{if } i \notin \alpha \end{cases} \quad \text{and} \quad S_\alpha[i] = \begin{cases} 0 & \text{if } i \in \alpha \\ +\infty & \text{if } i \notin \alpha \end{cases} \ .$$

The vector $\mathbf{c}$ has exactly n negative coordinates if and only if there exists a set α of cardinal n s.t. $I_\alpha < \mathbf{c} < S_\alpha$. Since the intervals $([I_\alpha, S_\alpha])_\alpha$ are disjoints, the probability that exactly n coordinates of $\mathbf{c}$ be negative can be written as:

$$p_n = \sum_{\alpha;|\alpha|=n} \mathrm{P}\left[I_\alpha \leqslant \mathbf{c} \leqslant S_\alpha\right] \ . \tag{4.27}$$

The o^{th}-order success rate equals the sum $p_0 + p_1 + \cdots + p_{o-1}$ which from (4.27) gives:

$$\text{Succ-}o = \sum_{\alpha;|\alpha|<o} \mathrm{P}\left[I_\alpha \leqslant \mathbf{c} \leqslant S_\alpha\right] = \sum_{\alpha;|\alpha|<o} \Phi_{m_{\mathbf{c}},\Sigma_{\mathbf{c}}}\left(I_\alpha, S_\alpha\right) \ , \tag{4.28}$$

where $\Phi_{m,\Sigma}$ denotes the Gaussian cdf that satisfies $\Phi_{m,\Sigma} : (a,b) \mapsto \int_a^b \phi_{m,\Sigma}(x)\, dx$.

Relation (4.28) shows that the o^{th}-order success rate can be computed by performing $\sum_{i<o}\binom{|\mathcal{K}|-1}{i}$ multivariate Gaussian cdf calculations (on $(|\mathcal{K}|-1)$-size Gaussian vectors). The numerical computation of multivariate Gaussian cdf is a classical issue in statistics. Some solutions exist (see for instance [101, 102]) that can be used to precisely compute the success rate according to (4.28).

This approach has some drawbacks. Firstly, the numerical computations of Gaussian cdf may be difficult with covariance matrices having particular forms and/or quite high dimensions. For instance it requires that the covariance matrix is not singular, which is not always the case in our context. Yet another drawback of this approach is that the computation of high-order success rates requires an important number of Gaussian cdf computations. Regarding these issues, a possible alternative is presented in the next section.

4.5.2 Gaussian simulation

Another possibility to compute the success rate is to perform a Gaussian simulation. The principle is to simulate several times the distribution $\mathcal{N}(m_{\mathbf{d}}, \Sigma_{\mathbf{d}})$. This amounts to randomly pick up several distinguishing vectors each one corresponding to the result of an attack. The success rate is estimated based on these different results. In other words, this approach works as an attack simulation, but instead of performing the attack several times, we perform several Gaussian random vectors simulation which is clearly more efficient especially when the number of leakage measurements is high and/or the leakage dimension is high. Another advantage of this approach is that the success rate at the different orders as well as the guessing entropy (see Section 3.4.4.2) can all be computed using the same simulated distinguishing vectors. Finally Gaussian simulation is sound even when the covariance matrix is singular which may happen in our context.

4.6 Empirical validation

In order to empirically validate the theoretical analyses conducted in the previous sections we performed several simulations. We chose $Z = X \oplus K$ as target variable where X and K are 8-bit variables. The leakage means $(m_z)_{z\in\mathcal{Z}}$ and the leakage covariance matrix Σ were drawn with random coefficients. Their dimensions were set to 1 for a correlation attack, and to 3 for a likelihood attack (this is a typical dimension when subspace-based profiling is involved [28,213]). The attacker model/estimations were first assumed to be exact (*i.e.* $\mathsf{M}(z) = m_z$, $\widehat{m}_z = m_z$ and $\widehat{\Sigma} = \Sigma$) and then assumed to be slightly erroneous (by inserting random errors).

On the one hand, the success rate was estimated empirically by simulating the attack. Namely, the leakage measurements l_i corresponding to random inputs x_i were randomly picked up according to the leakage parameters $(m_{x_i,k^*}, \Sigma_{x_i,k^*})$. The attack was performed several times (a few thousands) on such simulated measurements in order to obtain an empirical success rate. On the other hand, the success rate was estimated using our approach. We computed the distinguishing vector expectation and covariance matrix (such as described in Sections 4.3 and 4.4) according to the leakage parameters and assuming $\tau_x = 1/256$ for every x. Then we performed Gaussian simulations (see Section 4.5.2) to get an estimation of the success rate.

As expected, for the correlation attacks, the different success rates obtained with our approach always perfectly match the success rates obtained by attack simulations. For likelihood attacks, the success rates obtained with our approach also match quite well the success rates obtained by attack simulations. The precision of this matching depends on the number of leakage measurements required for the attack to succeed (with a high success rate). When this number is quite low (*i.e.* around a few hundreds), our estimation slightly overvalues the real success rate. This overvaluation becomes less marked as the number of required leakage measurements for the attack to succeed (with high success rate) increases. As an illustration the success rate of four attacks requiring different amounts of leakage measurements is plotted Figure 4.1. The success rates that were obtained by attack simulation are plotted in black while the corresponding ones obtained with our approach are plotted in grey. The convergence can be clearly observed. Figure 4.2 shows both success rates for an attack requiring around 200 leakage measurements to succeed (with high success rate). When moving up to 500 required leakage measurements, the curves completely mix up.

The different empirical results that we obtained have demonstrated the soundness of our theoretical analysis. They also show that the approximation $\tau_x \approx 1/|\mathcal{X}|$ is sound when the x_i's are randomly drawn (*i.e.* in a known plaintext/ciphertext attack setting).

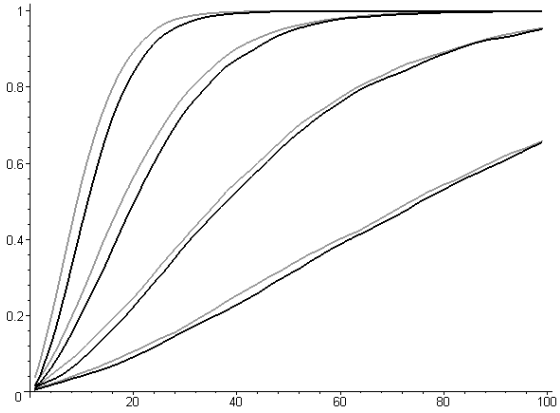


Figure 4.1: Success rates of different likelihood attacks over an increasing number of leakage measurements.

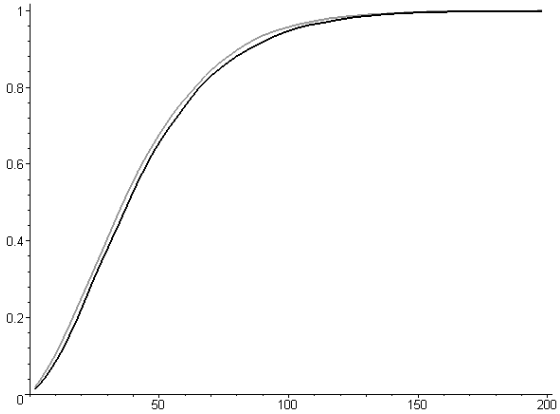


Figure 4.2: Success rates of a likelihood attack over an increasing number of leakage measurements.

4.7 Number of leakage measurements *vs.* leakage variance

We investigate hereafter how an increase in the leakage noise variance impacts the number of leakage measurements required for a given success rate.

4.7.1 Correlation distinguisher

From (4.3) we see that the distinguishing vector expectation does not depend on the leakage variance nor on the number of leakage measurements. Conversely, (4.4) shows that the covariance matrix depends on these parameters. If the leakage variance is multiplied by a factor λ then so does the covariance matrix. And if the number of measurements is multiplied by a factor λ then the covariance matrix is multiplied by $1/\lambda$. As a result, if the leakage variance is increased by a given factor, the number of leakage measurements must also be increased by the same factor to keep unchanged the distinguisher distribution and hence the attack success rate.

4.7.2 Likelihood distinguisher

In order to simplify our analysis, we make the independent noise assumption (see Section 3.4.3). Under this assumption, the covariance matrix Σ_z is the same for every $z \in \mathcal{Z}$.

Let us denote the leakage covariance matrix by Σ and its estimation by $\widehat{\Sigma}$. Under the independent noise assumption, (4.9) and (4.10) can be rewritten as:

$$\mathrm{E}\left[\lambda_k\right] = C_1 - \frac{1}{2}\sum_{x\in\mathcal{X}} \tau_x \left\| \widehat{\Sigma}^{-1/2}\left(m_{x,k^*} - \widehat{m}_{x,k}\right)\right\|^2 \ , \tag{4.29}$$

and

$$\mathrm{Cov}\left[\lambda_{k_1}, \lambda_{k_2}\right] = C_2 + \frac{1}{N}\sum_{x\in\mathcal{X}} \tau_x \ \left(m_{x,k^*} - \widehat{m}_{x,k_1}\right)' \widehat{\Sigma}^{-1}\,\Sigma\,\widehat{\Sigma}^{-1}\left(m_{x,k^*} - \widehat{m}_{x,k_2}\right) \ , \tag{4.30}$$

where $C_1 = \log\left(2\pi|\widehat{\Sigma}|\right) + \mathrm{Tr}\Big(\widehat{\Sigma}^{-1/2}\,\Sigma\left(\widehat{\Sigma}^{-1/2}\right)'\Big)$ and $C_2 = \frac{1}{2N}\left\|\widehat{\Sigma}^{-1/2}\,\Sigma\,(\widehat{\Sigma}^{-1/2})'\right\|_{hs}^2$ are constant with respect to k.

We show in Section 4.5.1 that the success rate depends of the distribution of the comparison vector $\mathbf{c} = (c_k)_{k\in\mathcal{K}/\{k^*\}}$ that is defined, for a likelihood attack , by $c_k = \lambda_{k^*} - \lambda_k$ for every $k \in \mathcal{K}$. Assuming $(\lambda_k)_{k\in\mathcal{K}}$ Gaussian, $\mathbf{c}$ has a Gaussian distribution whose parameters satisfies:

$$\mathrm{E}\left[c_k\right] = \mathrm{E}\left[\lambda_{k^*}\right] - \mathrm{E}\left[\lambda_k\right] \ , \tag{4.31}$$

and

$$\mathrm{Cov}\left[c_{k_1}, c_{k_2}\right] = \mathrm{Var}\left[\lambda_{k^*}\right] + \mathrm{Cov}\left[\lambda_{k_1}, \lambda_{k_2}\right] - \mathrm{Cov}\left[\lambda_{k^*}, \lambda_{k_1}\right] - \mathrm{Cov}\left[\lambda_{k^*}, \lambda_{k_2}\right] \ . \tag{4.32}$$

From these expressions, we can see that the constant terms C_1 and C_2 of (4.29) and (4.30) cancel each other out in the expectation and the covariance matrix of $\mathbf{c}$. It thus appears that multiplying the leakage covariance matrix by a factor λ (and assuming that its estimation is also multiplied by λ) results in the multiplication of $m_{\mathbf{c}}$ and $\Sigma_{\mathbf{c}}$ by $1/\lambda$ while multiplying the number of leakage measurements by λ results in the multiplication of $\Sigma_{\mathbf{c}}$ by $1/\lambda$.

One can verify that the Gaussian cdf satisfies for every (a, b):

$$\Phi_{m/\lambda, \Sigma/\lambda^2}(a, b) = \Phi_{m,\Sigma}(\lambda a, \lambda b) \ . \tag{4.33}$$

As shown in Section 4.5.1, the success rate can be expressed as a sum of cdf $\Phi_{m_{\mathbf{c}},\Sigma_{\mathbf{c}}}$ with inputs in $\{0, +\infty, -\infty\}^{|\mathcal{K}|-1}$. One thus deduces from (4.33) that multiplying $m_{\mathbf{c}}$ by $1/\lambda$ and $\Sigma_{\mathbf{c}}$ by $1/\lambda^2$ keeps the success rate unchanged. Hence we obtain that multiplying the leakage covariance matrix and multiplying the number of leakage measurements have complementary effects on the success rate of a likelihood attack.

Fact 4.1. *Under the Gaussian noise assumption and the independent noise assumption, a correlation attack and a likelihood attack are affected in the same way when the leakage noise increases. Besides, when the leakage noise increases, the ratio between the number of required leakage measurements for both attacks remains constant.*

Chapter 5

Analysis and improvement of mutual information analysis

Contents

5.1 Introduction

In this chapter, we study theoretical and practical aspects of MIA. We first conduct a theoretical analysis of MIA under the Gaussian noise assumption. Afterwards we address the major practical issue of MIA: the entropy estimation. In particular, we introduce a parametric estimation based MIA which is clearly more efficient than the original proposal. Eventually, we provide simulations and practical experiments that validate our analysis. Further analysis of MIA can be found in [170, 241].

Part of the results presented hereafter have been published in collaboration with Emmanuel Prouff in the international conference on *Applied Cryptography and Network Security (ACNS 2009)* [5].

5.2 Preliminaries

We consider an attacker that uses a model $\mathsf{M} = \hat{\varphi} \circ f$ and that estimates the mutual information $\mathrm{I}(\mathsf{M}(X,k);L)$ to distinguish key guesses $k \in \mathcal{K}$. In the following, we shall denote the function $f(\cdot,k)$ by f_k and we shall denote the variable $\mathsf{M}(X,k)$ by $Y(k)$ (*i.e.* we have $Y(k) = \hat{\varphi} \circ f_k(X)$). The set $\mathrm{Im}(\hat{\varphi})$ of the possible prediction values will be firther denoted y. For clarity reasons, we shall further use Y to denote $Y(k)$ when there is no ambiguity.

An MIA succeeds at the o^{th}-order iff the estimations $\hat{\mathrm{I}}(L, Y(k))$ of $\mathrm{I}(L, Y(k))$ satisfy:

$$k^* \in \underset{k \in \mathcal{K}}{\text{argmax-}o} \ \hat{\mathrm{I}}(L, Y(k)) \ . \tag{5.1}$$

We therefore deduce two necessary conditions for an MIA to succeed at the o^{th}-order:

— *Theoretical.* The mutual information $(\mathrm{I}(L, Y(k)))_{k \in \mathcal{K}}$ must satisfy:

$$k^* \in \underset{k \in \mathcal{K}}{\text{argmax-}o} \ \mathrm{I}(L, Y(k)) \ . \tag{5.2}$$

— *Practical.* The estimations of $(\mathrm{I}(L, Y(k)))_{k \in \mathcal{K}}$ must be good enough to satisfy (5.1) while (5.2) is satisfied.

In the next section, we study when Relation (5.2) is satisfied. This will allow us to characterize (with regards to f, φ, $\hat{\varphi}$) when an MIA is theoretically possible. Then, for 3-tuples $(f, \varphi, \hat{\varphi})$ s.t. (5.2) is satisfied, we shall study in Section 5.5 the success probability of the MIA according to the estimation method used to compute $\hat{\mathrm{I}}$ and according to the noise variation. This will allow us to characterize when an MIA is practically feasible (*i.e.* when (5.1) is satisfied) and when it is more efficient than the other SCA attacks.

5.3 Study of MIA in the Gaussian model

In this section we focus on first-order MIA and, in a second time, we extend our analysis to the higher-order case *i.e.* when the target implementation is protected by masking [65]. We conduct our analysis assuming a univariate leakage model and under the independent noise and the Gaussian noise assumptions (see Section 3.4.3). Namely, the leakage can be expressed as:

$$L = \varphi \circ f_{k^*}(X) + B \ , \tag{5.3}$$

where $B \sim \mathcal{N}\left(0, \sigma^2\right)$.

We shall further make the two following assumptions which are realistic in a side channel analysis context and make the formalization easier.

Assumption 5.1 (Uniformity)**.** *The plaintext X has a uniform distribution over $\mathbb{F}_2^n$.*

Assumption 5.2 (Balancedness)**.** *For every $k \in \mathcal{K}$, the function $f_k : x \in \mathbb{F}_2^n \mapsto f_k(x) \in \mathbb{F}_2^n$ is s.t. $\#\{x \in \mathbb{F}_2^n ; y = f_k(x)\}$ equals 2^{n-m} for every $y \in \mathbb{F}_2^m$.*

Remark 5.1. This assumption states that the algorithmic functions targeted by the SCA are balanced which is usually the case in a cryptographic context.

5.3.1 First-order MIA

The mutual information $\mathrm{I}(L, Y(k))$ equals the difference of entropies $\mathrm{H}[L] - \mathrm{H}[L|Y(k)]$. Since $\mathrm{H}[L]$ does not depend on the key prediction, $\mathrm{I}(L|Y(k))$ reaches one of its o highest values when k ranges over $\mathcal{K}$ iff the conditional entropy $\mathrm{H}[L|Y(k)]$ reaches one of its o smallest values. One deduces that an MIA is theoretically possible iff the 3-tuple $(f, \varphi, \hat{\varphi})$ is s.t.:

$$k^* \in \underset{k \in \mathcal{K}}{\text{argmin-}o} \ \mathrm{H}[L|Y(k)] \ , \tag{5.4}$$

where argmin-o is defined analogously to argmax-o .

The starting point of our analysis is that studying the MIA effectiveness is equivalent to investigating the minimality of $\mathrm{H}[L|Y(k)]$ over $\mathcal{K}$. According to (2.14), we have:

$$\mathrm{H}[L|Y] = \sum_{y \in y} \mathrm{P}\left[Y = y\right] \mathrm{H}[L|Y = y] \ . \tag{5.5}$$

Under the Gaussian noise assumption, L is a continuous random variable, therefore $\mathrm{H}[L|Y=y]$ is defined as:

$$\mathrm{H}[L|Y] = -\sum_{y\in y} \mathrm{P}\left[Y=y\right] \int_{\ell} g_{L|Y=y}(\ell) \log g_{L|Y=y}(\ell) d\ell \ , \tag{5.6}$$

where $g_{L|Y=y}$ denotes the pdf of the random variable $(L|Y=y)$.

To reveal the relationship between $\mathrm{H}[L|Y(k)]$ and the key-prediction k, the expression of the pdf $g_{L|Y=y}$ in (5.6) needs to be developed. Let us denote by $E_k(y)$ the set $[\hat{\varphi}\circ f_k]^{-1}(y)$. Since X has a uniform distribution over $\mathbb{F}_2^n$, for every $\ell \in \mathcal{L}$ and every $y \in \mathrm{Im}(\hat{\varphi}\circ f_k)$ we have:

$$g_{L|Y=y}(\ell) = \frac{1}{\#E_k(y)} \sum_{x\in E_k(y)} \phi_{\varphi\circ f_{k^*}(x),\sigma}(\ell) \ . \tag{5.7}$$

The next proposition directly follows.

Proposition 5.1. *For every pair $(k^*,k)\in\mathcal{K}^2$ and every $y\in y$ the pdf of the random variable $(L|Y(k)=y)$ is a Gaussian mixture g_θ whose parameter θ satisfies:*

$$\theta = \left((w_{y,t}, t, \sigma^2)\right)_{t\in\mathrm{Im}(\varphi)} \ ,$$

with

$$w_{y,t} = \mathrm{P}\left[\varphi\circ f_{k^*}(X) = t \mid \hat{\varphi}\circ f_k(X) = y\right] = \frac{\#\left([\varphi\circ f_{k^*}]^{-1}(t)\cap E_k(y)\right)}{\#E_k(y)} \ .$$

In Proposition 5.1, the key hypothesis k only plays a part in the definition of the weights $w_{y,t}$ of the Gaussian mixture. In other terms, $g_{L|Y(k)=y}$ is always composed of the same Gaussian pdfs and the key hypothesis k only impacts the way how the Gaussian pdfs are mixed. To go further in the study of the relationship between k and $\mathrm{H}[L|Y(k)=y]$, let us introduce the following diagram where y is an element of y, where F', F and T are image sets:

$$y \xrightarrow{\hat{\varphi}^{-1}} F' \xrightarrow{f_k^{-1}} E_k(y) \xrightarrow{f_{k^*}} F \xrightarrow{\varphi} T \ ,$$

Based on the diagram above, we can make the two following observations:

— If the set T is reduced to a singleton set $\{t_1\}$ (*i.e.* if $\hat{\varphi}\circ f_k$ is constant equal to t_1 on $E_k(y)$), then all the probabilities $w_{y,t}$ s.t. $t\neq t_1$ are zero and w_{y,t_1} equals 1. In this case, one deduces from Proposition 5.1 that the distribution of $(L|Y(k)=y)$ is Gaussian and, due to (2.17), its conditional entropy satisfies

$$\mathrm{H}[L|Y(k)=y] = \frac{1}{2}\log(2\pi e\sigma^2) \ .$$

— If the set T contains more than one element (*i.e.* $\varphi\circ f_{k^*}$ is not constant over $E_k(y)$), then there exists at least two probabilities w_{y,t_1} and w_{y,t_2} which are non-zero and the distribution of $(L|Y(k)=y)$ is a Gaussian mixture (not Gaussian). Due to (2.18), its entropy satisfies:

$$\mathrm{H}[L|Y(k)=y] \geqslant \frac{1}{2}\log(2\pi e\sigma^2) \ .$$

When φ is constant on F' (*e.g.* when $\hat{\varphi}=\varphi$ or $\hat{\varphi}=\mathrm{Id}$), the two observations above provide us with a discriminant property. If $k^*=k$, then we have $F=F'$ and thus T is a singleton and $\mathrm{H}[L|Y(k)=y]$ equals $\frac{1}{2}\log(2\pi e\sigma^2)$. Otherwise, if $k\neq k^*$, then $f_{k^*}\circ f_k$ is likely to behave as a random function[4]. In this case, F is most of the time different from F' and T is therefore likely to have more than one element[5]. This implies that $\#\varphi\circ f_{k^*}(E_k(y))$ is strictly greater than 1 and thus that $\mathrm{H}[L|Y(k)=y]$ is greater than or equal to $\frac{1}{2}\log(2\pi e\sigma^2)$. Eventually, we get the following proposition in which we exhibit a tight lower bound for the differential entropy $\mathrm{H}[L|Y(k)]$.

[4] This property, sometimes called *wrong-key assumption* [63], is often assumed to be true in a cryptographic context, due to the specific properties of the primitive f.

[5] As detailed later, this is only true if $\hat{\varphi}\circ f_k$ is non-injective.

Proposition 5.2. *For every $(k^*, k) \in \mathcal{K}^2$, the conditional entropy $\mathrm{H}[L|Y(k)]$ satisfies:*

$$\frac{1}{2}\log\left(2\pi e\sigma^2\right) \leqslant \mathrm{H}[(L|Y(k)] \ . \tag{5.8}$$

Moreover, if $\varphi \circ f_{k^*}$ is constant on $E_k(y)$ for every $y \in y$, then the lower bound is tight.

Proof. Relation (5.8) is a straightforward consequence of (5.5) and of Propositions 2.1 and 5.1. The tightness is a direct consequence of (2.17) and Proposition 5.1. □

Remark 5.2. Intuitively, the entropy $\mathrm{H}[L]$ is a measure of the diversity or randomness of L. It is therefore reasonable to think that the more components in the Gaussian mixture pdf of $(L|Y(k) = y)$, the greater its entropy. Relation (5.8) provides a first validation of this intuition. The entropy is minimal when the pdf is a Gaussian one (*i.e.* when the Gaussian mixture has only one component). In our experiments (partially reported in Section 5.5), we noticed that the entropy of a Gaussian mixture whose components have the same variance, increases with the number of components.

Corollary 5.1. *If $\hat{\varphi} \circ f_k$ is injective, then $\mathrm{H}[L|Y(k)]$ equals $\frac{1}{2}\log(2\pi e\sigma^2)$ for every $k \in \mathcal{K}$.*

Proof. If $\hat{\varphi} \circ f_k$ is injective, then $E_k(y)$ is a singleton and $\varphi \circ f_{k^*}$ is thus constant on $E_k(y)$. □

If the functions $\hat{\varphi} \circ f_k$'s are all injective, then Corollary 5.1 implies that the MIA cannot succeed at any order. Indeed, in this case the entropy $\mathrm{H}[L|Y(k)]$ stays unchanged when k ranges over $\mathcal{K}$ and thus, k^* does not satisfy (5.4). As a consequence, when the f_k's are injective (which is for instance the case when f_k consists of a key addition followed by the AES S-box), then the attacker has to choose $\hat{\varphi}$ to be non-injective (*e.g.* the Hamming weight function). It must be noticed that this is a necessary but not sufficient condition since the function $\hat{\varphi}$ must also be s.t. $\mathrm{I}(\hat{\varphi}, \varphi)$ is non-negligible (otherwise the MIA would clearly fail). In this case, the attacker must have a certain knowledge about the leakage function φ in order to define an appropriate function $\hat{\varphi}$ and hence, the MIA does no longer benefit from one of its main advantages. This drawback can be overcome by exclusively targeting intermediate variables s.t. the f_k's are not injective (in AES, the attacker can for instance target the bitwise addition between two S-box outputs during the MixColumns operation).

5.3.2 Generalization to the higher-order case

In this section, we extend the analysis of MIA to higher orders and we assume that the target implementation is protected by Boolean masking. The sensitive variable $f_{k^*}(X)$ is now masked with $d-1$ independent random variables $M_1, \ldots, M_{d-1}$ which are uniformly distributed over $\mathcal{Z}$.

The masked data $f_{k^*}(X) \oplus M_1 \oplus \cdots \oplus M_{d-1}$ and the different masks M_j's are processed at different times. The leakage about $f_{k^*}(X) \oplus M_1 \oplus \cdots \oplus M_{d-1}$ is denoted by L_0 and the leakages about the M_j's are denoted by $L_1, \ldots, L_{d-1}$. Under the Gaussian noise and the independent noise assumptions, the L_j's satisfy:

$$L_j = \begin{cases} \varphi[f_{k^*}(X) \oplus \bigoplus_{t=1}^{d-1} M_t] + B_0 & \text{if } j = 0, \\ \varphi_j(M_j) + B_j & \text{if } j \neq 0, \end{cases} \tag{5.9}$$

where the B_j's are independent Gaussian noises with mean 0 and standard deviations σ_j, and where φ, $\varphi_1, \ldots, \varphi_{d-1}$ are d device dependent functions that are *a priori* unknown to the attacker. The vector $(L_0, \ldots, L_{d-1})$ is denoted by $\mathbf{L}$. The vector of masks $(M_1, \ldots, M_{d-1})$ is denoted by $\mathbf{M}$. We denote by $\Phi_{k^*}(X, \mathbf{M})$ the vector $(\varphi(f_{k^*}(X) \oplus \bigoplus_{t=1}^{d-1} M_t), \varphi_1(M_1), \ldots, \varphi_{d-1}(M_{d-1}))$.

To simplify our analysis, we assume that the attacker knows the manipulation times exactly and is therefore able to get a sample for the random variable $\mathbf{L}$. Under this assumption and for the same reasons as in the univariate case, higher-order MIA essentially consists in looking for the key candidate k which minimizes an estimation of the conditional entropy $\mathrm{H}[\mathbf{L}|Y(k)]$. Similarly to (5.5), this entropy satisfies:

$$\mathrm{H}[\mathbf{L}|Y] = \sum_{y \in y} \mathrm{P}\left[Y = y\right] \mathrm{H}[\mathbf{L}|Y = y] \ .$$

Since Y equals $\hat{\varphi} \circ f_k(X)$, the probabilities $\mathrm{P}[Y = y]$ in this sum can be exactly computed by the attacker. Once this computation has been performed, estimating $\mathrm{H}[\mathbf{L}|Y]$ amounts to estimate the entropies $\mathrm{H}[\mathbf{L}|Y = y]$ for all the prediction values y. These entropies are estimated as for the first-order case (see (5.6)), but the pdfs $g_{\mathbf{L}|Y(k)=y}$ are multivariate. More precisely, after denoting by Σ the matrix $(\mathrm{Cov}[B_i, B_j])_{i,j}$, we get:

$$g_{\mathbf{L}|Y(k)=y}(\ell) = \frac{1}{\#E_k(y)(\#\mathcal{Z})^{d-1}} \sum_{\substack{x \in E_k(y) \\ \mathbf{m} \in \mathcal{Z}^{d-1}}} g_{\Phi_{k^*}(x,\mathbf{m}),\Sigma}(\ell) \ . \tag{5.10}$$

In a similar way than in Section 5.3.1, the next proposition directly follows.

Proposition 5.3. *For every pair $(k^*, k) \in \mathcal{K}^2$ and every $y \in \mathrm{y}$ the pdf of the random variable $(\mathbf{L}|Y(k) = y)$ is a Gaussian mixture g_θ whose parameter θ satisfies:*

$$\theta = ((w_{y,\mathbf{t}}, \mathbf{t}, \Sigma))_{\mathbf{t}} \ ,$$

with

$$\Sigma = (\mathrm{Cov}[B_i, B_j])_{i,j} \ ,$$

and

$$w_{y,\mathbf{t}} = \mathrm{P}[\Phi_{k^*}(X, \mathbf{M}) = \mathbf{t} \mid \hat{\varphi} \circ f_k(X) = y] = \frac{\#\Phi_{k^*}^{-1}(\mathbf{t}) \cap (E_k(y) \times \mathcal{Z}^{d-1})}{\#(E_k(y) \times \mathcal{Z}^{d-1})} \ .$$

We deduce from Propositions 2.1 and 5.3 the following result.

Proposition 5.4. *For every $(k^*, k) \in \mathcal{K}^2$, the entropy $\mathrm{H}[\mathbf{L}|(Y(k), \mathbf{M})]$ satisfies:*

$$\frac{1}{2} \log\left((2\pi e)^d |\Sigma|\right) \leqslant \mathrm{H}[\mathbf{L}|(Y(k), \mathbf{M})] \ . \tag{5.11}$$

Moreover, if $\varphi \circ f_{k^*}$ is constant on $E_k(y)$ for every $y \in \mathrm{y}$, then the bound is tight.

We cannot deduce from the proposition above a wrong-key discriminator as we did in the univariate case. Indeed, to compute the entropy in (5.11) the attacker must know the mask values, which is impossible in our context. However, if the 3-tuple $(f, \varphi, \hat{\varphi})$ satisfies the condition of Proposition 5.4, then it can be checked that for every y the number of components in the multivariate Gaussian mixture pdf of $(\mathbf{L}|Y(k) = y)$ reaches its minimum for $k = k^*$. As discussed in Remark 5.2, this implies that the entropies of the random variables $(\mathbf{L}|Y(k) = y)$ are likely to be minimum for $k = k^*$. The simulations and experiments presented in Section 5.5 provides us with an experimental validation of this fact.

In the next sections, we assume that an MIA is theoretically possible. Namely, we assume that k^* belongs to argmin-o_k $\mathrm{H}[\mathbf{L}|Y(k)]$ for a given order o. At first, we study the success probability of an MIA according to the method used to estimate $\mathrm{H}[\mathbf{L}|Y(k)]$ and the noise variation. Secondly, we compare the efficiency of an MIA with the one of the DPA in different contexts.

5.4 Conditional entropy estimation

Let $\mathbf{L}$ be a d-dimensional random variable defined over $\mathcal{L}^d$ and let k be a key-candidate. We assume that the attacker has a sample of N leakage-message pairs $(\mathbf{l}_i, x_i) \in \mathcal{L}^d \times \mathcal{X}$ corresponding to a key k^*, and that he wants to estimate $\mathrm{H}[\mathbf{L}|Y(k)]$ to discriminate key-candidates k. Due to (5.5), estimating $\mathrm{H}[\mathbf{L}|Y(k)]$ from the sample $((\mathbf{l}_i, x_i))_i$ essentially amounts to estimate the entropy $\mathrm{H}[\mathbf{L}|Y(k) = y]$ for every $y \in \mathrm{y}$. For such a purpose, a first step is to compute estimations $\hat{g}_{\mathbf{L}|Y=y}$ of the pdfs $g_{\mathbf{L}|Y=y}$. Then, depending on the estimation method that has been applied, the $\mathrm{H}[\mathbf{L}|Y(k) = y]$'s are either directly computable or must still be estimated. In the following we present three estimation methods and we discuss their pertinency in our context.

5.4.1 Histogram method

Description. We choose *d bin widths* $h_0, \ldots, h_{d-1}$ (one for each coordinate of the leakage vectors) and we partition the leakage space $\mathcal{L}^d$ into regions $(\mathcal{R}_\alpha)_\alpha$ with equal volume $\upsilon = \prod_j h_j$. Let k be a key-candidate and let y be an element of y. We denote by $\mathcal{S}_y$ the sub-sample $\left(\mathbf{l}_i; x_i \in [\varphi \circ f_k]^{-1}(y)\right)_i \subseteq (\mathbf{l}_i)_i$ and by $\ell_{i,j}$ the jth coordinate of $\mathbf{l}_i$. To estimate the pdf $g_{\mathbf{L}|Y=y}$, we first compute the density vector D_y whose coordinates are defined by:

$$D_y(\alpha) = \frac{\#(\mathcal{S}_y \cap \mathcal{R}_\alpha)}{\#\mathcal{S}_y} \ , \tag{5.12}$$

where $\mathcal{S}_y \cap \mathcal{R}_\alpha$ denotes the sample of all the $\mathbf{l}_i$'s in $\mathcal{S}_y$ that belong to $\mathcal{R}_\alpha$.

The estimation $\hat{g}_{L|Y=y}$ is then defined for every $\mathbf{l} \in \mathcal{L}^d$ by $\hat{g}_{\mathbf{L}|Y=y}(\mathbf{l}) = \frac{D_y(i_\mathbf{l})}{\upsilon}$, where $i_\mathbf{l}$ is the index of the region $\mathcal{R}_{i_\mathbf{l}}$ that contains $\mathbf{l}$. Integrating the pdf estimation according to formula (2.12) gives the following estimation for the conditional entropy: $\hat{\mathrm{H}}(\mathbf{L}|Y = y) = -\sum_\alpha D_y(\alpha) \log(D_y(\alpha)/\upsilon)$. We eventually get:

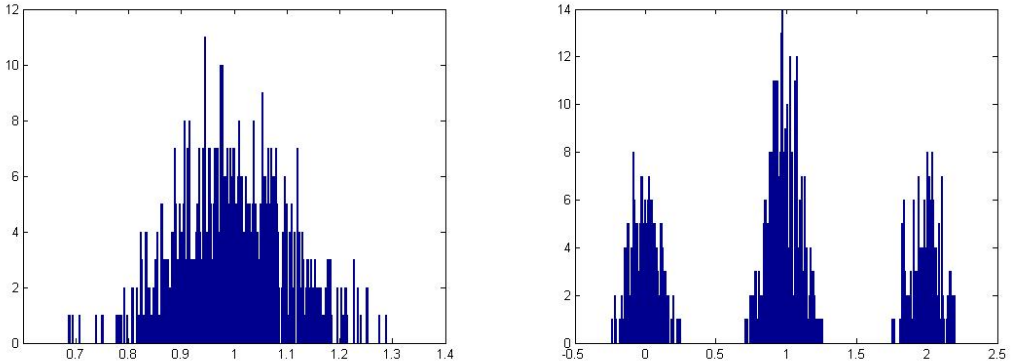
$$\hat{\mathrm{H}}(\mathbf{L}|Y) = -\sum_{y \in y} \mathrm{P}\left[Y = y\right] \sum_\alpha D_y(\alpha) \log\left((\frac{D_y(\alpha)}{\upsilon}\right) \ . \tag{5.13}$$

The optimal choice of the bin widths h_j is an issue of statistics. Actually, there are several rules that aim at providing *ad hoc* formulae for computing the h_j's based on the nature of the samples (see for instance [238, 244]). In our simulations, we chose to follow the Scott Rule, which is a common choice. Namely, if $\hat{\sigma}_j$ denotes the estimated standard deviation of the sample $(\ell_{i,j})_i$ of size N_j, then h_j satisfies:

$$h_j = 3.49 \times \hat{\sigma}_j \times N_j^{-\frac{1}{3}} \ .$$

Notice that in our context all the N_j's are equal to N.

Simulations. In order to illustrate the histogram method in the context of an MIA attack, we generated 10000 leakage measurements in the Gaussian model (5.3) for φ being the Hamming weight function, for f being the first DES S-box parameterized with the key $k^* = 11$ and for $\sigma = 0.1$. Since the DES S-box is non-injective, we chose the identity function for $\hat{\varphi}$. Figure 5.1 plots the estimations of the pdf $g_{L|Y=1}$ when $k = 11$ and when $k = 5$ (for a number of bins equal to 285). As expected (Proposition 5.1 and



(a) For $k = 11$ (correct guess).

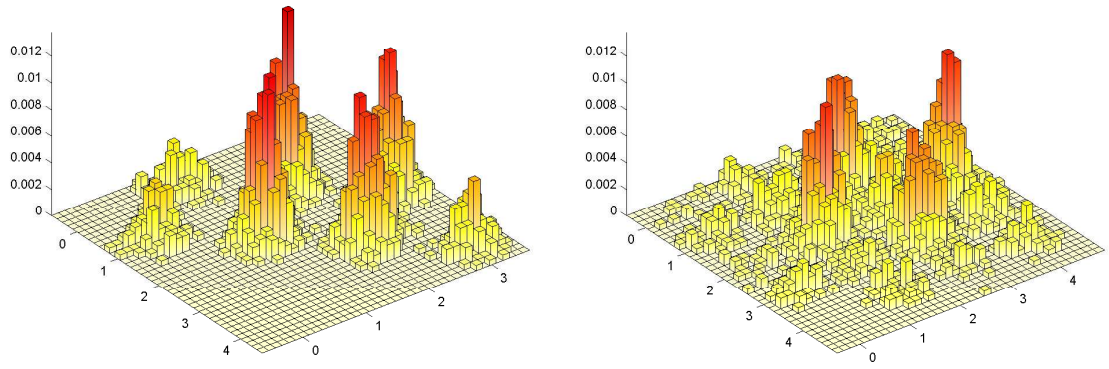
(b) For $k = 5$ (wrong guess).

Figure 5.1: Histogram method in the first-order case.

Corollary 5.1), a Gaussian pdf seems to be estimated when $k = 11$ (good key prediction), whereas a mixture of three Gaussian distributions seems to be estimated when $k = 5$ (wrong key prediction). For the experimentation described in the left-hand figure we obtained $\hat{\mathrm{H}}(L(11)|Y(11) = 1) = -1.31$ (due to

(2.17) we have $\mathrm{H}[L(11)|Y(11) = 1] = -1.27$) and we got $\hat{\mathrm{H}}(L(11)|Y(5) = 1) = -0.0345$ for the one in the right-hand side. Moreover, we validated that the estimated conditional entropy is minimum for the good key hypothesis.

In order to illustrate the histogram method in the context of a second-order MIA attack, we generated 10000 pairs of leakage measurements in the higher-order Gaussian model (5.9) with $d = 2$, with φ and φ_1 being the Hamming weight function, with f being the first DES S-box parametric with the key $k^* = 11$ and with $\sigma_0 = \sigma_1 = 0.1$. Figure 5.2 plots the estimations of the pdf $g_{\mathbf{L}|Y=1}$ when $k = 11$ and when $k = 5$. As expected, the mixture of Gaussian distributions for $k = 11$ have less components than for



(a) For $k = 11$ (correct guess). (b) For $k = 5$ (wrong guess).

Figure 5.2: Histogram method in the second-order case.

$k = 5$. For the experimentation in the left-hand figure we obtained $\hat{\mathrm{H}}(\mathbf{L}(11)|Y(11) = 1) = 0.22$ (and $\hat{\mathrm{H}}(\mathbf{L}(11)|Y(11)) = 0.14$), whereas we got 1.12 for $\hat{\mathrm{H}}(\mathbf{L}(11)|Y(5) = 1)$ (and 1.15 for $\hat{\mathrm{H}}(\mathbf{L}(11)|Y(5))$). Here again, the estimated conditional entropy was minimum for the good key hypothesis.

5.4.2 Kernel density method

Description. Although the histogram method can be made to be asymptotically consistent, other methods can be used that converge at faster rates. For instance, rather than grouping observations together in bins, the so-called *kernel density estimator* (or *Parzen window* method) can be thought to place small "bumps" at each observation, determined by the kernel function (see for instance [210]). The estimator consists of a "sum of bumps" and is clearly smoother as a result than the histogram method.

The kernel density estimation $\hat{g}_{\mathbf{L}|Y=y}$ based on the sample $\mathcal{S}_y$ is defined for every $\mathbf{l} = (\ell_0, \ldots, \ell_{d-1}) \in \mathcal{L}^d$ by:

$$\hat{g}_{L|Y=y}(\mathbf{l}) = \frac{1}{\#\mathcal{S}_y} \sum_{\mathbf{l}_i = (\ell_{i,0}, \ldots, \ell_{i,d-1}) \in \mathcal{S}_y} \frac{1}{\upsilon} \times \prod_{j=0}^{d-1} \mathbf{K}\left(\frac{\ell_j - \ell_{i,j}}{h_j}\right) \ ,$$

where $\mathbf{K}$ is a *kernel function* chosen among the classical ones (see for instance [245]), where the h_i's are *kernel bandwidths* and where υ equals $\prod_j h_j$. As recalled in [33], the following Parzen-windows entropy estimation of $\mathrm{H}[\mathbf{L}|Y = y]$ is sound when the sample size is large enough:

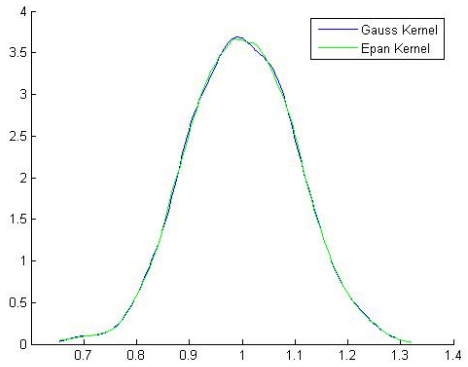
$$\hat{\mathrm{H}}(\mathbf{L}|Y = y) = -\frac{1}{\#\mathcal{S}_y} \sum_{\mathbf{l}_i \in \mathcal{S}_y} \log\left(\frac{1}{\#\mathcal{S}_y} \sum_{\mathbf{l}_r \in \mathcal{S}_y} \frac{1}{\upsilon} \times \prod_{j=0}^{d-1} \mathbf{K}\left(\frac{\ell_{i,j} - \ell_{r,j}}{h_j}\right)\right) \ ,$$

In our attack simulations, we chose the kernel function to be the Epanechnikov one defined for every u by $\mathbf{K}(u) = \frac{3}{4}(1 - u^2)$ if $|u| \leqslant 1$ and by $\mathbf{K}(u) = 0$ otherwise (another common choice is the Gaussian

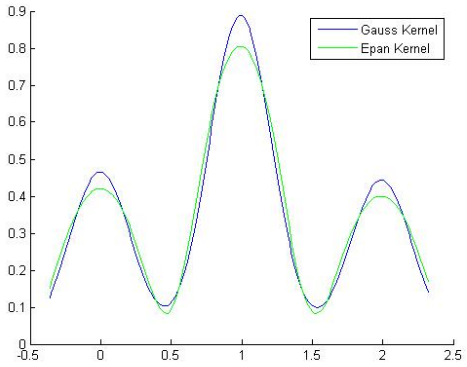
kernel [245]). Our choice was motivated not only by the fact that this kernel function has a simple form, but also by the fact that its efficiency is asymptotically optimal among all the kernels [120]. Let $\hat{\sigma}_j$ denotes the estimated standard deviation of the sample $(\ell_{i,j})_i$ of size N_j. To select the kernel bandwidth h_j, we followed the *normal scale rule* [210]. Namely, we chose the h_j's s.t.

$$h_j = 1.06 \times \hat{\sigma}_j \times N_j^{-\frac{1}{5}} \ . \tag{5.14}$$

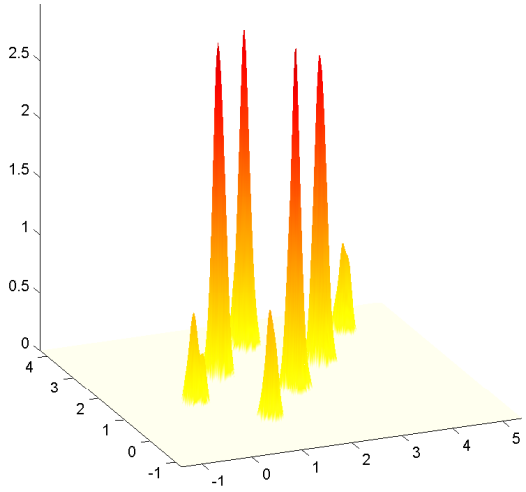
Simulations. In order to illustrate the effectiveness of the kernel method, we applied it for the same simulated traces used for our first and second-order histogram experiments (Figure 5.1 and Figure 5.2). We present our results in Figure 5.3(a–b) for the first-order and in Figure 5.3(c–d) for the second-order.



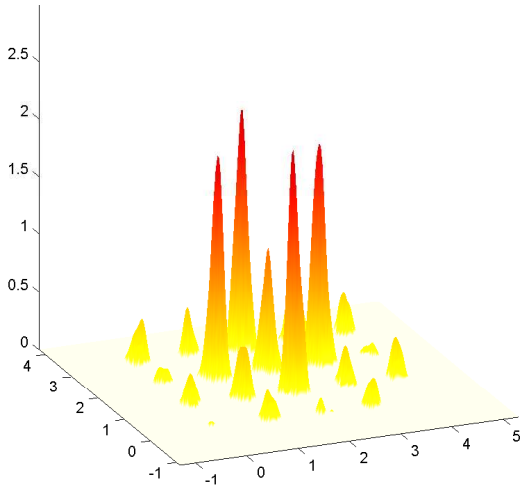
(a) First-order for $k = 11$ (correct key guess).



(b) First-order for $k = 5$ (wrong key guess).



(c) Second-order for $k = 11$ (correct key guess).



(d) Second-order for $k = 5$ (wrong key guess).

Figure 5.3: Kernel method

As expected, the pdf estimated in Figure 5.3(a) when $k = 11$ seems to be a Gaussian one, whereas the pdf estimated when $k = 5$ seem to be a mixture of three Gaussian distributions. Moreover, the estimations are smoother than in the case of the histogram method and there is no noticeable differences between the estimation with Gaussian kernel and the estimation with the Epanechnikov one. For the

experimentation described in the left-hand figure we obtained $\hat{\mathrm{H}}(L(11)|Y(11)=1) = -0.88$ and we got 0.54 for $\hat{\mathrm{H}}(L(11)|Y(5)=1)$ (right-hand side).

As expected, in Figure 5.3(c) the mixture of Gaussian distributions for $k=11$ have less components than for $k=5$. For the experimentation in the left-hand figure we obtained $\hat{\mathrm{H}}(\mathbf{L}(11)|Y(11)) = 0.17$, whereas we got 0.52 for $\hat{\mathrm{H}}(\mathbf{L}(11)|Y(5))$. Moreover, we validated that the conditional entropy $\hat{\mathrm{H}}(\mathbf{L}(11)|Y(k))$ is minimum for $k=k^*=11$.

5.4.3 Parametric estimation

Description. As argued in Section 5.3, under the Gaussian noise assumption, the conditional leakage pdfs are Gaussian mixtures. One can hence directly estimate the parameters θ of these Gaussian mixtures to obtain sound estimations of the pdfs.

First-order case. Relation (5.7) shows that $g_{L|Y=y}$ is a Gaussian mixture g_θ whose parameter θ satisfies:

$$\theta = \left(\frac{1}{\#E_k(y)}, \varphi \circ f_{k^*}(x), \sigma^2 \right)_{x \in E_k(y)} . \tag{5.15}$$

Based on this relation, an alternative to the methods presented above is to compute an estimation $\hat{\theta}$ of the parameter θ so that we get $\hat{g}_{L|Y=y} = g_{\hat{\theta}}$ and thus:

$$\hat{\mathrm{H}}(\mathbf{L}|Y=y) = -\int_{\mathbf{l}\in\mathcal{L}^d} g_{\hat{\theta}}(\mathbf{l}) \log_2 g_{\hat{\theta}}(\mathbf{l}) d\mathbf{l} \ .$$

For every x, the mean value $\varphi \circ f(x, k^*)$ in (5.15) can be estimated by:

$$\bar{\mathbf{l}}_x = \frac{1}{\#\{i; x_i = x\}} \sum_{i; x_i = x} \mathbf{l}_i$$

and the noise variance σ^2 by:

$$\hat{\sigma}^2 = \sum_i \left(\mathbf{l}_i - \bar{\mathbf{l}}_{x_i} \right)^2 .$$

On the whole, this provides us with the following estimation $\hat{\theta}$ of θ:

$$\hat{\theta} = \left(\frac{1}{\#E_k(y)}, \bar{\mathbf{l}}_x, \hat{\sigma}^2 \right)_{x \in E_k(y)} .$$

Higher-order case. Relation (5.10) shows that $g_{\mathbf{L}|Y=y}$ is a Gaussian mixture g_θ whose parameter θ satisfies:

$$g_\theta = \frac{1}{\#E_k(y)} \sum_{x \in E_k(y)} g_{\theta_x} \ , \tag{5.16}$$

where g_{θ_x} denotes the Gaussian mixture pdf of the random variable $(\mathbf{L}|X=x)$ whose parameter satisfies:

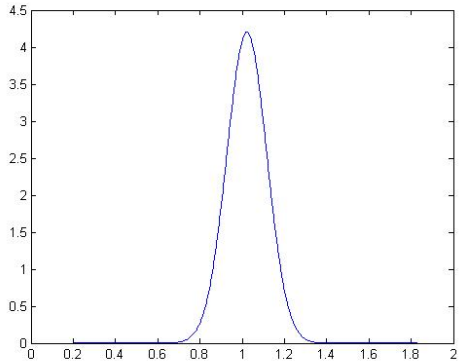
$$\theta_x = \left(\frac{1}{(\#\mathcal{Z})^{d-1}}, \Phi_k(x, \mathbf{m}), \Sigma \right)_{\mathbf{m} \in \mathcal{Z}^{d-1}} .$$

The mean values $\Phi_k(x, \mathbf{m})$ of the different components cannot be directly estimated as in the first-order case since the values taken by the masks $\mathbf{m}$ for the different leakage observations $\mathbf{l}_i$ are not assumed to be known. To deal with this issue, a solution is to involve Gaussian mixture estimation methods such as the *expectation maximization algorithm*. By applying it on the sample $(\mathbf{l}_i \ ; \ x_i = x)_i$ we get an estimation $\hat{\theta}_x$ of θ_x for every $x \in \mathcal{X}$. Then, according to (5.16), we obtain:

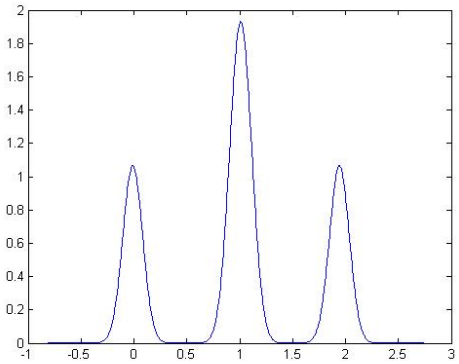
$$\hat{\mathrm{H}}(\mathbf{L}|Y=y) = -\sum_x \int_{\mathbf{l}\in\mathcal{L}^d} g_{\hat{\theta}_x}(\mathbf{l}) \log g_{\hat{\theta}_x}(\mathbf{l}) d\mathbf{l} \ .$$

Remark 5.3. As an advantage of the parametric estimation method, the mean values $\mathbf{l}_x$'s (resp. the estimated parameters $\hat{\theta}_x$'s) are only computed once for every x and are then used to compute $\hat{\mathrm{H}}(\mathbf{L}|Y(k)=y)$ for every pair (k, y).

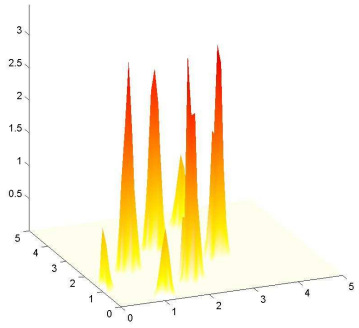
Simulations. As for the previous estimation methods, we applied the parametric estimation to the same simulated traces. The resulting estimated pdfs $(\hat{g}_{\mathbf{L}(11)|Y(k)=1})_{k=11,5}$ are plotted in Figure 5.4(a–b) for the first-order and in Figure 5.4(c–d) for the second-order.



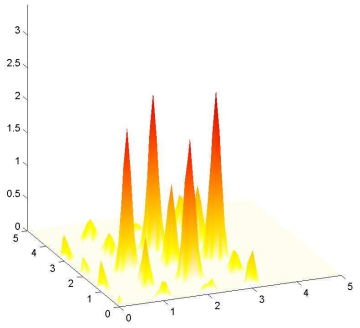
(a) First-order for $k = 11$ (correct key guess).



(b) First-order for $k = 5$ (wrong key guess).



(c) Second-order for $k = 11$ (correct key guess).



(d) Second-order for $k = 5$ (wrong key guess).

Figure 5.4: Parametric Estimation

The results are similar to those of the previous estimation methods. For the first-order case, we distinguish a mixture of three Gaussian distributions for the wrong key hypothesis while a single Gaussian pdf is observed for the correct one. For the second-order case, the Gaussian mixture obtained for the wrong key hypothesis contains more components than the one for the correct key hypothesis. Once again, the estimated entropy is lower for the correct key hypothesis than for the wrong one. For instance, the entropies of the plotted pdfs equal -0.94 (correct hyp.) and 0.13 (wrong hyp.) for the first-order case and 0.24 (correct hyp.) and 0.60 (wrong hyp.) for the second-order case.

5.5 Experimental results

5.5.1 First-Order Attack Simulations

To compare the efficiency of MIA with respect to the estimation method, we simulated leakage measurements in the Gaussian model (5.3) with φ being the Hamming weight function and f being the first DES S-box (we therefore have $n = 6$ and $m = 4$). For various noise standard deviations σ and for the estimation methods described in previous sections, we estimated the number of messages required to have an attack first-order success rate greater than or equal to 90% (this success rate being computed for 1000

attacks). Moreover, we included first-order DPA in our tests to determine whether and when an MIA is more efficient than a DPA[6]. Each attack was performed with $\hat{\varphi}$ being the identity function in order to test the context in which the attacker has no knowledge about the leakage model. Moreover, each attack was also performed with $\hat{\varphi}$ being the Hamming weight function in order to test the context where the attacker has a good knowledge of the leakage model. The results are given in Table 5.1 where MIA_H, MIA_K and MIA_P respectively stand for the histogram, the kernel and the parametric MIA.

Table 5.1: Attack on the first DES S-box – Number of measurements required to achieve a success rate of 90% according to the noise standard deviation σ.

Attack \ σ	0.5	1	2	5	10	15	20	50	100
DPA, $\hat{\varphi} = \text{Id}$	30	30	100	1000	3000	7000	15000	70000	260000
MIA_H (hist.), $\hat{\varphi} = \text{Id}$	80	160	600	4000	20000	50000	95000	850000	10^6+
MIA_K (kernel), $\hat{\varphi} = \text{Id}$	70	140	500	3000	15000	35000	60000	500000	10^6+
MIA_P (param.), $\hat{\varphi} = \text{Id}$	60	100	300	2000	5000	15000	20000	150000	500000
DPA, $\hat{\varphi} = \text{HW}$	30	30	70	400	2000	4000	7000	45000	170000
MIA_H (hist.), $\hat{\varphi} = \text{HW}$	40	70	300	1500	7000	20000	40000	320000	10^6+
MIA_K (kernel), $\hat{\varphi} = \text{HW}$	30	60	190	1500	5500	15000	25000	190000	900000
MIA_P (param.), $\hat{\varphi} = \text{HW}$	70	70	150	1000	3000	7000	15000	65000	300000

It can be checked in Table 5.1 that DPA is always better than MIA when $\hat{\varphi} = \text{HW}$. This is not an astonishing result in our model, since the deterministic part of the leakage corresponds to the Hamming weight of the target variable. More surprisingly, this stays true when $\hat{\varphi}$ is chosen to be the identity function. This can be explained by the strong linear dependency between the identity function and the Hamming weight function over $\mathbb{F}_2^4 = \{0, \ldots, 15\}$. Eventually, both results suggest that DPA is more suitable than MIA for attacking a device leaking first-order information in a model close to the Hamming weight model with Gaussian noise. When looking at the different MIAs, we can notice that MIA_P becomes much more efficient than MIA_H and MIA_K when the noise standard deviation increases.

5.5.2 Second-Order Attack Simulations

In a DPA, the attacker computes Pearson's correlation coefficients which is a function of two univariate samples. Thus, when DPA is applied against dth-order masking (see (5.9)) a multivariate function must be defined to combine the different leakage signals (see Chapter 6). This signal processing induces an information loss which strongly impacts the higher-order DPA efficiency when the noise increases. Because an higher-order MIA can operate on multivariate samples, it does not suffer from the aforementioned drawback. We could therefore expect MIA to become more efficient than DPA when it is performed against masking. To compare higher-order DPA and higher-order MIA, we simulated power consumption measurements such as in (5.9) with $d = 2$, with $\varphi = \varphi_1 = \text{HW}$, with $\sigma_1 = \sigma_2 = \sigma$ and with f being the first DES S-box. For various noise standard deviations σ and for the estimation methods described in previous sections, we estimated the number of measurements required to have an attack success rate greater than or equal to 90% (this success rate being computed over 100 attacks). Table 5.2 reports the results that we obtained[7] for second-order DPA (2O-DPA) and for second-order MIA with histogram estimation method (2O-MIA_H) and with kernel estimation method (2O-MIA_K). We performed second-order DPA with Hamming weight prediction function and for two different combining functions: the absolute difference and the normalized product (see Chapter 6). For MIA, we tried both Hamming weight and identity prediction functions. Moreover, for MIA with histogram estimation, we tried two rules for the choice of the bin-width: Scott's rule (see Section 5.4.1) and the rule proposed in [105].

Remark 5.4. We experimented that second-order MIA with parametric estimations using the expectation maximization algorithm is inefficient. In fact, estimating a Gaussian mixture using the EM algorithm

[6] Attacks have been performed for measurements numbers ranging over 50 different values from 30 to 10^6.

[7] The results given in the paper [5] for second-order MIA simulations are erroneous. Table 5.2 provides the corrected results.

requires a great number of samples, especially when the number of components in the mixture is not small. In our context, the number of components equals the number of possible mask values[8], that is 16 when attacking a DES S-box. To lower the number of components, one could focus on a restricted number of bits (considering the remaining ones as an algorithmic noise). Such an approach has been followed by Lemke-Rust and Paar in [152] in the context of higher-order profiled attacks. Another approach could be to look for other estimation methods dedicated to Gaussian mixtures and, possibly, to adapt them for masked implementations. We let such investigations for future research.

Table 5.2: Second-order attack on DES S-box – Number of measurements required to achieve a success rate of 90% according to the noise standard deviation σ.

Attack $\setminus$ σ	0.5	1	2	5	7	10
2O-DPA ($\hat{\varphi}$ = HW, abs. difference)	300	800	5000	200000	10^6+	10^6+
2O-DPA ($\hat{\varphi}$ = HW, norm. product)	300	400	3000	70000	300000	10^6+
2O-MIA$_{\mathrm{H}}$ ($\hat{\varphi}$ = Id, Scott's Rule)	1200	7000	75000	10^6+	10^6+	10^6+
2O-MIA$_{\mathrm{H}}$ ($\hat{\varphi}$ = Id, Rule in [105])	1800	7000	40000	1000000	10^6+	10^6+
2O-MIA$_{\mathrm{K}}$ ($\hat{\varphi}$ = Id)	600	2500	25000	600000	10^6+	10^6+
2O-MIA$_{\mathrm{H}}$ ($\hat{\varphi}$ = HW, Scott's Rule)	600	2700	34000	10^6+	10^6+	10^6+
2O-MIA$_{\mathrm{H}}$ ($\hat{\varphi}$ = HW, Rule in [105])	350	1300	9000	350000	10^6+	10^6+
2O-MIA$_{\mathrm{K}}$ ($\hat{\varphi}$ = HW)	300	1300	9000	n.a.	n.a.	n.a.

Table 5.2 shows that, contrary to what we could have expected, second-order DPA is always better than second-order MIA. As for the first-order case, we deduce that DPA is more suitable to attack masked implementations that leak the Hamming weight of the processed data with Gaussian noise. However, we also note that the efficiency of MIA is strongly impacted by the estimation methods and the related parameters (*e.g.* the choice of the bin-width for histograms). Determining the estimation method/parameters that optimize (or at least improve) the attack efficiency is hence a relevant open issue. Results reported in Table 5.2 also show that in the considered context, kernels perform better than histograms and that a Hamming weight prediction is better than an identity prediction. These observations are quite natural since, on the one hand, kernels are known to give tighter pdf estimations than histograms and, on the other hand, a Hamming weight prediction enables better discrimination of the wrong key guesses than an identity prediction in the presence of a Hamming weight leakage function[9]. Another observation is that, the bin-width selection rule proposed in [105] for histogram-based MIA leads to a more efficient attack than Scott's rule. More generally, we experimented that increasing the bin-width improve the attack efficiency until reaching a small number of bins. The analysis of the underlying reasons for this phenomenon and the study of the bin-width choice optimizing the MIA efficiency are open issues that deserve more investigations.

5.5.3 Practical Attacks

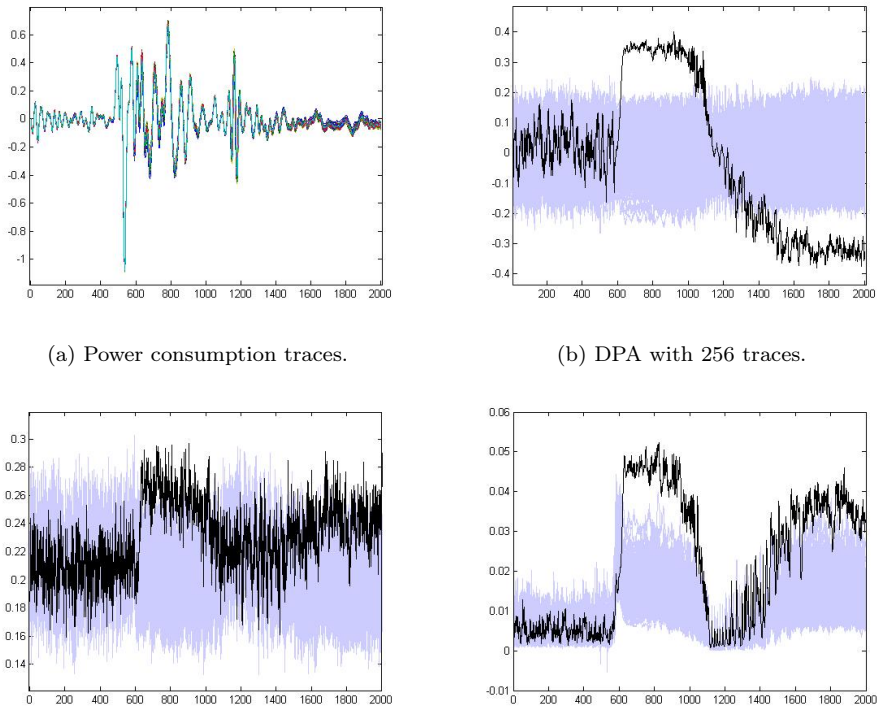
To experimentally validate our theoretical analysis and the simulations reported in Sections 5.4, 5.5.1 and 5.5.2, we experimented MIA with real-life leakage traces measured for different kinds of implementations. We first performed univariate MIA attacks against hardware and software implementations of the AES S-box. Then, we applied second-order MIA attacks against a masked software implementation of the first DES S-box. In both contexts, we also performed a DPA attack to compare its efficiency with that of MIA.

[8] It may be less in a particular leakage model (*e.g.* Hamming weight model) but the attacker does not *a priori* have such an information.

[9] More precisely, it can be checked that if $\varphi = \varphi_1 = \mathrm{HW}$ holds, then we have $\mathrm{I}(L; \mathrm{HW} \circ f_k(X)) \leq \mathrm{I}(L; f_k(X))$ for every k, with equality for $k = k^*$.

5.5.3.1 First Order Attacks.

We performed the attacks against two AES S-box implementations that use a lookup-table (*i.e.* f_k corresponds to the AES S-box). The first one is a hardware implementation on the chip SecMat V3/2 (see [122] for details about the chip and the circuit layout). The corresponding power consumption measurements are plotted in Fig. 5.5(a) over the time. It can be noticed that they are not very noisy. The second one is a software implementation running on a smart card with 8-bit architecture. As it can be seen in Fig. 5.6(a), the signal is much more noisy in this case.



(a) Power consumption traces.

(b) DPA with 256 traces.

(c) Histogram-based MIA with 1024 traces.

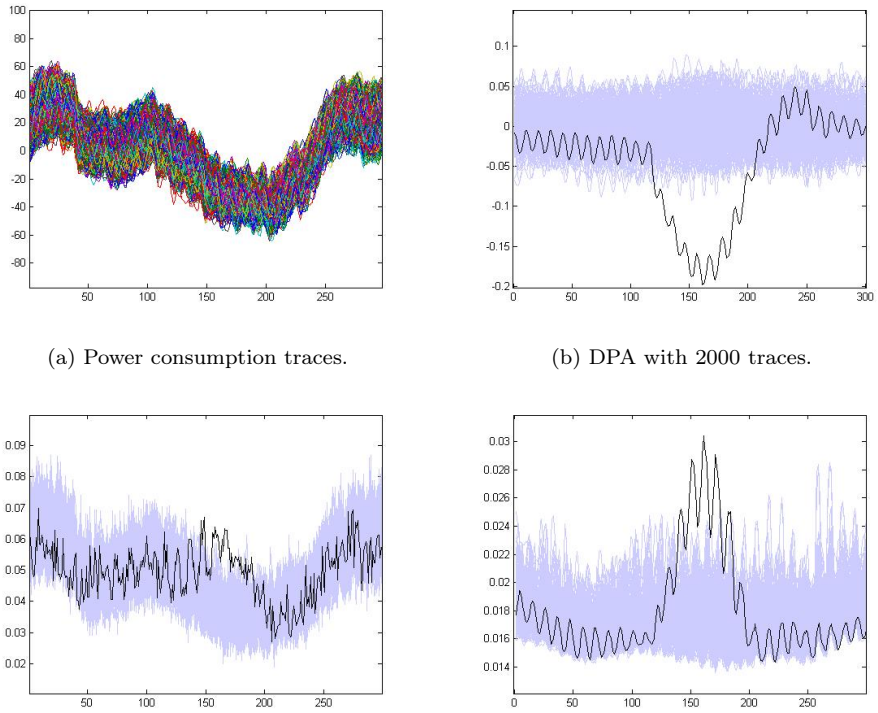
(d) Parametric MIA with 1024 traces.

Figure 5.5: Practical attacks on a hardware AES implementation.

For both set of traces, we performed DPA and MIA attacks with the histogram estimation method and the parametric estimation method (see Section 5.4). For all of these attacks the prediction function $\hat{\varphi}$ was chosen to be the Hamming weight function (since $\hat{\varphi} \circ f_k$ must be non-injective – see Corollary 5.1 –). The obtained correlation and mutual information curves are plotted in Fig. 5.5(b–d) and Fig. 5.6(b–d) over the time. For each attack the curve corresponding to the correct (resp. wrong) key hypothesis is drawn in black (resp. gray).

In both cases, the attacks succeed with a few number of traces. It can be noticed that MIA with a parametric estimation is more discriminating than MIA with the histogram estimation. This confirms the simulations performed in Section 5.5.1. However, even when the parametric estimation method is involved, DPA is always more discriminating than MIA. Those results suggest that the power consumption of the attacked devices has in fact a high linear dependency with the Hamming weight of the manipulated data. This implies in particular that the Hamming weight model is sound in this context and that looking for non-linear dependencies is not useful.

To corroborate that the leakage measured in Figure 5.5 and 5.6 is close to the one simulated in Section 5.4, we plotted in Fig. 5.7 the estimation of the pdf $g_{\mathbf{L}(0)|Z(k)=1}$ when $k = 0 = k^*$ and $k = 5 \neq k^*$ for the hardware implementation. We could verify that actually the conditional pdfs that are estimated look like Gaussian mixture pdfs (a Gaussian pdf when k^* is correctly guessed and a mixture of two pdfs when



(a) Power consumption traces.

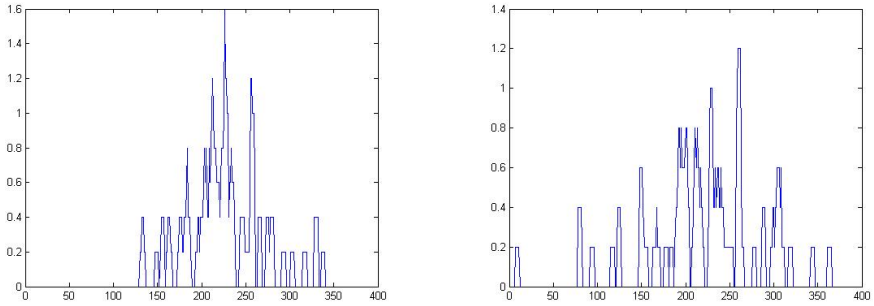
(b) DPA with 2000 traces.

(c) Histogram-based MIA with 2000 traces.

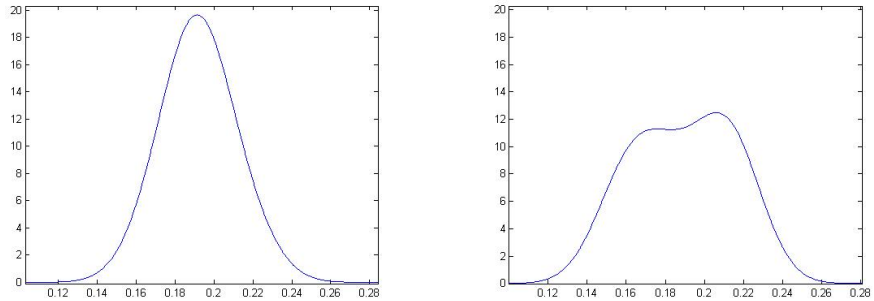
(d) Parametric MIA with 2000 traces.

Figure 5.6: Practical attacks on a software AES implementation.

it is not).



(a) Histogram estimation ($k = k^*$). (b) Histogram estimation ($k \neq k^*$).



(c) Parametric estimation ($k = k^*$). (d) Parametric estimation ($k \neq k^*$).

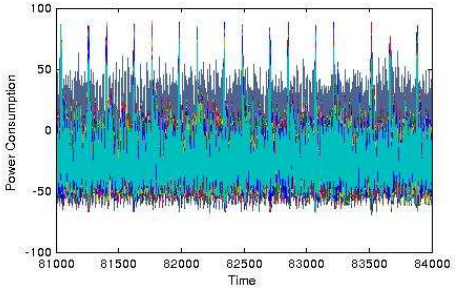
Figure 5.7: Pdf estimations on power measurements.

5.5.3.2 Second Order Attack.

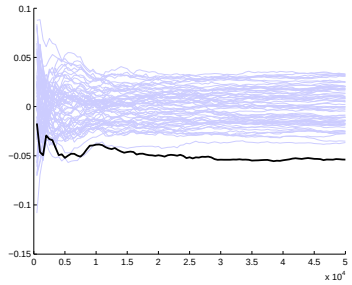
We performed second-order MIA attacks against a DES S-box implementation that uses a lookup-table and is protected by first-order masking. Namely, the targeted variable $f_{k^*}(X)$ corresponds to the DES S-box output and the leakage measurement consists in two points L_0 and L_1 satisfying (5.9) for $d = 2$. The power consumption traces have been measured for a software implementation running on a smart card with 8-bit architecture. They are plotted in Fig. 5.8(a). The traces are composed of 3000 points and we identified that the masked value $f_{k^*}(X) \oplus M$ is manipulated at time $t_0 = 81789$ whereas the mask M is manipulated at time $t_1 = 83238$. We also performed a second-order CPA attack involving the normalized product combining with $\hat{\varphi} = \mathrm{HW}$. For the MIA attacks, we choose to define the prediction function $\hat{\varphi}$ either as the identity function or as the Hamming weight function. For the histogram-based MIA, we used the Scott's rule for the bin-width. For the kernel-based MIA, we applied the normal scale rule recalled in (5.14) to select the kernels bandwidth.

Our attacks results for $\hat{\varphi}$ being the identity function are plotted in Fig. 5.8(c–d). For each key-candidate, the mutual information/correlation values are plotted over the number of leakage measurements. The curve corresponding to the correct (resp. wrong) key hypothesis is drawn in black (resp. gray). In Fig. 5.9, we plotted for each attack the rank of the good key hypothesis according to the number of traces exploited by the attack. The dot-line corresponds to the second-order DPA attack. Black curves refer to 2O-MIA$_{\mathrm{K}}$ attacks whereas gray curves refer to 2O-MIA$_{\mathrm{H}}$ attacks. In both cases, plain-lines correspond to attacks with $\hat{\varphi} = \mathrm{Id}$ and dashed-lines correspond to $\hat{\varphi} = \mathrm{HW}$.

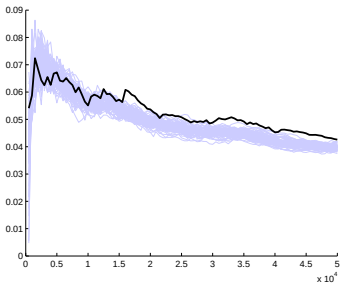
As we can see from Fig. 5.9, the obtained results validate our simulations. In particular, we see that second-order DPA is clearly more efficient than second-order MIA and that kernel-based MIA is better than histogram-based MIA. We further observe that compared to our simulations where the Hamming



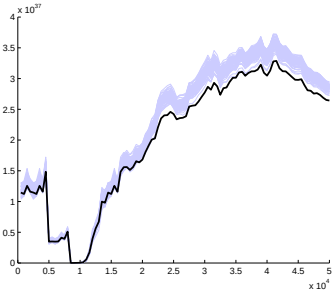
(a) Power consumption traces.



(b) 2O-DPA, $\hat{\varphi} = \mathrm{HW}$.



(c) 2O-MIA$_{\mathrm{H}}$, $\hat{\varphi} = \mathrm{Id}$.



(d) 2O-MIA$_{\mathrm{K}}$, $\hat{\varphi} = \mathrm{Id}$.

Figure 5.8: Practical second-order attacks on a software DES implementation.

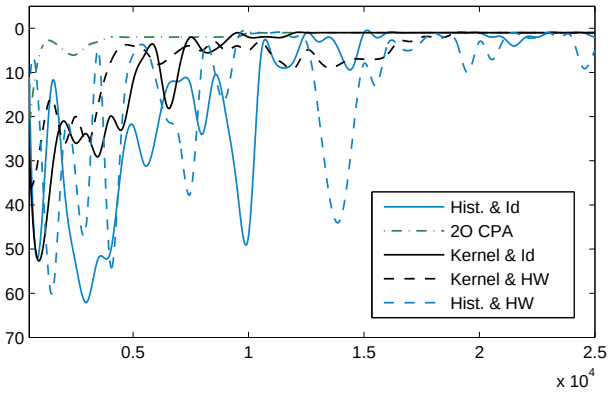


Figure 5.9: Rank of the good key hypothesis according to the attack type

weight prediction leads to better efficiency than the identity prediction, both predictions lead to similar results for our practical attacks. This suggests that the power consumption of the attacked device does not fully depend on the Hamming weight of the processed data but rather on some leakage function between Hamming weight and identity (*e.g.* each bit of the data has a different weight in the power consumption).

Chapter 6

Analysis and improvement of higher-order differential power analysis

Contents

6.1 Introduction

In this chapter, we investigate higher-order DPA that involve a combining function (see Section 3.7). We first exhibit the optimal prediction function for such attacks. Then we study second-order DPA involving the product combining [65] or the absolute difference combining [164]. We study them under the Hamming weight model and the Gaussian noise assumptions (see Section 3.4.3). After showing a way to improve the product combining, we argue that in this model, the product combining is more efficient not only than absolute difference combining, but also than all the other combining techniques proposed in the literature. Eventually, we analyze higher-order DPA involving the improved product combining to attack masking at any order and possibly combined with shuffling.

The results presented in this chapter have been published in collaboration with Emmanuel Prouff and Régis Bévan in the journal *IEEE Transactions on Computers vol. 58 no. 6 (June 2009)* [6] and in collaboration with Emmanuel Prouff and Julien Doget in the international workshop on *Cryptographic Hardware and Embedded Systems (CHES 2009)* [11].

6.2 Attack efficiency metric

We recall that when d^{th}-order masking is involved, every sensitive variable Z is split into $d+1$ shares $M_0, \ldots, M_d$ such that $M_0 \star \cdots \star M_d = Z$ for a group operation $\star$. In this chapter, we focus on *Boolean*

masking i.e. using the XOR as masking operation (which is the most widely used to protect block ciphers in practice). The attacker combines the $d+1$ leakage signals $L_0, \ldots, L_d$ corresponding to the $d+1$ shares respectively in order to create a combined signal that is correlated to the target variable. This is done by means of a combining function $\mathcal{C}$. The attack then consists in estimating the correlation between the combined leakage $\mathcal{C}(L_0, \ldots, L_d)$ and the prediction $\hat{\varphi} \circ f(X, k)$ for every key guesses $k \in \mathcal{K}$.

As argued in several works (see for instance [6, 156, 157, 204]), the absolute value of the correlation coefficient $\rho\left[\hat{\varphi}(Z), \mathcal{C}(L_1, \ldots, L_d)\right]$ (*i.e.* corresponding to the correct key guess) is a sound estimator of the efficiency of a correlation based (higher-order) DPA characterized by the pair of functions $(\hat{\varphi}, \mathcal{C})$. In [157, 220], it is even shown that the number of leakage measurements required for the attack to succeed can be approximated by $c \cdot \rho\left[\hat{\varphi}(Z), \mathcal{C}(L_1, \ldots, L_d)\right]^{-2}$ where c is a constant depending on the number of key guesses and on the required success rate. In this chapter, we will therefore compare attack efficiencies by means of the correlation values $\rho\left[\hat{\varphi}(Z), \mathcal{C}(L_1, \ldots, L_d)\right]$. For a given higher-order DPA attack, we will refer to this coefficient as the correlation of the attack: the higher the correlation of an attack, the more efficient the attack.

6.3 Optimal prediction function

Let us begin our discussion with the following important result which will be intensively used in the rest of the chapter. In the following proposition as well as in the rest of the chapter, we shall consider the conditional expectation $\mathrm{E}\left[\mathcal{C}|Z\right]$ as a function $\mathrm{E}\left[\mathcal{C}|\cdot\right]$ applied to Z.

Proposition 6.1. *Let $\mathcal{C}$ and Z be two random variables. Then, for every function $\hat{\varphi}$ defined over $\mathcal{Z}$, we have*

$$\rho\left[\hat{\varphi}(Z), \mathcal{C}\right] = \rho\left[\hat{\varphi}(Z), \mathrm{E}\left[\mathcal{C}|Z\right]\right] \times \rho\left[\mathrm{E}\left[\mathcal{C}|Z\right], \mathcal{C}\right] \ . \tag{6.1}$$

Before proving Proposition 6.1, let us introduce the following useful lemma.

Lemma 6.1. *Let $\mathcal{C}$ and Z be two random variables. Then, for every function $\hat{\varphi}$ defined over $\mathcal{Z}$, we have*

$$\mathrm{E}\left[\hat{\varphi}(Z)\mathcal{C}\right] = \mathrm{E}\left[\hat{\varphi}(Z)\mathrm{E}\left[\mathcal{C}|Z\right]\right] \ . \tag{6.2}$$

Proof. We assume that $\mathcal{C}$ and Z are discrete (the continuous case holds straightforwardly from the discrete one). We have:

$$\mathrm{E}\left[\hat{\varphi}(Z)\mathcal{C}\right] = \sum_{z,c} \mathrm{P}\left[Z = z, \mathcal{C} = c\right] \hat{\varphi}(z) c \ . \tag{6.3}$$

Since $\mathrm{P}\left[Z = z, \mathcal{C} = c\right]$ equals $\mathrm{P}\left[Z = z\right] \mathrm{P}\left[\mathcal{C} = c|Z = z\right]$, we get:

$$\begin{aligned} \mathrm{E}\left[\hat{\varphi}(Z)\mathcal{C}\right] &= \sum_{z} \mathrm{P}\left[Z = z\right] \hat{\varphi}(z) \sum_{c} \mathrm{P}\left[\mathcal{C} = c|Z = z\right] c \\ &= \sum_{z} \mathrm{P}\left[Z = z\right] \hat{\varphi}(z) \mathrm{E}\left[\mathcal{C}|Z = z\right] \ , \end{aligned}$$

which leads to (6.2). □

Remark 6.1. Lemma 6.1 implies $\mathrm{E}\left[\mathcal{C}\right] = \mathrm{E}\left[\mathrm{E}\left[\mathcal{C}|Z\right]\right]$ (for $f : z \mapsto 1$), which is known as the law of total expectation, and it implies $\mathrm{E}\left[\mathrm{E}\left[\mathcal{C}|Z\right]\mathcal{C}\right] = \mathrm{E}\left[\mathrm{E}\left[\mathcal{C}|Z\right]^2\right]$ (for $f : z \mapsto \mathrm{E}\left[\mathcal{C}|Z = z\right]$).

Based on Lemma 6.1, we give hereafter the proof of Proposition 6.1.

Proof. (Proposition 6.1) According to Remark 6.1, the covariance between $\hat{\varphi}(Z)$ and $\mathcal{C}$ satisfies:

$$\begin{aligned} \mathrm{Cov}\left[\hat{\varphi}(Z), \mathcal{C}\right] &= \mathrm{E}\left[\hat{\varphi}(Z)\mathrm{E}\left[\mathcal{C}|Z\right]\right] - \mathrm{E}\left[\hat{\varphi}(Z)\right] \mathrm{E}\left[\mathrm{E}\left[\mathcal{C}|Z\right]\right] \\ &= \mathrm{Cov}\left[\hat{\varphi}(Z), \mathrm{E}\left[\mathcal{C}|Z\right]\right] \ . \end{aligned}$$

Hence, the correlation $\rho\left[\hat{\varphi}(Z), \mathcal{C}\right]$ satisfies:

$$\rho\left[\hat{\varphi}(Z), \mathcal{C}\right] = \rho\left[\hat{\varphi}(Z), \mathrm{E}\left[\mathcal{C}|Z\right]\right] \times \frac{\sigma\left[\mathrm{E}\left[\mathcal{C}|Z\right]\right]}{\sigma\left[\mathcal{C}\right]} \ . \tag{6.4}$$

On the other hand, we have:

$$\rho\left[\mathrm{E}\left[\mathcal{C}|Z\right], \mathcal{C}\right] = \frac{\mathrm{Cov}\left[\mathrm{E}\left[\mathcal{C}|Z\right], \mathcal{C}\right]}{\sigma\left[\mathrm{E}\left[\mathcal{C}|Z\right]\right]\sigma\left[\mathcal{C}\right]} \ . \tag{6.5}$$

Due to Lemma 6.1, the covariance $\mathrm{Cov}\left[\mathrm{E}\left[\mathcal{C}|Z\right], \mathcal{C}\right]$ equals $\mathrm{Cov}\left[\mathrm{E}\left[\mathcal{C}|Z\right], \mathrm{E}\left[\mathcal{C}|Z\right]\right]$, namely it equals the variance $\mathrm{Var}\left[\mathrm{E}\left[\mathcal{C}|Z\right]\right]$. Hence, (6.4) and (6.5) together imply (6.1). □

As a direct consequence of Proposition 6.1, we have the next corollary.

Corollary 6.1. *Let $\mathcal{C}$ denote a combined leakage $\mathcal{C}(L_1, \ldots, L_d)$. The prediction function $\hat{\varphi}$ that maximizes the correlation $\rho\left[\hat{\varphi}(Z), \mathcal{C}\right]$ is defined by*

$$\hat{\varphi}_{opt}(z) = \mathrm{E}\left[\mathcal{C}|Z = z\right] \ . \tag{6.6}$$

Let ρ_{opt} be the correlation $\rho\left[\hat{\varphi}_{opt}(Z), \mathcal{C}\right]$. If $\hat{\varphi}_{opt}$ is not constant, then ρ_{opt} satisfies:

$$\rho_{opt} = \frac{\sigma\left[\mathrm{E}\left[\mathcal{C}|Z\right]\right]}{\sigma\left[\mathcal{C}\right]} \ . \tag{6.7}$$

Proof. Let $\hat{\varphi}$ be a function defined over $\mathcal{Z}$ and let ρ' denote the correlation $\rho\left[\hat{\varphi}(Z), \mathcal{C}\right]$. Then, due to Proposition 6.1, we have $\rho' = \rho\left[\hat{\varphi}(Z), \mathrm{E}\left[\mathcal{C}|Z\right]\right] \times \rho_{opt}$. As $\rho\left[\hat{\varphi}(Z), \mathrm{E}\left[\mathcal{C}|Z\right]\right]$ is always smaller than or equal to 1 and since ρ_{opt} is greater than or equal to 0, we deduce $\rho' \leqslant \rho_{opt}$. This implies that the function $f = f_{opt} : z \mapsto \mathrm{E}\left[\mathcal{C}|Z = z\right]$ maximizes ρ'. Finally, (6.7) holds by definition of $\hat{\varphi}_{opt}$ and by Lemma 6.1. □

Corollary 6.1 exhibits the optimal prediction function $\hat{\varphi}_{opt}$ and the optimal correlation of an HO-DPA given a combining function and the leakage distribution. Moreover, Proposition 6.1 gives us a means to quantify the effectiveness loss occurring when a sub-optimal function $\hat{\varphi}$ is involved. Indeed, in this case (6.1) implies that making a suboptimal prediction $\hat{\varphi}$ decreases the optimal correlation ρ_{opt} by a factor $\rho\left[\hat{\varphi}, \hat{\varphi}_{opt}\right]$.

In practice, the kind of adversary considered in this chapter is not able to compute the optimal prediction function exhibited in Corollary 6.1. Indeed, such a computation requires to determine the exact relationship between the leakages L_i's and the shares M_i's. In the next section, we will estimate this relationship by modeling the leakage and then we will study the optimal prediction function and the optimal correlation for two widely used second-order combining functions. We will show that some prediction functions proposed in the literature are in fact sub-optimal and we will compute how much they decrease the correlation ρ_{opt} (and thus the attack efficiency) from the optimal one defined in (6.7).

6.4 Analysis and improvement of second-order combining functions

The different second-order DPA that are studied in this section are assumed to target an implementation that processes a masked sensitive variable $Z \oplus M$ at a time t_1 and the corresponding mask M at a time t_2. Variables Z and M are assumed to be mutually independent and uniformly distributed over $\mathbb{F}_2^n$.

Studying a second-order DPA essentially amounts to studying the combining function it involves. Hereafter, we pay particular attention to the product combining [65] and to the absolute difference combining [164] which are the most widely used functions in the literature. For both combining functions, we exhibit the optimal prediction $\hat{\varphi}_{opt}$ and we calculate the optimal correlation ρ_{opt} by applying (6.7). We also compare $\hat{\varphi}_{opt}$ with the Hamming weight prediction function (which is often involved in the published HO-DPA) and we study their impact on the attack efficiency. Eventually, we analyze the obtained results and we address other combining functions that have been proposed in the literature.

Before presenting our analysis (and to allow us to exhibit explicit formulae), we need to make the following assumption which we claim is very common and realistic.

Assumption 6.1 (Leakage Model)**.** *The leakages L_1 and L_2 satisfy:*

$$L_1 = \delta_1 + \mathrm{HW}(Z \oplus M) + B_1 \ , \tag{6.8}$$
$$L_2 = \delta_2 + \mathrm{HW}(M) + B_2 \ , \tag{6.9}$$

where δ_1 and δ_2 denote the constant parts of the leakages and $\mathrm{HW}(\cdot)$ is the Hamming weight function. B_1 and B_2 are two Gaussian random variables centered in zero with a standard deviation σ and Z, M, B_1 and B_2 are mutually independent[10]*.*

Remark 6.2. In some cases, it may be sound to assume that the device does not leak the Hamming weight of the processed data but the Hamming distance between this data and an initial state (see for instance Section 3.4.3). Extending our analysis to the Hamming distance model is straightforward. Let L_1 equal $\delta_1 + \mathrm{HW}(IS_1 \oplus Z \oplus M) + B_1$ and L_2 equal $\delta_2 + \mathrm{HW}(IS_2 \oplus M) + B_2$ where IS_1 and IS_2 are two initial states independent of Z and M. After denoting by Z' the summation $IS_1 \oplus IS_2 \oplus Z$ and by M' the summation $M \oplus IS_2$, it can be checked that L_1 and L_2 respectively equal $\delta_1 + \mathrm{HW}(Z' \oplus M') + B_1$ and $\delta_2 + \mathrm{HW}(M') + B_2$. As Z' and M' are uniformly distributed and mutually independent, this model is equivalent to the one defined in Assumption 6.1.

When the noises B_1 and B_2 are both zero, we shall say that the model is *idealized*. The analysis of second-order DPA in this model is of interest. Firstly because some devices leak quite perfect non-noisy information. Secondly because it is generic (it does not take the component noise into account) and theoretical analyses conducted in this model are usually simple. In such an idealized model, exhibiting relevant properties and/or characteristics for new combining and prediction functions $(\mathcal{C}, \hat{\varphi})$ is often much more simple than in a model with noise. However, this primary study is not sufficient alone and, once defined in the idealized model, a pair of functions $(\mathcal{C}, \hat{\varphi})$ must also be analyzed in the noisy model. Indeed, the combining of leakage points always results in an amplification of the noise (*e.g.* the noises B_1 and B_2 are added or multiplied) and it is therefore important to study the relationship between the efficiency of a combining function and the noise variations. For this reason, in the following we conduct our analysis in the context of both the idealized and the non-idealized model.

6.4.1 Product combining second-order DPA

In this section we investigate the product combining function:

$$\mathcal{C}_{prod}(L_1, L_2) = L_1 \times L_2 \ . \tag{6.10}$$

This function has already been studied by Kai Schramm and Christof Paar in [204]. Our main contribution compared to their work is that we consider a leakage model where the offsets δ_i are not zero. This makes our analysis more practical since the leakage often has a non-zero offset due to the contribution of the device activity aside from the variable manipulation. During our study we show in particular that the efficiency of the product combining is related to the values of these offsets and we show how to significantly improve it by applying a pre-processing to the leakage signals before combining them.

Let us start our analysis by computing the optimal prediction function corresponding to $\mathcal{C}_{prod}$. According to Corollary 6.1, it is the function $\hat{\varphi}_{opt} = z \mapsto \mathrm{E}[L_1 \times L_2 | Z = z]$. In the next proposition we give an explicit formula for it.

Proposition 6.2. *Let L_1 and L_2 satisfy* (6.8) *and* (6.9)*. Then, for every $z \in \mathbb{F}_2^n$, we have*

$$\mathrm{E}[L_1 \times L_2 | Z = z] = -\frac{1}{2}\mathrm{HW}(z) + \frac{n^2 + n}{4} + \frac{n}{2}(\delta_1 + \delta_2) + \delta_1\delta_2 \ . \tag{6.11}$$

Before giving the proof of Proposition 6.2, let us introduce a useful lemma.

[10] For the sake of simplicity we assume that both noises B_1 and B_2 have the same standard deviation. The analysis can be straightforwardly generalized for $\sigma[B_1] \neq \sigma[B_2]$.

Lemma 6.2. *Let n be a positive integer and let M be a random variable uniformly distributed over $\mathbb{F}_2^n$. Then, for every $z \in \mathbb{F}_2^n$, we have:*

$$\mathrm{E}\left[\mathrm{HW}(z \oplus M)\mathrm{HW}(M)\right] = -\frac{1}{2}\mathrm{HW}(z) + \frac{n^2+n}{4} \ . \tag{6.12}$$

A generalized version of Lemma 6.2 is given and proved in Section 6.5 (see Lemma 6.3). We now give the proof of Proposition 6.2.

Proof. (Proposition 6.2) Since B_1 and B_2 are independent from M and satisfy $\mathrm{E}[B_1] = \mathrm{E}[B_2] = 0$, the expectation $\mathrm{E}[L_1 \times L_2 | Z = z]$ is equal to $\mathrm{E}\left[\mathrm{HW}(z \oplus M)\mathrm{HW}(M)\right] + \delta_1 \mathrm{E}\left[\mathrm{HW}(M)\right] + \delta_2 \mathrm{E}\left[\mathrm{HW}(z \oplus M)\right] + \delta_1\delta_2$. Moreover, since M is uniformly distributed over $\mathbb{F}_2^n$, we have $\mathrm{E}\left[\mathrm{HW}(z \oplus M)\right] = \mathrm{E}\left[\mathrm{HW}(M)\right] = \frac{n}{2}$ and, from Lemma 6.2, we have $\mathrm{E}\left[\mathrm{HW}(z \oplus M)\mathrm{HW}(M)\right] = -\frac{1}{2}\mathrm{HW}(z) + \frac{n^2+n}{4}$. Hence we get (6.11). □

Proposition 6.2 together with Corollary 6.1 implies that the function $z \mapsto \mathrm{HW}(z)$, or any decreasing affine function of it, may be used as an optimal prediction function for a second-order DPA involving the product combining.

Corollary 6.2. *In the Hamming weight model, the optimal prediction function $\hat{\varphi}_{opt}$ corresponding to $\mathcal{C}_{prod}$ is of the form:*

$$\hat{\varphi}_{opt} : z \mapsto A \circ \mathrm{HW}(z) \ , \tag{6.13}$$

where A is an affine decreasing function defined over $\mathrm{HW}(\mathcal{Z})$.

Proof. Corollary 6.2 is a straightforward consequence of Corollary 6.1 and of Proposition 6.2. □

It must be noticed that the Hamming weight function has already been used as prediction function in previous works [176,204]. Corollary 6.2 shows that this choice maximizes the amplitude of the correlation coefficient (in the Hamming weight model) and that it results in a negative correlation (as observed in [176] for instance).

To compute the optimal correlation corresponding to one of the function satisfying (6.13), we exhibit in the following a formula for the variance of $L_1 \times L_2$.

Proposition 6.3. *Let L_1 and L_2 satisfy* (6.8) *and* (6.9)*. Then, the variance of $L_1 \times L_2$ satisfies*

$$\mathrm{Var}\left[L_1 \times L_2\right] = \frac{2n^3+n^2}{16} + \frac{n}{4}\left(n\delta_1 + \delta_1^2 + n\delta_2 + \delta_2^2\right) + \frac{n^2+n}{2}\sigma^2 + \left(n\delta_1 + \delta_1^2 + n\delta_2 + \delta_2^2\right)\sigma^2 + \sigma^4 \ . \tag{6.14}$$

Proof. As Z and M are mutually independent and uniformly distributed, one can check that M and $Z \oplus M$ are mutually independent. This implies that L_1 and L_2 are also mutually independent and we get:

$$\mathrm{Var}\left[L_1 \times L_2\right] = \mathrm{E}\left[L_1^2\right]\mathrm{E}\left[L_2^2\right] - \mathrm{E}\left[L_1\right]^2\mathrm{E}\left[L_2\right]^2 \ . \tag{6.15}$$

Since Z and M are uniformly distributed over $\mathbb{F}_2^n$ and mutually independent we have $\mathrm{E}\left[\mathrm{HW}(M)^2\right] = \mathrm{E}\left[\mathrm{HW}(Z \oplus M)^2\right] = \frac{n^2+n}{4}$. Then, since we have $B_i \sim \mathcal{N}(0, \sigma)$, one deduces that $\mathrm{E}[L_i]$ and $\mathrm{E}\left[L_i^2\right]$ equal respectively $\frac{n}{2} + \delta_i$ and $\frac{n^2+n}{4} + n\delta_i + \delta_i^2 + \sigma^2$ for $i = 1, 2$. Finally, simplifying (6.15) leads to (6.14). □

It can be noticed in (6.14) that $\mathrm{Var}\left[L_1 \times L_2\right]$ is an increasing function of $n\delta_1 + \delta_1^2 + n\delta_2 + \delta_2^2$. Hence the offsets values that minimize the variance are $\delta_1 = \delta_2 = -n/2$. Actually, this is not surprising: with such offsets, the leakages are centered in zero (*i.e.* $\mathrm{E}[L_1] = \mathrm{E}[L_2] = 0$) which alleviates the noise amplification caused by the product combining. As a direct consequence, minimizing the variance of $L_1 \times L_2$ (and thus maximizing the correlation) can be done by centering the leakage signals L_1 and L_2 in zero (namely by substituting $L_i - \mathrm{E}[L_i]$ for L_i). This can be simply achieved by averaging the leakage for a large number of measurements then subtracting the average to each measurements. In the sequel, this pre-processing is called *normalization step*.

In the Hamming weight model, if variable D_i manipulated at time t_i is uniformly distributed over $\mathbb{F}_2^n$, then the leakage after the pre-processing step equals $L_i - \mathrm{E}[L_i]$ and satisfies:

$$L_i - \mathrm{E}\left[L_i\right] = -\frac{n}{2} + \mathrm{HW}(D_i) + B_i \ .$$

After assuming that the pre-processing step is part of the combining computation, we get the improved product combining function:

$$\mathcal{C}_{prod^\star}(L_1, L_2) = (L_1 - \mathrm{E}[L_1]) \times (L_2 - \mathrm{E}[L_2]) \ .$$

Then, we have the following proposition.

Proposition 6.4. *For every $z \in \mathbb{F}_2^n$, we have:*

$$\mathrm{E}[\mathcal{C}_{prod^\star}(L_1, L_2) | Z = z] = -\frac{1}{2}\mathrm{HW}(z) + \frac{n}{4} \ ,$$

and,

$$\mathrm{Var}[\mathcal{C}_{prod^\star}(L_1, L_2)] = \frac{n^2}{16} + \frac{n}{2}\sigma^2 + \sigma^4 \ .$$

Proof. Proposition 6.4 straightforwardly results from Proposition 6.2 and Proposition 6.3 by setting $\delta_1 = \delta_2 = -n/2$. □

As a consequence of the proposition above, in the Hamming weight model, an optimal prediction function $\hat{\varphi}_{opt}$ corresponding to $\mathcal{C}_{prod^\star}$ is of the form:

$$\hat{\varphi}_{opt} : z \mapsto A \circ \mathrm{HW}(z) \ ,$$

where A is an affine decreasing function defined over $\mathrm{HW}(\mathcal{Z})$.

Due to Proposition 6.4 and Corollary 6.1, we can propose an explicit formula for the optimal correlation $\rho_{opt}^{prod^\star}$ corresponding to the improved product combining $\mathcal{C}_{prod^\star}$ and $\hat{\varphi}_{opt}$. In the Hamming weight, the correlation satisfies

$$\rho_{opt}^{prod^\star} = \frac{\sqrt{n}}{\sqrt{n^2 + 8n\sigma^2 + 16\sigma^4}} \ . \tag{6.16}$$

In particular, in the idealized model ($\sigma = 0$) it satisfies $\rho_{opt}^{prod^\star} = 1/\sqrt{n}$ and in the *very* noisy model ($\sigma \gg n$), it satisfies $\rho_{opt}^{prod^\star} \approx \sqrt{n}/4\sigma^2$. As an illustration to (6.16), Table 6.1 gives some values of the correlation for $n \in \{0, \ldots, 8\}$ and $\sigma \in \{0, 1, 5, 10\}$.

Table 6.1: (Optimal) correlation for the improved product combining

$\sigma \backslash n$	1	2	3	4	5	6	7	8
0	1.00	0.707	0.577	0.500	0.447	0.408	0.378	0.354
1	0.200	0.236	0.247	0.250	0.248	0.245	0.241	0.236
5	0.010	0.014	0.017	0.019	0.021	0.023	0.025	0.026
10	0.002	0.004	0.004	0.005	0.006	0.006	0.007	0.007

To illustrate the gain of efficiency resulting from the normalization step we propose in this chapter, let us now consider the correlation ρ_{opt}^{prod-0} for the classical product combining function (6.10) in the Hamming weight model without offsets (such as computed in [204]). It satisfies:

$$\rho_{opt}^{prod-0} = \frac{\sqrt{n}}{\sqrt{2n^3 + n^2 + 8(n^2 + n)\sigma^2 + 16\sigma^4}} \ .$$

It can be checked that ρ_{opt}^{prod-0} is strictly lower than the correlation $\rho_{opt}^{prod^\star}$ we obtained for the product combining with pre-processing $\mathcal{C}_{prod^\star}$. Figures 6.1 and 6.2 show how the value of the offsets (assuming $\delta_1 = \delta_2 = \delta$) affects the correlation ρ_{opt}^{prod} for $n \in \{1, 4, 8\}$ in the idealized model and in a noisy model ($\sigma = 2$). The maximum of this correlation is always reached for $\delta = -n/2$. Moreover, we observe that the correlation quickly decreases when the offset deviates from $-n/2$ which demonstrates the effectiveness of our improvement.

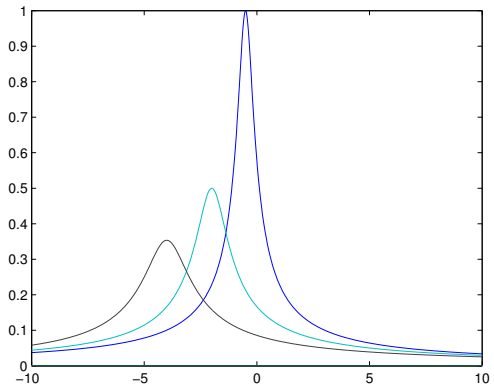


Figure 6.1: Correlation ρ_{opt}^{prod} for $n = 8$ (on the left), $n = 4$ (in the middle) and $n = 1$ (on the right), in the idealized model, according to the offset δ.

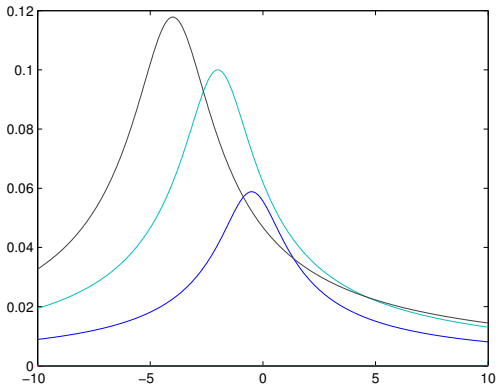


Figure 6.2: Correlation ρ_{opt}^{prod} for $n = 8$ (on the left), $n = 4$ (in the middle) and $n = 1$ (on the right), in a noisy model ($\sigma = 2$), according to the offset δ.

Remark 6.3. It was already noted in [65] that such a normalization step should be performed before combining the leakages by product. However, [65] suggests to subtract the offset δ_i to the leakage while we show that subtracting $\mathrm{E}[L_i] = \delta_i + \frac{n}{2}$ is optimal (in the Hamming weight model with uniform data).

6.4.2 Absolute difference combining second-order DPA

In this section, we investigate the absolute difference combining function *i.e.* we take interest in the variable

$$\mathcal{C}_{diff}(L_1, L_2) = |L_1 - L_2| .$$

The absolute difference combining has already been studied by Marc Joye, Pascal Paillier, and Berry Schoenmakers in [133]. In their paper, the authors consider the idealized model (*i.e.* without noise) and analyze a single-bit second-order DPA (*i.e.* with a binary prediction function: $\hat{\varphi}(Z) \in \{0, 1\}$). We extend hereafter this analysis to the multi-bit case (*i.e.* where $\hat{\varphi}$ is not a binary function but the optimal prediction function) not only in the idealized but also in the noisy model. In the Hamming weight model, $\mathcal{C}_{diff}(L_1, L_2)$ equals $|\delta_1 - \delta_2 + \mathrm{HW}(Z \oplus M) - \mathrm{HW}(M) + B_1 - B_2|$. For this combining to work correctly, it is important that δ_1 be equal to δ_2. Indeed, if there is a great difference between these values, then the effect of the absolute value is reduced (or even canceled) by the constant term $\delta_1 - \delta_2$. For instance (neglecting the noise), if we have $|\delta_1 - \delta_2| > n$ then $\delta_1 - \delta_2 + \mathrm{HW}(Z \oplus M) - \mathrm{HW}(M)$ is either strictly positive or strictly negative and, as noticed in [164], difference without absolute value is not a sound combining function (*i.e.* the difference between the two leakages is not correlated to the sensitive variable). Consequently, as for the product combining, we point out that the leakages must be normalized in order to have identical offsets in both leakage signals. Thus, as in Section 6.4.1, we will consider in this section that the leakages are normalized before being combined in order to ensure that they have similar offsets (*i.e.* we define the combining function $\mathcal{C}_{diff^\star}$ such that $\mathcal{C}_{diff^\star}(L_1, L_2) = |L_1 - \mathrm{E}[L_1] - L_2 + \mathrm{E}[L_2]|$). In that case, the combined leakage after pre-processing satisfies

$$\mathcal{C}_{diff^\star}(L_1, L_2) = |\mathrm{HW}(Z \oplus M) - \mathrm{HW}(M) + B| , \tag{6.17}$$

where B denotes $B_1 - B_2$ and satisfies $B \sim \mathcal{N}(0, \sqrt{2}\sigma)$.

For the absolute difference combining, it is not possible to exhibit a simple formula for the expectation that would be relevant in the general case. Hence we structure our study of the combining function in two steps: the first one is performed in the idealized model and the second one in the noisy model.

6.4.2.1 Study in the idealized model

If B is zero, then (6.17) becomes:

$$\mathcal{C}_{diff^\star}(L_1, L_2) = |\mathrm{HW}(Z \oplus M) - \mathrm{HW}(M)| \ .$$

In the following proposition, we exhibit an explicit formula for the expectation of $|\mathrm{HW}(Z \oplus M) - \mathrm{HW}(M)|$.

Proposition 6.5. *Let z be an element of $\mathbb{F}_2^n$. Then we have*

$$\mathrm{E}\left[|\mathrm{HW}(M) - \mathrm{HW}(z \oplus M)|\right] = 2^{1-\mathrm{HW}(z)}\mathrm{HW}(z)\binom{\mathrm{HW}(z)-1}{\lfloor \frac{\mathrm{HW}(z)}{2} \rfloor} \ . \tag{6.18}$$

Proof. For every pair $(z, m) \in \mathbb{F}_2^n$, Property 2.2 implies $|\mathrm{HW}(z \oplus m) - \mathrm{HW}(m)| = |\mathrm{HW}(z) - 2\mathrm{HW}(z \wedge m)|$ from which we deduce:

$$\mathrm{E}\left[|\mathrm{HW}(z \oplus M) - \mathrm{HW}(M)|\right] = \sum_{i=0}^{\mathrm{HW}(z)} |\mathrm{HW}(z) - 2i| \ \mathrm{P}\left[\mathrm{HW}(z \wedge M) = i\right] \ . \tag{6.19}$$

Since M is uniformly distributed $\mathrm{P}\left[\mathrm{HW}(z \wedge M) = i\right]$ equals $2^{-\mathrm{HW}(z)}\binom{h}{i}$. Hence we deduce

$$\mathrm{E}\left[|\mathrm{HW}(z \oplus M) - \mathrm{HW}(M)|\right] = 2^{-\mathrm{HW}(z)} \sum_{i=0}^{\lfloor \frac{\mathrm{HW}(z)}{2} \rfloor} \binom{\mathrm{HW}(z)}{i} (\mathrm{HW}(z) - 2i) \ . \tag{6.20}$$

By symmetry, we have $\sum_{i=0}^{\lfloor \frac{\mathrm{HW}(z)}{2} \rfloor} \binom{\mathrm{HW}(z)}{i}$ equal to $\frac{1}{2}\left(\sum_{i=0}^{\mathrm{HW}(z)} \binom{\mathrm{HW}(z)}{i} + \binom{\mathrm{HW}(z)}{\frac{\mathrm{HW}(z)}{2}}(\mathrm{HW}(z) \mod 2)\right)$. Then $\sum_i \binom{\mathrm{HW}(z)}{i} = 2^{\mathrm{HW}(z)}$ implies

$$\sum_{i=0}^{\lfloor \frac{\mathrm{HW}(z)}{2} \rfloor} \binom{\mathrm{HW}(z)}{i} = 2^{\mathrm{HW}(z)-1} + \frac{1}{2}\binom{\mathrm{HW}(z)}{\frac{\mathrm{HW}(z)}{2}}(\mathrm{HW}(z) + 1 \mod 2) \ . \tag{6.21}$$

On the other hand $\binom{\mathrm{HW}(z)}{i} i$ equals $\mathrm{HW}(z)\binom{\mathrm{HW}(z)-1}{i-1}$ which in a similar way gives

$$\sum_{i=0}^{\lfloor \frac{\mathrm{HW}(z)}{2} \rfloor} \binom{\mathrm{HW}(z)}{i} i = \frac{\mathrm{HW}(z)}{2} 2^{\mathrm{HW}(z)-1} - \frac{\mathrm{HW}(z)}{2}\binom{\mathrm{HW}(z)-1}{\frac{\mathrm{HW}(z)-1}{2}} \times (\mathrm{HW}(z) \mod 2) \ . \tag{6.22}$$

Finally, (6.20), (6.21) and (6.22) lead to (6.18) □

As a consequence of Proposition 6.5, the optimal prediction for the absolute difference combining in the idealized Hamming weight model is not the Hamming weight of Z but a non-affine function of it.

Corollary 6.3. *In the Hamming weight model, the optimal prediction function $\hat{\varphi}_{opt}$ corresponding to $\mathcal{C}_{diff*}$ is of the form:*

$$\hat{\varphi}_{opt} : z \mapsto [A \circ \hat{\varphi}](z) \ ,$$

where $\hat{\varphi}$ is the function $z \mapsto 2^{1-\mathrm{HW}(z)}\mathrm{HW}(z)\binom{\mathrm{HW}(z)-1}{\lfloor \frac{\mathrm{HW}(z)}{2} \rfloor}$ and where A is either the identity function or an affine increasing function defined over $\hat{\varphi}(\mathcal{Z})$.

Proof. This a straightforward consequence of Corollary 6.1 and of Proposition 6.5. □

Our main interest in Corollary 6.3 is that it tells us that even when the leakage satisfies the Hamming weight model, the Hamming weight of the targeted variable is not necessarily the optimal prediction for an HO-DPA. It actually depends on the combining function.

The variance of $|\mathrm{HW}(Z \oplus M) - \mathrm{HW}(M)|$ has already been computed in [133]. The authors prove that it satisfies:

$$\mathrm{Var}\left[|\mathrm{HW}(Z \oplus M) - \mathrm{HW}(M)|\right] = \frac{n}{2} - \left(2^{-2n} n \binom{2n}{n}\right)^2 . \tag{6.23}$$

By Corollary 6.1 and in view of formulae (6.18) and (6.23), we deduce the optimal correlation related to $\mathcal{C}_{diff^\star}$:

$$\rho_{opt}^{diff^\star} = \frac{2^n \sum_{i=0}^{n} 2^{-2i} i^2 \binom{n}{i} \binom{i-1}{\lfloor \frac{i}{2} \rfloor}^2 - \left(\sum_{i=0}^{n} 2^{-i} i \binom{n}{i} \binom{i-1}{\lfloor \frac{i}{2} \rfloor}\right)^2}{2^{2n-2}\left(\frac{n}{2} - \left(2^{-2n} n \binom{2n}{n}\right)^2\right)} .$$

We have computed in Table 6.2 the optimal correlation $\rho_{opt}^{diff^\star}$ for some values of n. For comparison, we have also computed the correlation ρ_{HW} that corresponds to the Hamming weight prediction function (*i.e.* $\hat{\varphi} : z \mapsto \mathrm{HW}(z)$). As expected, choosing our new prediction function makes it possible to slightly increase the correlation value (especially for low values of n). Furthermore, it can be checked that, as stated in Proposition 6.1, the efficiency gain is $\rho(\hat{\varphi}, \hat{\varphi}_{opt})$.

Table 6.2: Correlations for the absolute difference combining in the idealized model.

n	1	2	3	4	5	6	7	8
HW	1.00	0.53	0.41	0.35	0.31	0.28	0.26	0.24
$\hat{\varphi}_{opt}$	1.00	0.65	0.50	0.41	0.35	0.31	0.28	0.26

When the leakage is noisy, the previous analysis is no longer valid and cannot be extended to take the noise into account. Therefore, in the next section, we conduct a complementary analysis which addresses the noisy model.

6.4.2.2 Study in the noisy model

In the analysis that follows, we shall use the notation erf to denote the *error function* defined for every $x \in \mathbb{R}$ by $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\ dt$. We recall that the cumulative distribution function Φ of the standard Gaussian distribution $\mathcal{N}(0,1)$ and the error function satisfy $\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(x/\sqrt{2}\right)\right)$. The following proposition shall be useful to study $\mathcal{C}_{diff^\star}$ when the leakage is noisy.

Proposition 6.6. *Let s be a real number and let B be a Gaussian random variable centered in zero with a standard deviation σ_0. The expectation of the variable $|s + B|$ satisfies:*

$$\mathrm{E}\left[|s + B|\right] = s\ \mathrm{erf}\left(\frac{s}{\sqrt{2}\sigma_0}\right) + \frac{\sqrt{2}\sigma_0}{\sqrt{\pi}} \exp\left(-\frac{s^2}{2\sigma_0^2}\right) . \tag{6.24}$$

Proof. Let ϕ_B and Φ_B respectively denote the probability density function and the probability distribution function of B (that is $\Phi_B(y) = \mathrm{P}\left[B \leqslant y\right] = \int_{-\infty}^{y} \phi_B(x)\ dx$). As B has a Gaussian distribution $\mathcal{N}(0, \sigma_0)$, we have $\phi_B(x) = \frac{1}{\sqrt{2\pi}\sigma_0} \exp(-x^2/2\sigma_0^2)$. Then we have:

$$\mathrm{E}\left[|s + B|\right] \ = \ \int_{-\infty}^{+\infty} |s + x|\ \phi_B(x)\ dx \ = \ s\int_{-s}^{s} \phi_B(x)\ dx \ + \int_{-s}^{s} x\phi_B(x)\ dx \ + 2\int_{s}^{+\infty} x\phi_B(x)\ dx \ .$$

Since the function $x \mapsto x\phi_B(x)$ is odd, the term $\int_{-s}^{s} x\phi_B(x)\ dx$ equals zero. Moreover, we have $\int_{-s}^{s} \phi_B(x)\ dx = 2\left(\Phi_B(s) - \frac{1}{2}\right)$ and $\int_{s}^{+\infty} x\phi_B(x)\ dx = \frac{\sigma_0}{\sqrt{2\pi}} \exp\left(-s^2/2\sigma_0^2\right)$. Hence, we get:

$$\mathrm{E}\left[|s + B|\right] = 2s\left(\Phi_B(s) - \frac{1}{2}\right) + \frac{\sqrt{2}\sigma_0}{\sqrt{\pi}} \exp\left(-s^2/2\sigma_0^2\right) . \tag{6.25}$$

Finally, since B has a Gaussian distribution $\mathcal{N}(0, \sigma_0)$, its probability distribution function Φ_B satisfies $\Phi_B(y) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{y}{\sqrt{2}\sigma_0}\right)\right)$ for every $y \in \mathbb{R}$ hence (6.25) directly implies (6.24). □

As a straightforward consequence of Proposition 6.6, we have the following corollary.

Corollary 6.4. *Let L_1 and L_2 satisfy* (6.8) *and* (6.9). *For every $z \in \mathbb{F}_2^n$, we have:*

$$\mathrm{E}\left[\mathcal{C}_{diff^\star}(L_1, L_2) \mid Z = z\right] = \mathrm{E}\left[(\mathrm{HW}(z \oplus M) - \mathrm{HW}(M))\ \mathrm{erf}\left(\frac{\mathrm{HW}(z \oplus M) - \mathrm{HW}(M)}{2\sigma}\right)\right] + \frac{2\sigma}{\sqrt{\pi}}\mathrm{E}\left[\exp\left(-\frac{(\mathrm{HW}(z \oplus M) - \mathrm{HW}(M))^2}{4\sigma^2}\right)\right], \quad (6.26)$$

and

$$\mathrm{Var}\left[\mathcal{C}_{diff^\star}(L_1, L_2)\right] = 2\sigma^2 + \frac{n}{2} - \mathrm{E}\left[\mathcal{C}_{diff^\star}(L_1, L_2)\right]^2 . \quad (6.27)$$

Proof. After denoting $S = \mathrm{HW}(z \oplus M) - \mathrm{HW}(M)$, we get $\mathrm{E}\left[|L_1 - L_2| \mid Z = z\right] = \mathrm{E}\left[|S + B|\right]$ and Proposition 6.6 directly leads to (6.26). Since we have $\mathrm{E}[B] = 0$, then $\mathrm{E}\left[|S + B|^2\right]$ equals $\mathrm{E}\left[S^2\right] + \mathrm{E}\left[B^2\right]$. Due to the linearity of the expectation, $\mathrm{E}\left[S^2\right]$ equals $\mathrm{E}\left[\mathrm{HW}(M)^2\right] + \mathrm{E}\left[\mathrm{HW}(z \oplus M)^2\right] - 2\mathrm{E}\left[\mathrm{HW}(M)\mathrm{HW}(z \oplus M)\right]$. Then, from Lemma 6.2, we deduce $\mathrm{E}\left[S^2\right] = \mathrm{HW}(z)$. On the other hand we have $\mathrm{E}\left[B^2\right] = 2\sigma^2$, hence we deduce $\mathrm{E}\left[|S + B|^2\right] = 2\sigma^2 + \mathrm{HW}(z)$ which finally gives (6.27) by definition of the variance. □

Corollary 6.4 does not allow to exhibit explicit formulae for $\hat{\varphi}_{opt}$ and ρ_{opt} in the noisy model. However, (6.26) and (6.27) may be involved in efficiently computing the optimal prediction function and the optimal correlation corresponding to $\mathcal{C}_{diff^\star}$ in the noisy model for every pair (n, σ). As an illustration we give in Table 6.3 the exact optimal correlation $\rho_{opt}^{diff^\star}$ for $n \in \{1, \ldots, 8\}$ and $\sigma \in \{0, 1, 5, 10\}$.

Table 6.3: Optimal correlation for the absolute difference combining.

$\sigma \backslash n$	1	2	3	4	5	6	7	8
0	1.00	0.655	0.495	0.405	0.348	0.308	0.280	0.258
1	0.143	0.166	0.173	0.173	0.171	0.168	0.164	0.161
5	0.007	0.009	0.011	0.013	0.014	0.015	0.016	0.017
10	0.002	0.002	0.003	0.003	0.004	0.004	0.004	0.005

In order to determine the efficiency loss resulting from the use of the Hamming weight as prediction function instead of the one defined in (6.26), we computed the correlation $\rho\left[\mathrm{HW}(Z), \hat{\varphi}_{opt}(Z)\right]$ (as suggested in Proposition 6.1) for different values of n and σ. Table 6.4 lists some of our results.

Table 6.4: Correlation between the optimal prediction function and the Hamming weight.

$\sigma \backslash n$	1	2	3	4	5	6	7	8
0	1	0.816	0.832	0.861	0.886	0.905	0.919	0.930
1	1	0.996	0.996	0.996	0.996	0.996	0.996	0.996
5	1	0.998	0.997	0.999	0.999	0.999	0.999	0.999
10	1	1	1	1	0.999	0.999	0.999	0.999

Table 6.4 suggests that whatever the dimension n, the correlation $\rho(\mathrm{HW}(Z), \hat{\varphi}_{opt}(Z))$ tends toward 1 when σ increases. This suggests that in the noisy model, the Hamming weight of Z (or an affine function of it) is a good prediction for the absolute difference combined leakage and that it becomes optimal as the noise increases. The following corollary brings an explanation to this phenomenon.

Corollary 6.5. *Let L_1 and L_2 satisfy* (6.8) *and* (6.9). *Then for every integer n and for every $z \in \mathbb{F}_2^n$, we have:*

$$\mathrm{E}\left[\mathcal{C}_{diff^\star}\left(L_1, L_2\right) \mid Z = z\right] = \frac{2\sigma}{\sqrt{\pi}} + \frac{\mathrm{HW}(z)}{2\sqrt{\pi}\sigma} + \varepsilon\left(\frac{1}{\sigma^3}\right) ,$$

and

$$\mathrm{Var}\left[\mathcal{C}_{diff^\star}\left(L_1, L_2\right)\right] = \frac{2\pi - 4}{\pi}\sigma^2 + \frac{\pi - 2}{2\pi}n + \varepsilon\left(\frac{1}{\sigma^2}\right) .$$

Proof. Let us focus on (6.26) asymptotically. For every a, we have $\mathrm{erf}(a) = \frac{2}{\sqrt{\pi}}a + \varepsilon\left(a^3\right)$ and $\exp(a) = 1 + a + \varepsilon\left(a^2\right)$. Since we also have $\mathrm{HW}(z \oplus M) - \mathrm{HW}(M) = \varepsilon(1)$ (as n is a constant), we can rewrite (6.26) in the following form:

$$\begin{aligned}\mathrm{E}\left[\mathcal{C}_{diff^\star}\left(L_1, L_2\right) | Z = z\right] = \frac{1}{\sqrt{\pi}\sigma}\mathrm{E}\left[\left(\mathrm{HW}(z \oplus M) - \mathrm{HW}(M)\right)^2\right] + \varepsilon\left(\frac{1}{\sigma^3}\right) \\ + \frac{2\sigma}{\sqrt{\pi}}\left(1 - \frac{1}{4\sigma^2}\mathrm{E}\left[\left(\mathrm{HW}(z \oplus M) - \mathrm{HW}(M)\right)^2\right] + \varepsilon\left(\frac{1}{\sigma^4}\right)\right) . \quad (6.28)\end{aligned}$$

Then, $\mathrm{E}\left[\left(\mathrm{HW}(z \oplus M) - \mathrm{HW}(M)\right)^2\right]$ equals $\mathrm{E}\left[\mathrm{HW}(M)^2\right] + \mathrm{E}\left[\mathrm{HW}(z \oplus M)^2\right] - 2\mathrm{E}\left[\mathrm{HW}(M)\mathrm{HW}(z \oplus M)\right]$. From Lemma 6.2 and since $\mathrm{E}\left[\mathrm{HW}(M)^2\right] = \mathrm{E}\left[\mathrm{HW}(z \oplus M)^2\right] = \frac{n^2+n}{4}$, one verifies that this expression equals $\mathrm{HW}(z)$ which together with (6.28) and (6.27) implies Corollary 6.5. □

Corollary 6.5 confirms the empirical study presented in Table 6.4: in the noisy model, the Hamming weight is a good prediction for the absolute difference combined leakage. Indeed, the function $z \mapsto \mathrm{E}\left[\,|L_1 - L_2| \mid Z = z\right]$ (which corresponds to the optimal prediction function) tends toward an affine function of $\mathrm{HW}(z)$ when the noise increases. Moreover, we can deduce from Corollary 6.1 and Corollary 6.5 an approximation of the correlation $\rho_{opt}^{diff^\star}$ when n is negligible compared to σ:

$$\rho_{opt}^{diff^\star} \approx \frac{\sqrt{n}}{4\sqrt{2\pi - 4}\sigma^2} .$$

6.4.3 Product vs. absolute difference

In the two previous sections, we have investigated the correlation of second-order DPA involving either the product or the absolute difference as combining function. Tables 6.1 and 6.3 give the correlations for $n \in \{0, \ldots, 8\}$ and $\sigma \in \{0, 1, 5, 10\}$ and show that, for all these parameters, the correlation for the product combining is greater than the correlation for the absolute difference combining.

In a *very* noisy model ($\sigma \gg n$), we have shown that the correlations satisfy:

$$\rho_{opt}^{prod^\star} \approx \frac{\sqrt{n}}{4\sigma^2} = 0.25\frac{\sqrt{n}}{\sigma^2} ,$$

and

$$\rho_{opt}^{diff^\star} \approx \frac{\sqrt{n}}{4\sqrt{2\pi - 4}\sigma^2} \approx 0.165\frac{\sqrt{n}}{\sigma^2} .$$

We observe a linear relationship between the two approximations of the correlations in the *very* noisy model: $\rho_{opt}^{prod^\star} \approx 1.5\rho_{opt}^{diff^\star}$. As a straightforward consequence of this relation, the correlation $\rho_{opt}^{prod^\star}$ is always greater than $\rho_{opt}^{diff^\star}$ when the noise is high and the two correlations are asymptotically equivalent when the noise increases.

6.4.3.1 Empirical verification

In order to empirically verify the analysis carryied out in the previous sections, we ran some second-order DPA attack simulations according to the defined Hamming weight model. The targeted sensitive variable Z was a vector of $n \leqslant 8$ bits chosen among the output bits of the AES S-Box (taking $X \oplus K$ as

input). The different values of X were randomly picked up to model a known (but not chosen) plaintext attack. Tables 6.5 and 6.6 give the number of measurements required to reach a success rate of either 90% or 99.9% for the product and the absolute difference according to the values of $n \in \{0, \ldots, 8\}$ and $\sigma \in \{0, 1, 5\}$ (10000 – resp. 1000 – simulations were performed for $\sigma \in \{0, 1\}$ – resp. $\sigma = 5$).

Table 6.5: Number of required measurements for the product combining.

n	1	2	3	4	5	6	7	8
$\sigma = 0,\ SR = 90.0\%$	20	30	40	60	80	100	110	130
$\sigma = 0,\ SR = 99.9\%$	30	50	80	130	150	190	230	280
$\sigma = 1,\ SR = 90.0\%$	430	310	280	280	280	290	300	310
$\sigma = 1,\ SR = 99.9\%$	940	690	600	600	570	600	650	700
$\sigma = 5,\ SR = 90.0\%$	190000	100000	65000	55000	40000	35000	30000	25000
$\sigma = 5,\ SR = 99.9\%$	410000	205000	135000	120000	85000	75000	65000	55000

Table 6.6: Number of required measurements for the absolute difference combining.

n	1	2	3	4	5	6	7	8
$\sigma = 0,\ SR = 90.0\%$	20	40	60	90	130	170	210	250
$\sigma = 0,\ SR = 99.9\%$	30	60	130	190	270	340	440	550
$\sigma = 1,\ SR = 90.0\%$	900	800	800	700	700	750	750	800
$\sigma = 1,\ SR = 99.9\%$	1800	1700	1650	1600	1550	1500	1600	1750
$\sigma = 5,\ SR = 90.0\%$	420000	300000	225000	155000	115000	90000	80000	70000
$\sigma = 5,\ SR = 99.9\%$	800000	770000	410000	380000	200000	200000	170000	160000

Remark 6.4. We can observe that the results printed in Tables 6.5 and 6.6 match very well the correlation values given in Tables 6.1 and 6.3. Indeed, there is a kind of one-to-one correspondance between the correlation values and the number of measurements required to reach a given success rate. These results confirm that the correlation is a good indicator of the efficiency of an HO-DPA.

The number of measurements required by an HO-DPA quickly increases as the noise increases. Consequently, we were not able to derive some precise success rates for $\sigma \geqslant 10$. However, we have done several simulations with different noise deviations that all led to the same results: the number of measurements required to retrieve the targeted secret was almost all the time smaller for the product combining than for the absolute difference combining.

From our observations, we conclude that the product combining is more efficient than the absolute difference combining not only in the idealized but also in the noisy model (under the assumption that the leakage is normalized before being combined as explained in Section 6.4.1).

6.4.4 Further combining functions

Other combining functions have been proposed in the literature [133,175,176]. In this section, we discuss these different proposals.

Raising to the power. In [133], Marc Joye *et al.* suggest to improve the efficiency of the absolute difference combining by raising it to a power α. They analyze the new combining functions $\mathcal{C}^{\alpha}_{diff^\star}$ in the idealized model (corresponding to our model with $\sigma = 0$) for a single-bit second-order DPA (*i.e.* with a binary combining function $f : z \mapsto z[i]$). Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich carry on with this approach in [176]: for a prediction function equal to the Hamming weight (*i.e.* $f : z \mapsto \mathrm{HW}(z)$), they evaluate the correlation coefficients for $\mathcal{C}^{\alpha}_{diff^\star}$ and $\mathcal{C}^{\alpha}_{prod^\star}$ according to different values of α in the idealized model without offset (corresponding to our model with $\delta_1 = \delta_2 = 0$).

For several values n and α, we have computed in the idealized model the optimal correlations for both $\mathcal{C}^{\alpha}_{prod^\star}$ and $\mathcal{C}^{\alpha}_{diff^\star}$[11]. Table 6.7 lists the obtained values.

Table 6.7: Optimal correlation for $\mathcal{C}^{\alpha}_{prod^\star}$ and $\mathcal{C}^{\alpha}_{diff^\star}$.

$\alpha \backslash n$	1	2	3	4	5	6	7	8
Product								
1	1.00	0.71	0.58	0.50	0.45	0.41	0.38	0.35
2	und.	0.58	0.37	0.27	0.21	0.17	0.15	0.13
3	1.00	0.71	0.50	0.39	0.33	0.29	0.26	0.24
4	und.	0.58	0.44	0.32	0.24	0.19	0.16	0.14
5	1.00	0.71	0.50	0.36	0.26	0.21	0.17	0.15
6	und.	0.58	0.45	0.33	0.24	0.18	0.14	0.11
Absolute difference								
1	1.00	0.65	0.50	0.41	0.35	0.31	0.28	0.26
2	1.00	0.58	0.45	0.38	0.33	0.30	0.28	0.26
3	1.00	0.60	0.45	0.37	0.33	0.29	0.27	0.25
4	1.00	0.62	0.45	0.36	0.31	0.28	0.25	0.24
5	1.00	0.64	0.45	0.35	0.30	0.26	0.24	0.22
6	1.00	0.65	0.45	0.35	0.29	0.25	0.22	0.20

For both combining functions and for every n, the maximum of the optimal correlations is reached for $\alpha = 1$. Thus, our analysis shows that raising the combined leakage to a power is not a sound approach to increase the efficiency of a second-order DPA when the noise is zero. This seems to contradict the analyses presented in [133, 176], where the authors report that raising to some values α improves the efficiency of the combining. The difference between our conclusions and the ones in [133, 176] is a consequence of the following fact: our study compares second-order DPA that have been optimized by involving the optimal prediction function (introduced in Section 6.3) and by normalizing the leakage signals (as shown in Section 6.4.1). Besides, for every α we have tested, our correlation values are greater than the ones reported by Elisabeth Oswald *et al.* in [176].

In fact we observed that raising to the power also decreases the efficiency of second-order DPA in the noisy model. To summarize, our analysis suggests that raising the combining function to a power α decreases the efficiency of the second-order DPA, the noise being zero or not.

Sine-based combining function. In [175], Elisabeth Oswald and Stefan Mangard propose a combining function based on the sine function. It takes as parameters the exact Hamming weights of the mask and of the masked variable[12]:

$$\mathcal{C}_{sin}(\mathrm{HW}(Z \oplus M), \mathrm{HW}(M)) = \sin\left((\mathrm{HW}(Z \oplus M) - \mathrm{HW}(M))^2\right) \ . \tag{6.29}$$

They also suggest to use the above combining function together with the following prediction function:

$$\hat{\varphi}_{sin}(Z) = -89.95 \sin(\mathrm{HW}(Z))^3 - 7.82 \sin(\mathrm{HW}(Z))^2 + 67.66 \sin(\mathrm{HW}(Z)) \ . \tag{6.30}$$

In the idealized model and for $n = 8$, the use of the couple $(\mathcal{C}_{sin}, \hat{\varphi}_{sin})$ allows an attacker to reach a correlation of 0.83 which is quite high. However, $\hat{\varphi}_{sin}$ is not optimal. Indeed, Corollary 6.1 states that the optimal prediction function for $\mathcal{C}_{sin}$ is the function $\hat{\varphi}_{opt}$ defined by:

$$\hat{\varphi}_{opt}(z) = \mathrm{E}\left[\mathcal{C}_{sin}(\mathrm{HW}(z \oplus M), \mathrm{HW}(M))\right] \ . \tag{6.31}$$

[11] When n equals 1 and α is even, the product of the leakages does not depend on Z (and the expectation is constant with Z) which results in an undefined correlation.

[12] The formulae given in [175] are erroneous and (6.29) and (6.30) are their corrected versions.

Actually, for such a function we have $\rho(\hat{\varphi}_{sin}, \hat{\varphi}_{opt}) = 0,97$, which implies that the use of $\hat{\varphi}_{sin}$ instead of $\hat{\varphi}_{opt}$ results in an efficiency loss of 3%.

With or without the above improvement, it is difficult to compare the efficiencies of $\mathcal{C}_{sin}$ and $\mathcal{C}_{prod^\star}$. Indeed, the attack scenario presented in [175] does not correspond to the kind of attacker we focus here. In [175] the authors consider a very strong adversarial model where the attacker is able to recover the exact Hamming weights of the mask and of the masked variable based on pre-processed leakage profiles (see Section 3.5.3 for further details on profiled attacks). However, in such a scenario, combining the obtained Hamming weights is a suboptimal attack strategy and, as explained in [175], a better strategy is to use a Bayesian classification (or maximum likelihood test). Moreover the recovering of the exact Hamming weight values is only possible in an almost noise-free model.

As argued at the beginning of this section, in a classical HO-DPA scenario, the evaluation process of a combining function must include an analysis in a noisy environment. Therefore, we analyzed the efficiency of the sine-based combining in the presence of noise. Namely, we added Gaussian noises $B_1, B_2 \sim \mathcal{N}(0, \sigma)$ to the Hamming weights in (6.29) and (6.31). We list in Table 6.8 the values of the correlation according to an increasing noise (with n equal to 8).

Table 6.8: Correlations for $\mathcal{C}_{sin}$ and $\mathcal{C}_{prod^\star}$ according to σ.

$(\mathcal{C}, \hat{\varphi}) \setminus \sigma$	0	0.1	0.3	0.4	0.5	0.7	1	5
$(\mathcal{C}_{sin}, \hat{\varphi}_{opt})$	0.87	0.74	0.38	0.21	0.11	0.05	0.037	0
$(\mathcal{C}_{sin}, \hat{\varphi}_{sin})$	0.83	0.70	0.35	0.19	0.08	0.01	0	0
$(\mathcal{C}_{prod^\star}, \mathrm{HW})$	0.36	0.36	0.34	0.33	0.32	0.29	0.24	0.03

It can be observed that the correlation for $\mathcal{C}_{sin}$ quickly decreases as σ increases. For a noise deviation σ greater or equal to 0.4 (which is quite low) the product combining offers a greater correlation. This suggests that in an HO-DPA scenario (where the leakage is noisy), the sine-based combining function is not suitable.

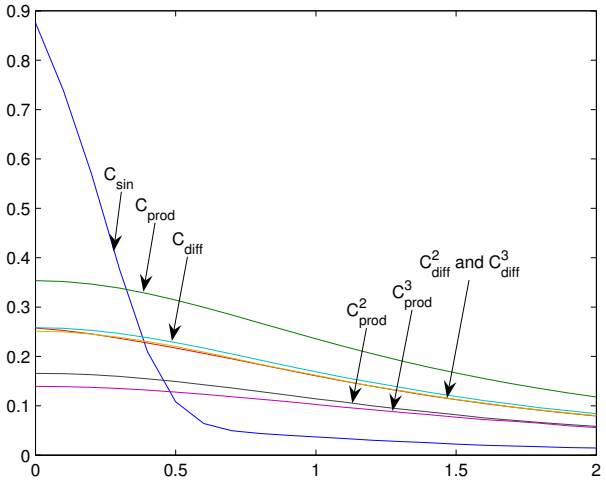


Figure 6.3: Attack correlation for different combining functions according to the noise deviation σ.

Final Comparison. To conclude this section, Figure 6.3 plots the attack correlations with respect to the noise deviation $\sigma \in [0, 2]$ for the combining functions $\mathcal{C}_{sin}$, $\mathcal{C}^{\alpha}_{prod^\star}$ and $\mathcal{C}^{\alpha}_{diff^\star}$, $\alpha \in \{1, 2, 3\}$. This

plot underlines the previous conclusion: among the known combining functions, the improved product combining offers the best efficiency in a general leakage model.

6.5 Analysis of higher-order normalized product combining

When d^{th}-order Boolean masking is used, any sensitive variable Z is split into $d+1$ shares $Z \oplus \mathbf{M}$, $M_1, \ldots, M_d$, where $\mathbf{M}$ denotes the sum $\bigoplus_i M_i$. In the following, we shall denote $Z \oplus \mathbf{M}$ by M_0. The processing of each share M_i results in a leakage signal L_i. In the Hamming weight model, the leakage L_i resulting from the manipulation of the share M_i is modeled as:

$$L_i = \delta_i + \beta_i \cdot \mathrm{HW}(M_i) + B_i \ , \tag{6.32}$$

where δ_i denotes a constant offset, β_i is a real value, $\mathrm{HW}(\cdot)$ denotes the Hamming weight function and B_i denotes a noise with mean 0 and standard deviation σ.

When several leakage signals L_i's are jointly considered, we shall make three additional assumptions: (1) the constant β_i is the same for the different L_i's (without loss of generality, we consider $\beta_i = 1$), (2) noises B_i's are mutually independent and (3) the noise standard deviation is the same for the different B_i's and is denoted σ.

6.5.1 General case

As explained in Section 3.7.1, higher-order DPA consists in combining the $d+1$ leakage signals by the mean of a combining function $\mathcal{C}$. This makes it possible to construct a signal that is correlated to the sensitive variable Z. In the previous section of this chapter we have shown that the best known combining function to perform a second-order DPA in a noisy context is the normalized product combining. We consider here the normalized product combining generalized to higher orders:

$$\mathcal{C}_{prod^\star}(L_0, L_1, \ldots, L_d) = \prod_{i=0}^{d} (L_i - \mathrm{E}[L_i]) \ . \tag{6.33}$$

We shall denote by $C_d(Z)$ the combined leakage signal $\mathcal{C}(L_0, L_1, \ldots, L_d)$ where the L_i's correspond to the processing of the shares $Z \oplus \mathbf{M}$, $M_1, \ldots, M_d$ in the Hamming weight model. The following proposition gives the expectation of $C_d(Z)$ given $Z = z$ for every $z \in \mathbb{F}_2^n$.

Proposition 6.7. *Let $z \in \mathbb{F}_2^n$, then the expectation of $C_d(z)$ satisfies:*

$$\mathrm{E}[C_d(z)] = \left(-\frac{1}{2}\right)^d \left(\mathrm{HW}(z) - \frac{n}{2}\right) \ . \tag{6.34}$$

The proof of Proposition 6.7 makes use of the following lemma.

Lemma 6.3. *We recall that the notation $x[j]$ stands for the j^{th} bit of a value $x \in \mathbb{F}_2^n$. Let $(M_i)_{1 \leqslant i \leqslant d}$ be d random variables uniformly distributed over $\mathbb{F}_2^n$ and mutually independent and let $\mathbf{M} = \bigoplus_i M_i$. For every $z \in \mathbb{F}_2^n$, the expectation of the product $\mathrm{HW}(z \oplus \mathbf{M}) \prod_i \mathrm{HW}(M_i)$ satisfies:*

$$\mathrm{E}\left[\mathrm{HW}(z \oplus \mathbf{M}) \prod_{i=1}^{d} \mathrm{HW}(M_i)\right] = \left(-\frac{1}{2}\right)^d \left(\mathrm{HW}(z) - \frac{n}{2}\right) + \left(\frac{n}{2}\right)^{d+1} \ . \tag{6.35}$$

Proof. According to Property 2.2 (see Section 2.5), we have:

$$\begin{aligned}\mathrm{E}\left[\mathrm{HW}(z \oplus \mathbf{M}) \prod_{i=1}^{d} \mathrm{HW}(M_i)\right] &= \mathrm{HW}(z)\ \mathrm{E}\left[\prod_{i=1}^{d} \mathrm{HW}(M_i)\right] + \mathrm{E}\left[\mathrm{HW}(\mathbf{M}) \prod_{i=1}^{d} \mathrm{HW}(M_i)\right] \\ &\quad - 2\ \mathrm{E}\left[\mathrm{HW}(z \wedge \mathbf{M}) \prod_{i=1}^{d} \mathrm{HW}(M_i)\right] \ . \end{aligned} \tag{6.36}$$

Since the M_i's are uniformly distributed and mutually independent, we have:

$$\mathrm{E}\left[\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] = \left(\frac{n}{2}\right)^d \ , \tag{6.37}$$

and

$$\mathrm{E}\left[\mathrm{HW}(z\wedge\mathbf{M})\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] = \frac{\mathrm{HW}(z)}{n}\ \mathrm{E}\left[\mathrm{HW}(\mathbf{M})\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] \ . \tag{6.38}$$

Relations (6.37) and (6.38) imply that (6.36) can be rewritten as:

$$\mathrm{E}\left[\mathrm{HW}\left(z\oplus\mathbf{M}\right)\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] = \mathrm{HW}(z)\left(\frac{n}{2}\right)^d + \left(1-2\frac{\mathrm{HW}(z)}{n}\right)\mathrm{E}\left[\mathrm{HW}(\mathbf{M})\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] \ . \tag{6.39}$$

The uniformity and mutual independence of the M_i's further imply:

$$\mathrm{E}\left[\mathrm{HW}(\mathbf{M})\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] = n\ \mathrm{E}\left[\mathbf{M}[1]\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] \ ,$$

which can be rewritten as:

$$\mathrm{E}\left[\mathrm{HW}(\mathbf{M})\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] = n\ \mathrm{E}\left[\mathbf{M}[1]M_1[1]\prod_{i=2}^{d}\mathrm{HW}(M_i)\right] + n\ \mathrm{E}\left[\mathbf{M}[1]\left(\sum_{j=2}^{n}M_1[j]\right)\prod_{i=2}^{d}\mathrm{HW}(M_i)\right] \ ,$$

and by induction on d:

$$\begin{aligned}\mathrm{E}\left[\mathrm{HW}(\mathbf{M})\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] &= n\ \mathrm{E}\left[\mathbf{M}[1]\prod_{i=1}^{d}M_i[1]\right] \\ &\quad + n\sum_{k=1}^{d}\ \mathrm{E}\left[\left(\mathbf{M}[1]\prod_{i=1}^{k-1}M_i[1]\right)\left(\sum_{j=2}^{n}M_k[j]\right)\left(\prod_{i=k+1}^{d}\mathrm{HW}(M_i)\right)\right] \ . \end{aligned}\tag{6.40}$$

Then, on the first hand, we have:

$$\mathrm{E}\left[\mathbf{M}[1]\prod_{i}M_i[1]\right] = 2^{-d}(d \bmod 2) \ , \tag{6.41}$$

and on the other hand, the mutual independence between the M_i's implies the mutual independence between $\mathbf{M}[1]$, $(M_i[1])_{1\leqslant i\leqslant k-1}$, $\sum_{j=2}^{n}M_k[j]$ and $(\mathrm{HW}(M_i))_{k+1\leqslant i\leqslant d}$ which leads to:

$$\begin{aligned}&\mathrm{E}\left[\left(\mathbf{M}[1]\prod_{i=1}^{k-1}M_i[1]\right)\left(\sum_{j=2}^{n}M_k[j]\right)\left(\prod_{i=k+1}^{d}\mathrm{HW}(M_i)\right)\right] \\ &\quad = \left(\mathrm{E}\left[\mathbf{M}[1]\right]\prod_{i=1}^{k-1}\mathrm{E}\left[M_i[1]\right]\right)\mathrm{E}\left[\sum_{j=2}^{n}M_k[j]\right]\left(\prod_{i=k+1}^{d}\mathrm{E}\left[\mathrm{HW}(M_i)\right]\right) = \left(\frac{1}{2}\right)^k\frac{n-1}{2}\left(\frac{n}{2}\right)^{d-k}\end{aligned}\tag{6.42}$$

From (6.41) and (6.42), (6.40) can be rewritten as:

$$\begin{aligned}\mathrm{E}\left[\mathrm{HW}(\mathbf{M})\prod_{i=1}^{d}\mathrm{HW}(M_i)\right] &= \frac{n(d \bmod 2)}{2^d} + \frac{n(n-1)}{2}\sum_{k=1}^{d}\left(\frac{1}{2}\right)^k\left(\frac{n}{2}\right)^{d-k} &(6.43)\\ &= \frac{n(d \bmod 2)}{2^d} + \frac{n(n-1)}{2^{d+1}}\sum_{k=1}^{d}n^k &(6.44)\\ &= \frac{n(d \bmod 2)}{2^d} + \frac{n(n^d-1)}{2^{d+1}} &(6.45)\end{aligned}$$

Finally, (6.39) and (6.45) yields (6.35) which conclude the proof. □

We now give the proof of Proposition 6.7.

Proof. (Proposition 6.7) From the expression of the L_i's, we have $\mathrm{E}[L_i] = \delta_i + \frac{n}{2}$ giving:

$$C_d(z) = \left(\mathrm{HW}(z \oplus \mathbf{M}) - \frac{n}{2} + B_0\right) \prod_{i=1}^{d} \left(\mathrm{HW}(M_i) - \frac{n}{2} + B_i\right) \ . \tag{6.46}$$

Since the B_i's have zero means, one deduces:

$$\begin{aligned} \mathrm{E}[C_d(z)] &= \mathrm{E}\left[\left(\mathrm{HW}(\mathbf{M}) - \frac{n}{2}\right) \prod_{i}^{d} \left(\mathrm{HW}(M_i) - \frac{n}{2}\right)\right] \\ &= \mathrm{E}\left[\mathrm{HW}(\mathbf{M}) \prod_{i}^{d} \left(\mathrm{HW}(M_i) - \frac{n}{2}\right)\right] - \frac{n}{2}\ \mathrm{E}\left[\prod_{i}^{d} \left(\mathrm{HW}(M_i) - \frac{n}{2}\right)\right] \end{aligned}$$

The uniformity and the mutual independence between the M_i's imply:

$$\mathrm{E}[C_d(z)] = \mathrm{E}\left[\mathrm{HW}(\mathbf{M}) \prod_{i}^{d} \left(\mathrm{HW}(M_i) - \frac{n}{2}\right)\right]$$

Similarly, we have:

$$\mathrm{E}[C_d(z)] = \mathrm{E}\left[\mathrm{HW}(\mathbf{M})\mathrm{HW}(M_1) \prod_{i>1} \left(\mathrm{HW}(M_i) - \frac{n}{2}\right)\right]$$

and by induction on d:

$$\mathrm{E}[C_d(z)] = \mathrm{E}\left[\mathrm{HW}(\mathbf{M})\mathrm{HW}(M_1) \cdots \mathrm{HW}(M_{d-1}) \left(\mathrm{HW}(M_d) - \frac{n}{2}\right)\right] \ . \tag{6.47}$$

Finally, the uniformity and the mutual independence between the M_i's lead to:

$$\mathrm{E}[C_d(z)] = \mathrm{E}\left[\mathrm{HW}(\mathbf{M}) \prod_{i}^{d} \mathrm{HW}(M_i)\right] - \left(\frac{n}{2}\right)^{d+1} \ , \tag{6.48}$$

which together with Lemma 6.3 imply (6.34). □

Proposition 6.7 shows that the expectation of $C_d(z)$ is an affine function of the Hamming weight of z. According to the analysis in Section 6.3, $\mathrm{HW}(Z)$ can therefore be considered as an optimal prediction for $C_d(Z)$. Hence, the higher-order DPA we focus here consists in estimating the correlation between the Hamming weight of the target variable $\mathrm{HW}(Z)$ and the combined leakage $C_d(Z)$. The next proposition provides the exact value of this correlation.

Proposition 6.8. *Let Z be a random variable uniformly distributed over $\mathbb{F}_2^n$. The correlation between $\mathrm{HW}(Z)$ and $C_d(Z)$ satisfies:*

$$\rho[\mathrm{HW}(Z), C_d(Z)] = (-1)^d \frac{\sqrt{n}}{\left(n + 4\sigma^2\right)^{\frac{d+1}{2}}} \ . \tag{6.49}$$

Proof. According to Lemma 6.1, the covariance between $\mathrm{HW}(Z)$ and $C_d(Z)$ satisfies:

$$\mathrm{Cov}[\mathrm{HW}(Z), C_d(Z)] = \mathrm{Cov}[\mathrm{HW}(Z), \mathrm{E}[C_d(Z)|Z]] \ .$$

By Proposition 6.7, we get:

$$\mathrm{Cov}[\mathrm{HW}(Z), C_d(Z)] = \left(-\frac{1}{2}\right)^d \mathrm{Var}[\mathrm{HW}(Z)] \ ,$$

which leads to:

$$\rho\left[\mathrm{HW}(Z), C_d(Z)\right] = \left(-\frac{1}{2}\right)^d \frac{\sigma\left[\mathrm{HW}(Z)\right]}{\sigma\left[C_d(Z)\right]} = \left(-\frac{1}{2}\right)^d \frac{\sqrt{n}}{2\,\sigma\left[C_d(Z)\right]} \ . \tag{6.50}$$

Since Z and the M_i's are uniformly distributed and mutually independent, then so do $Z \oplus \mathbf{M}$ and the M_i's. Moreover since the B_i's are mutually independent then we get:

$$\begin{aligned} \mathrm{Var}\left[C_d(Z)\right] &= \mathrm{E}\left[\left(\mathrm{HW}(Z \oplus \mathbf{M}) - \frac{n}{2} + B_0\right)^2\right] \prod_{i=1}^{d} \mathrm{E}\left[\left(\mathrm{HW}(M_i) - \frac{n}{2} + B_i\right)^2\right] \\ &= \mathrm{E}\left[\left(\mathrm{HW}(M) - \frac{n}{2} + B\right)^2\right]^{d+1} \ , \end{aligned}$$

where M is a uniform random variable over $\mathbb{F}_2^n$ and B is a random variable with mean 0 and variance σ^2. Since $\mathrm{E}\left[\mathrm{HW}(M)\right] = n/2$ and $\mathrm{E}\left[\mathrm{HW}(M)^2\right] = (n^2+n)/4$, one deduces:

$$\mathrm{E}\left[\left(\mathrm{HW}(M) - \frac{n}{2} + B\right)^2\right] = \mathrm{E}\left[\left(\mathrm{HW}(M) - \frac{n}{2}\right)^2\right] + \mathrm{E}\left[B^2\right] = \frac{n}{4} + \sigma^2 \ ,$$

which implies:

$$\mathrm{Var}\left[C_d(Z)\right] = \left(\frac{n}{4} + \sigma^2\right)^{d+1} \ . \tag{6.51}$$

Finally, (6.50) and (6.51) leads to (6.49). □

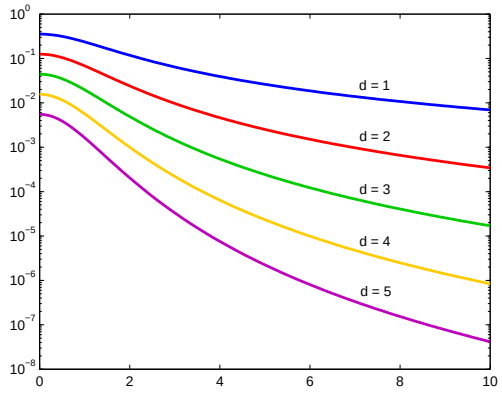


Figure 6.4: Correlation of the higher-order DPA (log-scale) for several orders d and for $n = 8$ according to the noise standard deviation σ (from 0 to 10).

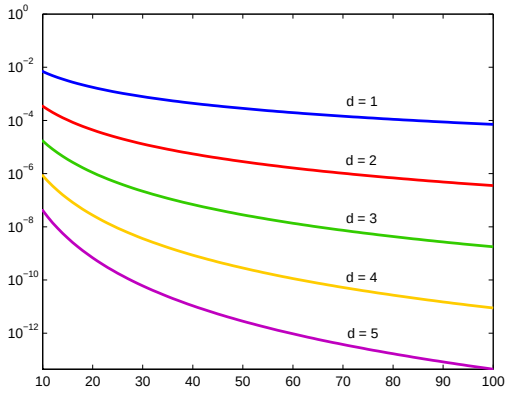


Figure 6.5: Correlation of the higher-order DPA (log-scale) for several orders d and for $n = 8$ according to the noise standard deviation σ (from 10 to 100).

Proposition 6.8 makes clearly appear the security provided by higher-order masking. Indeed, we see that the correlation of the higher-order DPA decreases exponentially with d. As an illustration, Figures 6.4 and 6.5 show the correlation values obtained for $n = 8$ and for different values of d according to an increasing noise.

As argued in [156, 157], a division of the attack correlation by a factor λ implies a multiplication of the number of leakage measurements by a factor λ^2 (for a given success rate). We can then deduce from Proposition 6.8 that incrementing by 1 the masking order results in a multiplication by $(n + 4\sigma^2)$ of the number of required measurements for the higher-order DPA based on the normalized product combining. The stronger is the noise, the more prohibitive is this factor for the attack. This illustrates the soundness of combining higher-order masking with noise addition to thwart DPA attacks.

When the leakage noise is weak, higher-order masking may not be enough to thwart DPA. In that case, one may combine higher-order masking with operations shuffling to improve the DPA resistance. In the next section, we address such a scenario and we describe and analyze some advanced DPA techniques in this context.

6.5.2 Combined higher-order and integrated DPA

In this section, we focus on attacks targeting an implementation protected by combining (higher-order) masking and shuffling. When shuffling is used without masking, the signal containing information about the sensitive variable Z is randomly spread over t different signals $L_1, \ldots, L_t$. As a result, the correlation between the prediction and one of these signals is reduced by a factor t compared to the correlation without shuffling. In [74], an *integrated DPA attack* (also called *windowing attack*) is proposed for this issue. The principle is to add the t signals all together to obtain an *integrated signal*. The correlation is then computed between the prediction and the integrated signal. This way, the resulting correlation is reduced by a factor $\sqrt{t}$ instead of t without integration.

When masking is combined with shuffling, a sensitive variable Z is split into $d+1$ shares $Z \oplus \mathbf{M}$, M_1, $\ldots$, M_d whose manipulations are randomly spread over t different times yielding t different signals L_i. The $(d+1)$-tuple of signals indices corresponding to the shares hence ranges over a subset I of the set of $(d+1)$-combinations from $\{1, \ldots, t\}$. This subset depends on how the shuffling is performed (*e.g.* the shares may be independently shuffled or shuffled all together).

To bypass such a countermeasure, an adversary may combine integrated and higher-order DPA techniques. The most relevant way to perform such a combined attack is to design a so-called *combined-and-integrated* signal by summing all the possible combinations of $d+1$ signals among $L_1, \ldots, L_t$ [228, 229]. That is, the combined-and-integrated signal, denoted $IC_{d,I}(Z)$, is defined by:

$$IC_{d,I}(Z) = \sum_{\mathbf{i} \in I} \mathcal{C}(L_{i_0}, \ldots, L_{i_d}) \ , \tag{6.52}$$

where $\mathbf{i}$ denotes the vector $(i_0, \ldots, i_d)$.

By construction of I, the family of signals $(L_i)_i$ corresponds to a family of processed data $(D_i)_i$ such that there always exists a single $(d+1)$-tuple $(i'_0, \ldots, i'_d) \in I$ for which we have $(D_{i'_0}, D_{i'_1}, \ldots, D_{i'_d}) = (Z \oplus \mathbf{M}, M_1, \ldots, M_d)$. Let us now view $(i'_0, \ldots, i'_d)$ as a random vector uniformly distributed over I and let us assume that the random variables D_j with $j \neq i'_0, \ldots, i'_d$ are uniformly distributed and mutually independent. Then, we have the following proposition:

Proposition 6.9. *Let Z be a random variable uniformly distributed over $\mathbb{F}_2^n$. The correlation between $\mathrm{HW}(Z)$ and $IC_{d,I}(Z)$ satisfies:*

$$\rho\left[\mathrm{HW}(Z), IC_{d,I}(Z)\right] = \frac{1}{\sqrt{\#I}} \rho\left[\mathrm{HW}(Z), C_d(Z)\right] \ .$$

Proof. According to (6.52) the variance of $IC_{d,I}(Z)$ satisfies:

$$\mathrm{Var}\left[IC_{d,I}(Z)\right] = \sum_{(\mathbf{i},\mathbf{j}) \in I^2} \mathrm{Cov}\left[\mathcal{C}(L_{i_0}, \ldots, L_{i_d}), \mathcal{C}(L_{j_0}, \ldots, L_{j_d})\right] \ .$$

Since by definition each monomial $\mathcal{C}(L_{i_0}, \ldots, L_{i_d})$ is a product of terms with zero expectation, the covariance between two different monomials equal zero. By construction, the $\#I$ monomials $\mathcal{C}(L_{i_0}, \ldots, L_{i_d})$ have equal variance and we therefore have $\sigma\left[IC_{d,I}(Z)\right] = \sqrt{\#I} \times \sigma\left[\mathcal{C}(L_{i'_0}, \ldots, L_{i'_d})\right]$. Moreover, only the combination $\mathcal{C}(L_{i'_0}, \ldots, L_{i'_d})$ is statistically dependent on Z. We hence deduce that the covariance $\mathrm{Cov}\left[\mathrm{HW}(Z), IC_{d,I}(Z)\right]$ equals $\mathrm{Cov}\left[\mathrm{HW}(Z), \mathcal{C}(L_{i'_0}, \ldots, L_{i'_d})\right]$. Since $\mathcal{C}(L_{i'_0}, \ldots, L_{i'_d})$ and $C_d(Z)$ have, by definition, equal distributions, we deduce that the correlation $\rho\left[\mathrm{HW}(Z), \mathcal{C}(L_{i'_0}, \ldots, L_{i'_d})\right]$ equals $\rho\left[\mathrm{HW}(Z), C_d(Z)\right]$ and Relation (6.9) straightforwardly follows. □

Proposition 6.9 shows that the effect of shuffling is similar with or without masking: the correlation is divided by the square root of the number of possible locations for the meaningful signal(s)[13]. Nevertheless, when shuffling is combined with higher-order masking, the number of possible locations soars up. Basically, we have two possible cases for such a combination:

1. Each share is independently shuffled among t operations. In that case, the number of possible combinations for the $d+1$ meaningful signals is:
$$\#I = t^{d+1} \ .$$
2. The $d+1$ shares are shuffled all together among t operations. Then, the number of possible combinations is:
$$\#I = \binom{t}{d+1} \ .$$

These values, together with Proposition 6.9, show that combining shuffling and higher-order masking is a sound approach to thwart DPA when the noise in the leakage and/or the masking order do not suffice to obtain a satisfactory resistance level. This observation is the motivation for the scheme described in Chapter 11, which combines higher-order masking and shuffling to protect implementations of block ciphers against DPA attacks.

[13]Note that this holds from the normalization of the leakage before product combining which implies a zero covariance between the different monomials in the sum $IC_{d,I}(Z)$ (see the proof of Proposition 6.9). Using a different combining function would make the effect of shuffling stronger (*i.e.* it would decrease the correlation more).

Chapter 7

Conclusions and perspectives

In this first part, we have investigated side channel analysis from a theoretical point of view. We have formalized the concept of side channel attack and the underlying notions (*e.g.* sensitive variable, leakage function, distinguisher) following the outlines of the theoretical model introduced in [216, 217]. We have then reviewed different metrics for the evaluation of side channel attacks. Among these metrics, we took particular interest to the success rate. We investigated how the success rate of an attack may be efficiently and precisely computed. Our approach was to consider the result of an attack (*i.e.* the distinguishing vector) as a random process and to characterize its distribution. We have shown that, under the Gaussian noise assumption, the result of an attack using either the correlation or the likelihood as distinguisher follows a multivariate Gaussian distribution. The parameters of this distribution were exhibited with respect to the leakage parameters and to the attacker's model/estimations. From these distributions, we were able to precisely estimate the success rate of the underlying attack. Although this analysis is purely theoretical, we believe that it has a real practical interest for the rigorous evaluation of embedded devices. In practice, an embedded cryptographic implementation is usually protected by some countermeasures and an attack on such an implementation is (hopefully) complex in terms of leakage measurements. This makes it difficult (or even impossible) to evaluate the success rate of an attack by performing it many times. That is why, security evaluations usually focus on the result of a single attack which is not very satisfactory in terms of security. Our analysis could be used in order to tackle this issue. As a next step, the effectiveness of our approach should be validated in practice. An extension of our analysis to higher-order side channel attacks would then be of particular interest for the evaluation of protected implementations.

Our researches also addressed some attacks techniques in more detail. First, we investigated mutual information analysis, a recently proposed attack technique that makes use of the mutual information as distinguisher [29, 104]. The motivation behind this approach is that, contrary to correlation, mutual information detects every kind of statistical dependences, not only linear ones. This makes it possible to relax some assumptions about the leakage which are necessary to classical DPA attacks. However, some questions had remained unanswered after the first publications. In particular, the estimation of the mutual information, which itself requires the estimation of statistical distributions, is a major practical issue that had not been fully investigated. Secondly, it was not clear to what extent MIA should be preferred or not to classical DPA attacks. To answer these questions, we conducted an in-depth analysis of the theoretical and the practical aspects of MIA. First, we studied MIA under the Gaussian noise assumption which allowed us to stress the theoretical foundations behind the attack and to generalize it to higher orders. Then, we presented some classical estimation methods for statistical distributions and we applied them in the context of MIA. We observed that the way to estimate the mutual information has a clear impact on the attack efficiency. In particular, we introduced a parametric estimation method which significantly improves the MIA efficiency compared to the initially proposed histogram method. Finally, our analysis showed that MIA is less efficient than classical DPA in a common context where the leakage linearly depends on the Hamming weight of the target variable.

We also studied higher-order side channel attacks which target protected implementations. We addressed higher-order DPA attacks and the underlying combining functions. At the time when we started

our study, two different combining functions were commonly used to perform second-order attacks: the absolute difference combining and the product combining. However, no formal analysis existed allowing one to decide between them. To fill this gap, we conducted a study of second-order DPA involving both combining functions under common assumptions about the leakage. Our analysis clearly showed that the product combining should be preferred to the absolute difference combining. Moreover, we have shown that a normalization of the leakage measurements before product combining enables a significant gain in the attack efficiency. The resulting normalized product combining is at this day the best combining function in the literature to perform a second-order DPA under common assumptions about the leakage. We also exhibited the optimal prediction function for higher-order DPA according to a combining function and given a leakage model. This result is not of practical purpose since a higher-order DPA adversary is not assumed to precisely know the leakage distribution (contrary to a profiled attack adversary). On the other hand, it allows one to determinate what a higher-order DPA attacker can ideally expect (for a given combining function) which is of great interest from a security point of view. Finally, we extended our analysis of the normalized product combining to higher orders and possibly in the presence of operation shuffling. From a more general point of view, our analysis introduced the basis for a practically oriented analysis of higher-order DPA attacks that may be used for further investigations. In particular, the proposed framework makes it possible to analyze the efficiency of new combining techniques in a general model. Future research could focus on better/optimal combining functions for higher-order DPA. However, it should be noted that higher-order DPA involving a combining function is not optimal from an information theoretic point of view. Indeed, the application of a combining function inevitably leads to a loss of information. Therefore, other types of higher-order side channel attacks should also be investigated. Regarding this issue, we showed how MIA can be naturally extended to higher orders. However, the first experimental results have showed that second-order MIA is still less efficient than second-order DPA involving the normalized product combining. For future work, a more accurate analysis of MIA at different orders and for different leakage noises would be of interest. In particular, an alternative method for higher-order MIA based on *multivariate mutual information* has been recently proposed in [103]. A comparison of the two approaches is missing at this day. Moreover, as we stressed in our study, the main practical issue of higher-order MIA is the estimation of Gaussian mixture distributions. This issue, which has been investigated in the context of profiled SCA in [152], could also be studied and deepened in the context of higher-order MIA.

Part II

Masking schemes, design and cryptanalysis

Chapter 8

Introduction to masking schemes

Contents

8.1 Introduction

Masking is nowadays the most widely used countermeasure to protect block ciphers implementations against side channel analysis. The basic principle of masking is to add one or several random value(s) to every sensitive variable occurring during the computation. This approach appeared in the literature soon after the publication of the DPA attack [147]. It was independently published by Suresh Chari, Charanjit Jutla, Josyula Rao, and Pankaj Rohatgi in [65] and by Louis Goubin and Jacques Patarin in [118]. In [65], the authors conduct a theoretical study that demonstrates the soundness of masking. On the other hand, [118] introduces a general framework to protect block ciphers using masking and applies it to the DES cipher. The terminologies employed in these two papers are different: in [65], masking is referred to as *sharing*, while in [118], it is referred to as *duplication*. In this thesis, we use the term *masking* as it is today the most widely used in the literature.

We present hereafter the outlines of masking and we explain why it is a sound approach to counteract side channel analysis. Then, we review the different *masking schemes* for block ciphers which have been proposed in the literature. Afterwards, we introduce a model to prove the security of such schemes. Eventually, we present the outlines of our different contributions.

8.2 Masking outlines

When masking is involved, every sensitive variable Z occurring during the computation is randomly split into $d+1$ shares $M_0, \ldots, M_d$ in such a way that the following relation is satisfied for a group operation

$\star$ (*e.g.* the XOR or the modular addition):

$$M_0 \star M_1 \star \cdots \star M_d = Z \tag{8.1}$$

Usually, the d shares $M_1, \ldots, M_d$ (called *the masks*) are randomly picked up and the share M_0 (called *the masked variable*) is processed such that it satisfies (8.1). Two types of masking have mainly been studied: the *Boolean masking* for which $\star$ is defined as the XOR operation and the *arithmetic masking* for which $\star$ is defined as the modular addition. Boolean masking is the most widely used since most ciphers use the XOR as elementary operation. When d random masks are involved per sensitive variable the masking is further said to be *of order d*.

Assuming that the masks are uniformly distributed, masking renders every intermediate variable of the computation statistically independent of any sensitive variable. As a result, classical side channel attacks exploiting the leakage related to a single intermediate variable are not possible anymore. However, masking can be overcome by higher-order side channel attacks that exploit the leakages related to several intermediate variables at the same time (see Section 3.7). Indeed, the leakages resulting from the $d+1$ shares (i.e. the masked variable and the d masks) are jointly dependent on the sensitive variable. Whatever its order d, a masking is always theoretically vulnerable to an attack of order $d+1$. Nevertheless, the noise effects imply that the difficulty in carrying out a higher-order SCA increases exponentially with its order, which makes masking a sound countermeasure.

8.3 Soundness of masking

The soundness of masking was formally demonstrated by Suresh Chari, Charanjit Jutla, Josyula Rao, and Pankaj Rohatgi in [65]. They assume a simplified but still realistic leakage model where a bit b is masked using d random bits $M_1, \ldots, M_d$ *i.e.* the masked bit is defined as $M_0 = b \oplus M_1 \oplus \cdots \oplus M_d$. The leakage of each share is modelled as $L_i = M_i + B_i$ where the noises B_i's are assumed to have Gaussian distributions $\mathcal{N}(\mu, \sigma^2)$ and to be mutually independent. Under this leakage model, they show that the number of samples N required by any adversary to distinguish the distribution $(L_0, \ldots, L_d | b = 0)$ from the distribution $(L_0, \ldots, L_d | b = 1)$ with a probability at least α satisfies:

$$N \geqslant \sigma^{d+\delta} \tag{8.2}$$

where $\delta = 4 \log \alpha / \log \sigma$.

This result encompasses all the possible distinguishers and hence formally states the resistance against every kind of side channel attack. Although the model is simplified, it could probably be extended to more common leakage models such as presented in Section 3.4.3. The point is that if an attacker observes noisy side channel information about $d+1$ shares corresponding to a variable masked with d random masks, the number of samples required to retrieve some information about the unmasked variable is lower bounded by an exponential function of the masking order whose base is related to the noise amplitude. This shows that (higher-order) masking is a sound countermeasure especially when combined with noise. Many works also made this observation for particular side channel distinguishers (see for instance Chapter 6, [204, 219]).

When masking is involved in protecting a block cipher implementation, a problem arises: how should the computation on masked data be performed? From a masked plaintext and a possibly masked key, one must perform the computation while ensuring that masks and masked variables are processed separately. For a d^{th}-order masking, the goal is to ensure that every tuple of d or less intermediate variables is independent of any sensitive variable. Otherwise an attack of order lower than $d+1$ may be possible. Any method to achieve this goal is called a *masking scheme*. Most masking schemes proposed in the literature only deal with first-order masking. That is why, we shall not specify the masking order when it is one but we shall specify it when it is higher.

8.4 Masking schemes in the literature

Following the first publications [65,118], several papers have been published that propose masking schemes for block ciphers[14]. The different proposals are reviewed hereafter.

Most block ciphers are composed of substitution boxes (S-boxes), linear operations and key additions (this is for instance the case of the two encryption standards DES and AES). While designing a masking scheme for such a cipher, the main issue is the masking of the S-boxes. This issue has been widely investigated in the literature. Some solutions have been proposed that are generic, namely they can be applied to any S-box. Other solutions have been proposed that are dedicated to the AES S-box taking advantage of its mathematical structure. Other block ciphers exist that involve an alternative use of Boolean and arithmetic operations. Masking such block ciphers poses a masking conversion problem.

8.4.1 Generic masking schemes

The first generic solution (although described for DES) is the one proposed by Louis Goubin and Jacques Patarin in [118]. This solution has recently been shown to be vulnerable to first-order SCA [96]. A very simple and efficient method known as the *table re-computation* was published by Thomas Messerges in [163]. It consists in computing a look-up table representing a masked version of the S-box. This look-up table is then involved each time the S-box must be computed. In order to avoid the memory overhead induced by this method, alternative solutions have been proposed that perform masked S-boxes computations on-the-fly without RAM allocation. A first method was introduced by Emmanuel Prouff, Christophe Giraud and Sébastien Aumônier in [192] that is based on the Fourier transform. As we show in Chapter 12, this method has a flaw with respect to first-order SCA. We then describe a patched version of the method that is secure with respect to first-order SCA. Furthermore, we propose in Chapter 9 another generic method for on-the-fly masked S-box computation which is more efficient than the Fourier transform based method.

Generic solutions have also been proposed to withstand higher-order SCA. A first solution was proposed by Mehdi-Laurent Akkar and Louis Goubin (and described for DES) [23] which was broken and improved in [21,153]. However, all these solutions are based on a first-order masking which prevents them from perfectly thwarting second-order SCA. A second proposal was made by Kai Schramm and Christof Paar in [204] that generalizes the table re-computation method to higher orders. Their scheme takes an order d as parameter and aims to protect an implementation against d^{th}-order SCA. However, we show in Chapter 13 that their method is broken for $d \geqslant 3$ and can only be used to thwart second-order SCA. In Chapter 10, we further present two alternative second-order masking schemes for S-boxes that provide time-memory tradeoffs to the Schramm and Paar's scheme.

8.4.2 Masking schemes dedicated to AES

Due to its strong mathematical structure, the AES S-box is well suited to dedicated masking schemes. Indeed, the AES S-box is defined as the multiplicative inversion over $GF(256)$ composed with an affine transformation (with respect to the XOR). Masking the AES S-box hence mainly amounts to masking the inversion over $GF(256)$. Regarding this issue, a first solution was proposed by Mehdi-Laurent Akkar and Christophe Giraud in [22]. Their solution is based on the use of a multiplicative masking which propagates well throughout the inversion. However, multiplicative masking has a serious drawback: it does not mask the zero value which enables a first-order SCA attack [114]. Another dedicated solution was proposed by Johannes Blömer, Jorge Guajardo Merchan, and Volker Krummel in [46] that performs the inversion over $GF(256)$ using an exponentiation algorithm with on-the-fly mask correction. In [178], Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen propose a method based on composite field arithmetic. They use the fact the inversion over $GF(256)$ can be broken down into a sequence of operations over $GF(16)$ and $GF(4)$ that are easy to mask. Their solution, which was initially devoted to hardware implementations, has been extended to software implementations in [179]. Furthermore, an improved solution for more compact hardware representation was proposed in [62]. Finally, an algebraic method was proposed by Nicolas Courtois and Louis Goubin in [83] that consists

[14] Masking has also been investigated at the logic gate level (see Section 3.6.2 for an overview).

in embedding the inversion over $GF(256)$ in a group of homographic transformations over the projective space.

8.4.3 Masking conversion problem

Some block ciphers exist that involve both Boolean and arithmetic operations. As suggested by Thomas Messerges in [163], masking such block cipher implies the use of both Boolean masking and arithmetic masking. When a cipher requires the two types of masking, conversion methods must be defined to switch from one to the other without introducing a flaw. Thomas Messerges proposed in [163] such conversion methods. However, these methods imply the computation of some intermediate variables that are non-uniformly masked, thereby implying a flaw with respect to first-order SCA [78]. Subsequently, secure methods to convert Boolean masking into arithmetic masking and *vice versa* were proposed by Louis Goubin in [117]. The proposed method for Boolean-to-arithmetic conversion is quite efficient but the converse is less efficient. In order to fix this imbalance, Jean-Sébastien Coron and Alexei Tchulkine proposed in [82] an improved method for arithmetic-to-Boolean conversion based on pre-computed tables. Their method was further improved and extended in [173].

8.5 Provably secure masking schemes for block ciphers

We present in this section a generic masking scheme for block ciphers that are composed of key additions, S-boxes and linear transformations (*e.g.* the encryption standards DES and AES). We show that the main issue while designing such a masking scheme is the masking of the S-boxes computations. We then introduce a security model that makes it possible to formally prove the security of such a masking scheme.

8.5.1 Block cipher model

A block cipher is a cryptographic algorithm that, from a secret key $\mathbf{k}$, transforms a plaintext block $\mathbf{p}$ into a ciphertext block $\mathbf{c}$ through the repetition of key-dependent permutations, called *round transformations*. Let us denote by p, and call *cipher state*, the temporary value taken by the ciphertext during the algorithm. In practice, the cipher is *iterative*, which means that it applies R times the same round transformation φ to the cipher state. This round transformation is parameterized by a *round key* k that is derived from $\mathbf{k}$ by iterating a key scheduling function α. We shall use the notations p^r and k^r when we need to indicate the round r during which the variables p and k are involved: we have $k^{r+1} = \alpha(k^r, r)$ and $p^{r+1} = \varphi[k^r](p^r)$, with $p^0 = \mathbf{p}, p^R = \mathbf{c}$ and $k^0 = \alpha(\mathbf{k}, 0)$.

We consider block ciphers for which the round transformation φ is composed of different operations: a key addition layer σ, a non-linear layer γ, and a linear layer λ:

$$\varphi[k] = \lambda \circ \gamma \circ \sigma[k].$$

The entire cipher transformation can thus be written:

$$\mathbf{c} = \bigcirc_{r=0}^{R-1} \quad \lambda \circ \gamma \circ \sigma[k^r] \ (\mathbf{p}).$$

We consider that the key scheduling function α is composed of linear and non-linear layers similar to those of the round transformation.

The key addition layer is a simple bitwise addition between the round key and the cipher state and we have $\sigma[k](p) = p \oplus k$. The non-linear layer consists of several non-linear vectorial functions S_j, called *S-boxes*, that operate independently on a limited number of state bits: $\gamma(p) = \big(S_1((p)_1), \ldots, S_N((p)_N)\big)$ where $(p)_j$ denotes the j^{th} part of the state p. For reasons of efficiency, S-boxes are most of the time implemented as look-up tables (LUT). We consider here that the linear layer λ, which mixes the outputs of the S-boxes, is linear with respect to the bitwise addition.

As an illustration, Figure 8.1 represents a typical round transformation with a non-linear layer made of four S-boxes.

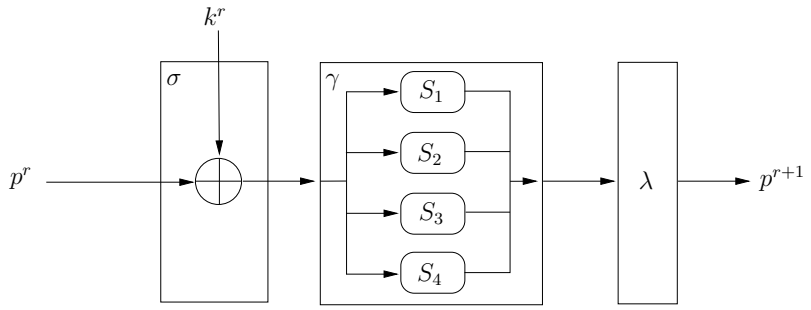


Figure 8.1: A typical round transformation with a non-linear layer composed of four S-boxes.

Remark 8.1. Note that this description is not restrictive regarding the structure of recent block ciphers. In particular, this description can be straightforwardly extended to represent the DES and the AES algorithms.

8.5.2 Generic masking scheme

We describe hereafter a generic masking scheme for any block cipher following the model above. The cipher state p and the secret key k are represented by $d+1$ shares – $(p_0, \ldots, p_d)$ and $(k_0, \ldots, k_d)$ respectively – that satisfy the following relations:

$$p = p_0 \oplus \cdots \oplus p_d \ , \tag{8.3}$$

$$k = k_0 \oplus \cdots \oplus k_d \ . \tag{8.4}$$

At the beginning of the computation, the shares $(p_1, \ldots, p_d)$ and $(k_1, \ldots, k_d)$ – the masks – are randomly generated and the shares p_0 and k_0 – the masked state and the masked key – are processed according to (8.3) and (8.4).

In order to keep track of the correct values of p and k, (8.3) and (8.4) must be satisfied at the different steps of the cipher. At the end of the algorithm, the desired ciphertext (which corresponds to the final value p^R) is re-built from the shares $\left(p_0^R, \ldots, p_d^R\right)$. To preserve the security throughout the cipher, the different shares must always be processed separately. Thus, the point is to process the algorithm by manipulating the shares separately, while maintaining (8.3) and (8.4) in such a way that the unmasked value can always be re-established. To maintain these relations along the algorithm, one must be able to maintain them throughout the three layers λ, σ and γ.

For the linear layer λ, maintaining (8.3) and (8.4) is simply done by applying λ to each share separately. Indeed, due to the linearity of the operations, $\lambda(p)$ and the new shares $\lambda(p_i)$ satisfy the desired relation:

$$\lambda(p) = \lambda(p_0) \oplus \cdots \oplus \lambda(p_d) \ .$$

An easy relation also stands for the key addition layer σ where each k_i can be separately added to each p_i since we have:

$$\sigma[k](p) = \sigma[k_0](p_0) \oplus \cdots \oplus \sigma[k_d](p_d) \ .$$

For the non-linear layer, no such relation exists and maintaining (8.3) is a much more difficult task. This makes the secure implementation of such a layer the principal issue while protecting block ciphers. Because of the non-linearity of γ, new random masks $p'_1, \ldots, p'_d$ must be generated. Then the masked output state p'_0 has to be processed from $(p_0, \ldots, p_d)$ and $(p'_1, \ldots, p'_d)$ in such a way that:

$$\gamma(p) = p'_0 \oplus \cdots \oplus p'_d \ .$$

Since γ is composed of several S-boxes, each operating on a subpart of p, the problem can be reduced to securely implement one S-box. The underlying problem is therefore the following.

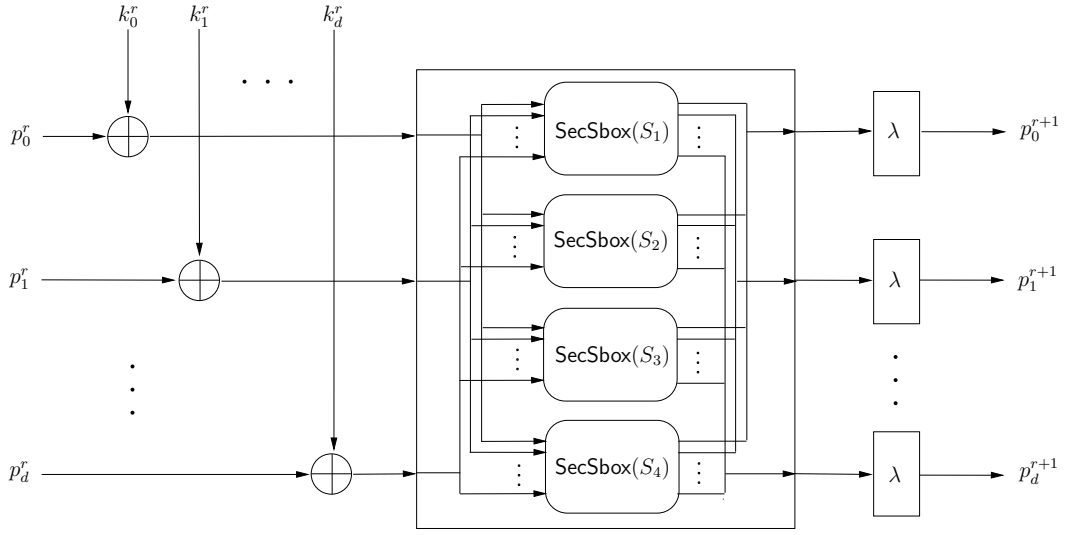


Figure 8.2: d^{th}-order masking scheme for a typical round transformation.

Problem 8.1. *Let S be a non-linear function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. From a masked input $x \oplus r_1 \oplus \cdots \oplus r_d \in \mathbb{F}_2^n$, the tuple of masks $(r_1, \ldots, r_d) \in \mathbb{F}_2^{n\,d}$ and a tuple of output masks $(s_1, \ldots, s_d) \in \mathbb{F}_2^{m\,d}$, compute $S(x) \oplus s_1 \oplus \cdots \oplus s_d$ without introducing any vulnerability with respect to d^{th}-order SCA.*

If the problem above can be resolved by an algorithm SecSbox, then the masked output state p'_0 can be constructed by performing each S-box computation through this algorithm.

Let us now assume that we have such a secure S-box implementation. The scheme described in Figure 8.2 can then be viewed as the protected version of the round transformation described in Figure 8.1. Finally, the entire block cipher algorithm protected against d^{th}-order SCA can be implemented as depicted in Algorithm 1.

Remark 8.2. We have described above a way to secure a round transformation φ. The secure implementation α_{sec} of the key scheduling function α can be straightforwardly deduced from this description since it is also composed of linear and non-linear layers.

Remark 8.3. When the notation $(p_0, p_1, \ldots, p_d) \leftarrow \ldots$ is used, the operation is assumed to be performed on each p_i separately.

Algorithm 1 Block Cipher secure against d^{th}-order SCA

INPUT: a plaintext $\mathbf{p}$, a masked key $k_0 = \mathbf{k} \oplus k_1 \oplus \cdots \oplus k_d$ and the masks $(k_1, \ldots, k_d)$
OUTPUT: the ciphertext $\mathbf{c}$

1. $(p_1, \ldots, p_d) \leftarrow \big(\mathsf{rand}(|\mathbf{p}|), \ldots, \mathsf{rand}(|\mathbf{p}|)\big)$
2. $p_0 \leftarrow \mathbf{p} \oplus p_1 \oplus \cdots p_d$
3. **for** $r = 0$ **to** $R - 1$ **do**
4. $\quad (k_0, k_1, \ldots, k_d) \leftarrow \alpha_{sec}\left((k_0, k_1, \ldots, k_d), r\right)$
5. $\quad (p_0, p_1, \ldots, p_d) \leftarrow (p_0 \oplus k_0, p_1 \oplus k_1, \ldots, p_d \oplus k_d)$
6. $\quad (p'_1, \ldots, p'_d) \leftarrow \big(\mathsf{rand}(|\mathbf{p}|), \ldots, \mathsf{rand}(|\mathbf{p}|)\big)$
7. $\quad$ **for** $j = 1$ **to** N **do** $(p'_0)_j \leftarrow \mathsf{SecSbox}\left(S_j, (p_0)_j, \ldots, (p_d)_j, (p'_1)_j, \ldots, (p'_d)_j\right)$
8. $\quad (p_0, \ldots, p_d) \leftarrow \left(\lambda\left(p'_0\right), \lambda\left(p'_1\right), \ldots, \lambda\left(p'_d\right)\right)$
9. **return** $p_0 \oplus p_1 \oplus \cdots \oplus p_d$

8.5.3 Security model

In order to prove the security of such a masking scheme, we need to introduce a few definitions. We shall say that a variable is *sensitive* if it is a function of the plaintext and the secret key (that is not constant with respect to the secret key). Additionally, we shall call *primitive random values* the intermediate variables that are generated by a random number generator (RNG) executed during the algorithm processing. Primitive random values are assumed to be uniformly distributed and mutually independent of each other.

In our security analyses, we will make a distinction between a statistical dependence and what we shall call a *functional dependence*. Every intermediate variable of a cryptographic algorithm can be expressed as a deterministic function of some sensitive variables and some primitive random values (generated by a RNG). When this function involves (resp. does not involve) a primitive or sensitive variable X, the intermediate variable is said to be *functionally dependent* (resp. *independent*) of X. If the distribution of an intermediate variable I varies (resp. does not vary) according to the value of a variable X then I is said to be *statistically dependent* (resp. *independent*) of X. It can be checked that the two notions are not equivalent since functional independence implies statistical independence but the converse is false. We give hereafter an example that illustrates the difference between these notions.

Example. Let X be a sensitive variable and let R be a primitive random value. The variable $I = X \oplus R$ is functionally dependent on X and on R. On the other hand, it is statistically independent of X since the probability $\mathrm{P}[X = x | I = i]$ is constant for every pair (x, i).

When the term (in)dependent is used alone, it refers to the statistical notion.

As introduced in Section 3.7, a d^{th}-order SCA is an SCA that exploits the leakages of d intermediate variables. According to [2, 46], we formally define hereafter the security against d^{th}-order SCA.

Definition 8.1. *A cryptographic algorithm is said to be* secure against d^{th}-order SCA *if every d-tuple of its intermediate variables is independent of any sensitive variable.*

Conversely, an algorithm is said to admit a *d^{th}-order flaw* if d of its intermediate variables jointly depend on a sensitive variable. Due to Definition 8.1, proving that an algorithm is secure against d^{th}-order SCA can be done by showing that all the d-tuples of its intermediate variables are independent of any sensitive variable. In order to simplify our security proofs, we introduce the notion of independence for a set.

Definition 8.2. *A set E is said to be* independent *of a variable X if every element of E is independent of X.*

By extension, Definition 8.2 implies that the cartesian product of d sets $E_1, \ldots, E_d$ is independent of a variable X if any d-tuple in $E_1 \times \cdots \times E_d$ is independent of X[15]. Thus, according to Definition 8.1, an algorithm processing a set $\mathcal{I}$ of intermediate variables is secure against d^{th}-order SCA if and only if $\mathcal{I}^d$ is independent of any sensitive variable. In the sequel we shall further say that a set admits a flaw (resp. admits no flaw) if one (resp. none) of its elements depends of a sensitive variable.

The following proposition provides a useful security reduction for the generic masking scheme described in Section 8.5.2.

Proposition 8.1. *Algorithm 1 is secure against d^{th}-order SCA if and only if* SecSbox *is secure against d^{th}-order SCA.*

Sketch of Proof. Let us denote by $\mathcal{I}$ the set of intermediate variables processed during one execution of Algorithm 1. Moreover, let us denote by $\mathcal{I}_r$ the set of intermediate variables processed during the r^{th} round of Algorithm 1; we have $\mathcal{I} = \bigcup_r \mathcal{I}_r$. Finally, let us denote by $\mathcal{S}_{r,j}$ the set of intermediate variables processed during the j^{th} execution of SecSbox at round r, and by $\mathcal{O}_r$ the set of the intermediate variables processed outside of the SecSbox calls at round r; we have $\mathcal{I}_r = \mathcal{O}_r \cup \bigcup_j \mathcal{S}_{r,j}$. We will argue that (1) $\mathcal{I}^d$ admits a flaw (namely a d-tuple of $\mathcal{I}^d$ depends on a sensitive variable) if and only if (2) there exists

[15] Unlike a set, a d-tuple is independent of a variable X if its d elements are jointly independent of X.

a pair (r, j) such that $\mathcal{S}^d_{r,j}$ admits a flaw. Since for every (r, j), we have $\mathcal{S}^d_{r,j} \subset \mathcal{I}^d$, (2) straightforwardly implies (1). Let us now show that (1) also implies (2).

Since the mask are re-generated at each round, if a d^{th}-order flaw occurs in the algorithm then it occurs within (at least) a given round r. That is, if $\mathcal{I}^d$ admits a flaw then there exists r such that $\mathcal{I}^d_r$ admits a flaw. This implies that an element of $\mathcal{I}^d_r$ depends on p^r. Since p^r is split into $d+1$ shares p^r_0, $\ldots$, p^r_d that are such that any d-tuple among them is independent of p^r, there exists at least one variable $x \in \mathcal{I}_r$ that jointly depends on at least two shares. It can be checked from Algorithm 1 that all the shares are manipulated strictly separately outside the SecSbox calls. Therefore, there exists j such that $x \in \mathcal{S}_{r,j}$ which implies that $\mathcal{S}^d_{r,j}$ has a flaw. □

Proposition 8.1 formally shows that plugging a secure S-box computation algorithm in the masking scheme described in the previous section implies the overall security for the block cipher implementation. That is why in the next chapters we shall focus on masking schemes for S-boxes. In order to prove the security of the proposed scheme, we will make use of the following lemmas.

Lemma 8.1. *Let X and Z be two independent random variables. Then, for every family $(f_i)_i$ of measurable functions the set $E = \{f_i(Z); i\}$ is independent of X.*

Lemma 8.2. *Let X be a random variable defined over $\mathbb{F}_2^n$ and let R be a random variable uniformly distributed over $\mathbb{F}_2^n$ and independent of X. Then $X \oplus R$ is independent of X. Moreover, let Z be a variable independent of R and functionally independent of X. Then the pair $(Z, X \oplus R)$ is independent of X.*

When a sensitive variable is masked with two primitive random values, then Lemmas 8.1 and 8.2 imply that no second-order leakage occurs if the three shares are always processed separately.

8.6 Design and cryptanalysis of masking schemes

In the next two chapters, we propose new first-order and second-order masking schemes for S-boxes. For each of them, we formally prove the security against first-order/second-order SCA with respect to the security model introduced in the previous section. According to Proposition 8.1, using one of these methods together with the block cipher masking scheme described above guarantees an overall first-order/second-order SCA security.

A different approach is followed in Chapter 11, where we propose a scheme combining higher-order masking with operation shuffling. This scheme does not aim at provable security against SCA of given order. Rather, it ensures a chosen resistance level against some advanced DPA attacks which are especially effective against masking and shuffling.

Finally, we present in Chapters 12 and 13 the cryptanalysis of two masking schemes. The first one aims to thwart first-order SCA. We show that a first-order attack is in fact possible in the presence of this scheme. The second one aims to thwart d^{th}-order SCA for a chosen security parameter d. We show that, whatever the value of d, the scheme is vulnerable to several third-order attacks, which invalidates its higher-order security. These results underline the importance of proving the security of masking schemes. For the first of the two broken schemes, a security proof was given which was flawed, while, for the second one, no security proof was provided.

Chapter 9

Generic first-order masking schemes for S-boxes

Contents

9.1 Introduction

In this chapter, we address generic first-order masking schemes for S-boxes. After a review of the existing solutions, we present a new generic method to perform an S-box computation on masked data. Our method performs the computation on-the-fly and it does not require any RAM allocation. We compare it to the previous solutions and we give implementation results.

The results presented in this chapter have been published in collaboration with Emmanuel Prouff in the international workshop on *Information Security Applications (WISA 2007)* [4]. The presented method has also been patented in [15].

9.2 State of the art

9.2.1 Table re-computation method

The table re-computation method [163] is the most natural way to mask an S-box computation using a look-up table. It consists in pre-computing a masked version of the S-box look-up table which is then involved each time the S-box must be computed. The simplicity and the efficiency of this method makes it widely used to protect S-boxes computations at the first order.

Let n and m be the input and output bit-lengths of S. Let T denote a table of 2^n m-bit words allocated in RAM. The table re-computation algorithm is described hereafter.

Algorithm 2 re-compute

INPUT: an input mask r, an output mask s
OUTPUT: the table $T[\cdot] = S(\cdot \oplus r) \oplus s$

1. **for** $a = 0$ **to** $2^n - 1$ **do**
2. $\quad T[a] \leftarrow S(a \oplus r) \oplus s$
3. **return** T

Usually the masked table is computed once at the beginning of the cipher for a pair of random masks (r^*, s^*) and it is then involved performing every masked S-box computation such as described hereafter.

Algorithm 3 SecSbox-trc

INPUT: a masked input $\tilde{x} = x \oplus r$, an input mask r, an output mask s, a table $T[\cdot] = S(\cdot \oplus r^*) \oplus s^*$
OUTPUT: the masked output $S(x) \oplus s$

1. $tmp \leftarrow \tilde{x} \oplus r^*$ $\quad [tmp = x \oplus r \oplus r^*]$
2. $tmp \leftarrow tmp \oplus r$ $\quad [\ tmp = x \oplus r^*]$
3. $tmp \leftarrow T[tmp]$ $\quad [tmp = S(x) \oplus s^*]$
4. $tmp \leftarrow tmp \oplus s$ $\quad [tmp = S(x) \oplus s^* \oplus s]$
5. $tmp \leftarrow tmp \oplus s^*$ $\quad [tmp = S(x) \oplus s]$
6. **return** tmp

Complexity analysis. The re-compute algorithm requires $2^n \times 2$ logical operations (2 XOR operations per loop iteration) and $2^n \times 2$ memory transfers (1 read operation and 1 write operation per loop iteration). Moreover, it requires the allocation of $2^n m$ bits of RAM. The SecSbox-trc algorithm requires 4 logical operations and 1 memory transfer.

Remark 9.1. When a block cipher involves several (say N_S) different S-boxes, a masked table must be pre-computed per S-box. Besides, the re-compute algorithm must be executed N_S times and N_S tables must be allocated in RAM. For instance, the DES algorithm involved 8 different S-boxes; it hence required 8 table re-computations.

Security analysis. It can be checked from Algorithm 3 that every intermediate variable of the computations is masked with a uniform random mask and is hence independent of x. Consequently, SecSbox-trc is secure against first-order SCA.

9.2.2 Global look-up table method

The global look-up table method makes use of a look-up table T for the function $(y_1, y_2) \mapsto S(y_1 \oplus y_2) \oplus y_2$. This look-up table is stored in ROM and it does not require any dynamic pre-computation. The masked S-box computations are then performed as follows.

Algorithm 4 SecSbox-glut

INPUT: a masked input $\tilde{x} = x \oplus r$, an input mask r, an output mask s, the table $T[y_1, y_2] = S(y_1 \oplus y_2) \oplus y_2$
OUTPUT: the masked output $S(x) \oplus s$

1. $tmp \leftarrow T[\tilde{x}, r]$ $\quad [tmp = S(x) \oplus r]$
2. $tmp \leftarrow tmp \oplus s$ $\quad [tmp = S(x) \oplus r \oplus s]$
3. $tmp \leftarrow tmp \oplus r$ $\quad [tmp = S(x) \oplus s]$
4. **return** tmp

Complexity analysis. The timing performances of the global look-up table method are ideal since it requires only two logical operations and one memory transfer. However, the bit-size $2^{2n}m$ of the look-up table T makes an application of the method difficult in a low resource context. For instance, if $n = m = 8$, the amount of ROM required (64 kilobytes!) is definitively too large. When n is lower, the feasibility of the method depends on the amount of ROM of the device and on the number of different S-boxes which must be protected. The method can become interesting when it comes to protect S-boxes mapping $\mathbb{F}_2^4$ into itself (as it is the case for FOX [136] where three such S-boxes are involved) or when the S-box computation can be performed in spaces of dimensions lower than or equal to 4 (as it is the case for the AES S-box, see Section 9.4.2).

Security analysis. From a security point of view, the global look-up table method has a flaw since it manipulates the pair $(\tilde{x}, r)$ to address the look-up table T (see Step 1 of Algorithm 4). In practice, this implies the processing of $\tilde{x}||r$ (where $||$ denotes the concatenation operator). The latter depends on x which induces a first-order flaw. For instance, in the Hamming weight model, it can be checked that x is statistically dependent of $\mathrm{HW}((x \oplus r)||r)$, which results in an information leakage. Moreover, the input and output masks being equal, this method has also a potential flaw in the Hamming Distance model. Indeed, if a transition occurs between the index $(x \oplus r)||r$ and the value $S(x) \oplus r$ accessed in the look-up table (which is very likely in a single bus architecture), the mask r is canceled and some information leaks about x and/or $S(x)$.

9.2.3 Fourier transform based method

The Fourier transform based method processes a masked S-box computation on-the-fly. Its main advantage is that it does not required any RAM allocation.

As we show in Chapter 12, the original version proposed in [192] has a first-order flaw which enables a first-order SCA attack. We also propose a corrected version of the method whose basic principle is to evaluate the following equation (m is assumed equal to n):

$$(-1)^{r'} S(x) + s \bmod 2^n = \left\lfloor \frac{1}{2^n} \left((2^n s + s') + \sum_{a \in \mathbb{F}n} \widehat{S}(a)(-1)^{(r' \oplus a \cdot \tilde{x}) \oplus a \cdot r} \bmod 2^{2n} \right) \right\rfloor \quad , \tag{9.1}$$

where $\widehat{S}$ denotes the Fourier transform of S, r' is a random mask over $\mathbb{F}_2$, s' is a random mask over $\mathbb{F}_2^n$.

It is worth noting that the obtained value is masked arithmetically (*i.e.* by modular addition). This implies that, once (9.1) has been computed, one must still convert the arithmetic mask s into a Boolean mask (after canceling the $(-1)^{r'}$ factor) to obtain the desired masked output. Such a conversion can be done using the method in [117]. All in all, the Fourier transform based method is quite expensive in terms of timings.

We do not give details about the complexity and security of the Fourier transform based method here (see Chapter 12 for details). The next section present an alternative on-the-fly masked S-box computation that neither require RAM allocation and that is clearly more efficient than the Fourier Transform based method.

9.3 A new on-the-fly masked S-box computation

9.3.1 Description

The core idea of our proposal is to compute $S(\tilde{x} \oplus a) \oplus s$ for every value a, storing the result in a register R_0 if a equals r and in a second register R_1 otherwise.

Let $\mathsf{compare} : x, y \mapsto \mathsf{compare}(x, y)$ be the function returning 0 if $x = y$ and 1 otherwise. We depict our proposal in the following algorithm.

Algorithm 5 SecSbox-otf

INPUT: a masked input $\tilde{x} = x \oplus r$, an input mask r, an output mask s, the S-box S
OUTPUT: the masked output $S(x) \oplus s$

1. **for** $a = 0$ **to** $2^n - 1$ **do**
2. $\quad cmp \leftarrow \mathsf{compare}(a, r)$
3. $\quad R_{cmp} \leftarrow S(\tilde{x} \oplus a) \oplus s$
4. **return** R_0

Remark 9.2. Many microprocessors implement the function compare by a single instruction. Thus, we will assume in the following that this function is an elementary logical operation.

To verify the correctness of Algorithm 5, it can be checked that Step 3 performs the following operation:

$$\begin{cases} R_0 \leftarrow S(\tilde{x} \oplus a) \oplus s & \text{if } a = r \ , \\ R_1 \leftarrow S(\tilde{x} \oplus a) \oplus s & \text{otherwise} \ . \end{cases} \tag{9.2}$$

Hence, R_0 contains the value $S(\tilde{x} \oplus r) \oplus s = S(x) \oplus s$ when the loop is completed.

9.3.2 Complexity analysis

Algorithm 5 requires $2^n \times 3$ logical operations (2 XOR operations and 1 comparison per loop iteration) and 2^n memory transfers (1 table look-up per loop operation).

9.3.3 Security analysis

The security of the new method highly depends on the assumption that the leakage generated by a register transfer does not enable to determine which register has been used. This assumption is commonly accepted and it is the security core of SPA countermeasures used to protect asymmetric cryptosystems (see for instance [135]).

Table 9.1: Intermediate variables of Algorithm 5.

Step	Intermediate Variables
2	r
2	$\mathsf{compare}(a, r)$
3	$S(\tilde{x} \oplus a)$
3	$S(\tilde{x} \oplus a) \oplus s$
4	$S(x) \oplus s$

We list in Table 9.1 the intermediate variables processed in Algorithm 5. Based on Lemma 8.2, we clearly see that all the intermediate variables processed in Algorithm 5 are independent of x. As a result, SecSbox-otf is secure against first-order SCA.

Remark 9.3. When $a \neq r$, Step 3 performs a dummy operation. As we explain in [4], this may induce a flaw with respect to combined side channel and fault attacks. Regarding this issue, a variant is provided in [4] that avoids dummy operations.

9.4 Comparison and application

9.4.1 Comparison

In what follows, we compare the complexity of our proposal with the ones of other generic methods.

Table 9.2 summarizes the costs of the table re-computation method, the global look-up table method and our on-the-fly computation method with respect to the number of logical operations (LO), the number of memory transfers (MT), the memory size (bits in RAM) and the code size (bits in ROM).

Table 9.2: Time & memory complexities of the generic first-order masking schemes for S-boxes.

Method	Pre-computation	S-box computation	RAM	ROM
Table re-computation	$2^{n+1}\mathtt{MT} + 2^{n+1}\mathtt{LO}$	$1\mathtt{MT} + 4\mathtt{LO}$	$2^n m$	$2^n m$
Global LUT	0	$1\mathtt{MT} + 2\mathtt{LO}$	0	$2^{2n} m$
On-the-fly computation	0	$2^n\mathtt{MT} + 3 \times 2^n\mathtt{LO}$	0	$2^n m$

The three methods provide different time-memory tradeoffs. Our method is clearly the least efficient in terms of timings. However, contrary to the table re-computation method, it does not require any RAM allocation. And contrary to the global LUT method, it does not require a huge amount of ROM and it does not suffer any security flaw. Note that for low-cost devices, timings are sometimes less critical than memory. In such a situation, our solution is of great interest compared to the others.

9.4.2 Application to AES

We recall that the AES S-box is composed of two parts: a non-linear function and an affine mapping. In the following we focus on the non-linear part, which will be denoted here by F. Let $p(x)$ denotes the irreducible polynomial $x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1 \in \mathbb{F}_2[x]$. The function F is defined in $\mathbb{F}_2[x]/p(x)$ by:

$$F(a) = \begin{cases} 0 & \text{if } a = 0, \\ a^{-1} & \text{otherwise.} \end{cases}$$

At first, we applied the method depicted in Algorithm 5 for $n = 8$ to protect the S-box access of the AES algorithm. We implemented the solution on a classical 8051 chip running at 8 Mhz and we studied the performances of the implementation. Clearly, the timing overhead of the resulting AES implementation (approximately 115 ms) is considerable compared to an implementation without countermeasures (approximately 5 ms).

Secondly, we represented $\mathbb{F}_{2^8}$ has an extension of $\mathbb{F}_{2^4}$, allowing us to perform the computations in $\mathbb{F}_{2^4}$ instead of $\mathbb{F}_{2^8}$ (such a method is usually called *composite field approach*). We chose the two irreducible polynomials $p'(x) = x^2 + x + \{e\}$ and $p''(x) = x^4 + x + 1$ in $\mathbb{F}_4[x]$ and $\mathbb{F}_2[x]$ respectively and we denoted by *map* the field isomorphism which takes an element a of $\mathbb{F}_2[x]/p(x)$ as input and outputs the pair $(a_h, a_l) \in (\mathbb{F}_2[x]/p''(x))^2$ corresponding to the coefficients of the linear polynomial $(a_h x + a_l) \in \mathbb{F}_{2^4}[x]/p'(x)$. Moreover, we denoted by $\mathrm{Inv}_{\mathbb{F}_{2^4}}$ the function which corresponds to the inverse function over $\mathbb{F}_2[x]/(x^4+x+1)\backslash\{0\}$ and which maps 0 into itself. The following algorithm describes the different steps of our computation.

Algorithm 6 Inversion of a masked element $\widetilde{a} = a \oplus m_a$ in $\mathbb{F}_{2^8}$

INPUT: $(\widetilde{a} = a \oplus m_a, m_a) \in {\mathbb{F}_{2^8}}^2$

OUTPUT: $(\widetilde{a^{-1}} = a^{-1} \oplus m'_a, m'_a)$

1. Pick up three 4-bit random m_d, m'_h and m'_l
2. $(m_h, m_l) \in \mathbb{F}_{2^4}^2 \leftarrow map(m_a)$
3. $(\widetilde{a_h}, \widetilde{a_l}) \in \mathbb{F}_{2^4}^2 \leftarrow map(\widetilde{a})$ $\quad [(\widetilde{a_h}, \widetilde{a_l}) = (a_h \oplus m_h, a_l \oplus m_l)]$
4. $\widetilde{d} \leftarrow \widetilde{a_h}^2 \otimes \{e\} \oplus \widetilde{a_h} \otimes \widetilde{a_l} \oplus \widetilde{a_l}^2 \oplus m_d \oplus \widetilde{a_h} \otimes m_l$ $\quad [\widetilde{d} = d \oplus m_d]$
   $\oplus\, \widetilde{a_l} \otimes m_h \oplus m_h^2 \otimes \{e\} \oplus m_l^2 \oplus m_h \otimes m_l$
5. $\widetilde{d^{-1}} \leftarrow \mathsf{SecSbox\text{-}otf}(\widetilde{d}, m_d, m_{d^{-1}}, \mathrm{Inv}_{\mathbb{F}_{2^4}})$ $\quad [\widetilde{d^{-1}} = d^{-1} \oplus m_{d^{-1}}]$
6. $\widetilde{a'_h} \leftarrow \widetilde{a_h} \otimes \widetilde{d^{-1}} \oplus m'_h \oplus m_h \otimes \widetilde{d^{-1}} \oplus m_{d^{-1}} \otimes \widetilde{a_h} \oplus m_{d^{-1}} \otimes m_h$ $\quad [\widetilde{a'_h} = a'_h \oplus m'_h]$
7. $\widetilde{a'_l} \leftarrow \widetilde{a_l} \otimes \widetilde{d^{-1}} \oplus m'_l \oplus \widetilde{a'_h} \oplus \widetilde{d^{-1}} \otimes m_l \oplus \widetilde{a_l} \otimes m_{d^{-1}} \oplus\, m'_h \oplus m_l \otimes m_{d^{-1}}$ $\quad [\widetilde{a'_l} = a'_l \oplus m'_l]$
8. $m'_a \leftarrow map^{-1}(m'_h, m'_l)$

9. $\widetilde{a^{-1}} \leftarrow map^{-1}(\widetilde{a'_h}, \widetilde{a'_l})$ $[\widetilde{a^{-1}} = a^{-1} \oplus m'_a]$
10. **return** $(\widetilde{a^{-1}}, m'_a)$

For this version, the timing of the resulting AES algorithm are interesting and the input and output masks can be changed at each execution of the algorithm. In Table 9.3, we have listed the timing and memory performances of our proposal and the ones of other methods proposed in the literature. As the performances have been measured for a particular implementation on a particular architecture, the table above does not aim at arguing that a method is better than another but aims at enlightening the main particularities (timing performances and ROM/RAM requirements) of each method.

Table 9.3: Comparison of several first-order masking schemes for AES.

Method	Timings (ms)	RAM (bytes)	ROM (bytes)
Unprotected implementation	5	0	1150
Table re-computation method			
Re-computation method in $\mathbb{F}_{2^8}$	×1.42	+256	+49%
Re-computation method in $\mathbb{F}_{2^4}$	×2.60	+16	+150%
Masking schemes with the composite field approach			
Oswald *et al.* [177, 178]	×5.20	0	+173%
Prouff *et al.* [192]	×6.40	0	+147%
Our scheme (Algo. 6)	×5.30	0	+150%
methods with security weaknesses			
Oswald and Schramm [179]	×2.40	0	+200%
Trichina *et al.* [235]	×4.20	+256	+165%

The AES implementations listed above only differ in their approaches to protect the S-box access. The linear steps of the AES have been implemented in the same way and the internal sensitive data have been masked by bitwise addition of a random value. We chose to protect only rounds 1 to 3 and 8 to 10, assuming that the diffusion properties of the AES algorithm make SCA attacks impossible to mount on inner rounds 4 to 7 (this implies that the S-box calculations made in rounds 4 to 7 are performed by simply accessing the table representation of the S-box which is stored in ROM).

The table re-computation method in $\mathbb{F}_{2^8}$ has the best timing performances but at least 256 bytes of RAM must be allocated to store the re-computed look-up table. As RAM is a sensitive resource in the area of embedded devices, we implemented a second version which follows the outlines of the composite field approach and then applies the re-computation method in $\mathbb{F}_{2^4}$. Because only 16 bytes of RAM are required to store the table re-computed from the $\text{Inv}_{\mathbb{F}_{2^4}}$ function, the new implementation requires much less RAM than the version in $\mathbb{F}_{2^8}$ and the timing performances are suitable for practical applications.

The masking schemes of Elisabeth Oswald *et al.* [178], Emmanuel Prouff *et al.* [192] and ours only differ in the way of securely computing the value $d^{-1} \oplus m_{d^{-1}}$ from $\tilde{d}$, m_d and $m_{d^{-1}}$ (*i.e.* to securely perform the fifth Step of Algorithm 6):

— In [177,178], the masked inversion is performed by going down to $\mathbb{F}_4$ and its complexity approximatively equals the one of Algorithm 6 excluding the 5^{th} Step which is replaced by a square operation (since the inversion operation in $\mathbb{F}_4$ is equivalent to squaring). For our implementation of [177,178], the number of cycles required for the fifth step is 267.

— In [192], the masked inversion is essentially performed by computing a Fourier transform on $\mathbb{F}_2^4$. For our implementation of [192], the number of cycles required for the fifth step is 468.

— For the new solution presented here, the masked inversion essentially corresponds to the computation of $y^{-1} \oplus m_{d^{-1}}$ for every $y \in \mathbb{F}_2^4$. For our implementation, the number of cycles required by the fifth step is 270.

The execution timings of AES implementations based on our proposal or on the scheme of Elisabeth Oswald *et al.* are very close and their RAM requirements are almost equal. The additional time required by the scheme of Emmanuel Prouff *et al.* is slightly greater, however the code seems to be shorter (2844 bytes of ROM *versus* 2881 and 3144 bytes of ROM for our method and the scheme of Elisabeth Oswald *et al.*).

Remark 9.4. Compared to the table re-computation method the three previous schemes have the advantage that they do not use the same mask for every S-box computation. Indeed, as explained in Section 9.2.1, the masked table is computed once at the beginning of the algorithm for a fix pair of masks. As already put forward in [114, 179], second-order SCA are especially effective when the same pair of masks is used to protect all the table look-ups. Therefore, depending on the context, the three previous schemes may have a better practical security than the table re-computation method.

The methods proposed by Elena Trichina and Lesya Korkishko [235] and by Elisabeth Oswald and Kai Shramm [179] have good timing performances but are not perfectly resistant to first-order SCA attacks.

— In the method of Elena Trichina and Lesya Korkishko, a *primitive element* of $\mathbb{F}_{2^8}$ is computed and every non-zero element of $\mathbb{F}_{2^8}$ is expressed as a power of that element. A pre-computed discrete logarithm table and an exponentiation table are then involved in the S-box operation. As argued in [179], the method has a faulty behavior when some intermediate values are zero and correcting the method without introducing a flaw with respect to first-order attacks seems to be an issue.

— The method proposed by Elisabeth Oswald and Kai Shramm offers the best timing performances. As for Algorithm 6, it is based on the composite field approach but steps 4 to 7 are replaced by a sequence of table look-ups and bitwise additions. The table look-ups have been render resistant to first-order SCA attacks by applying the global look-up table method (which is recalled in Section 9.2.2). For example, the computation of $d^{-1} \oplus m_{d^{-1}}$ (Step 5) is performed by accessing the table T_{inv} associated to the function $((d \oplus m_d), m_d) \in (\mathbb{F}_{2^4})^2 \mapsto (d^{-1} \oplus m_d) \in \mathbb{F}_{2^4}$. As argued in Section 9.2.2, the global look-up table method has a flaw with respect to first-order SCA attacks. Indeed, to address the T_{inv} table the value $(d \oplus m_d)||m_d$ is manipulated, which results in an information leakage on the sensitive value d.

Chapter 10

Generic second-order masking schemes for S-boxes

Contents

10.1 Introduction

In this chapter, we address generic second-order masking schemes for S-boxes. After presenting a state of the art, we introduce two new solutions to perform an S-box computation secure against second-order SCA. We prove their security in the security model introduced in Section 8.5.3 and we exhibit a way to significantly improve their efficiency by using the particularities of the targeted architectures. Finally, we compare our methods to the existing solutions and we give implementation results.

The results presented in the chapter have been published in collaboration with Emmanuelle Dottax and Emmanuel Prouff in the international workshop on *Fast Software Encryption (FSE 2008)* [10]. The presented methods have also been patented in [14–16].

10.2 State of the art

10.2.1 Higher-order SCA resistance in the literature

Only a few papers deal with higher-order SCA resistance in the literature. It seems that the first attempt has been proposed by Mehdi-Laurent Akkar and Louis Goubin in [23] for securing the DES algorithm. The proposed solution had some flaws that were fixed in two steps, firstly by Mehdi-Laurent Akkar, Régis Bévan and Louis Goubin in [21] and secondly by Jiqiang Lv and Yongfei Han in [153]. All these

works are more or less based on the same principle. At each DES execution, a few 32-bit masks are generated (2 masks in [21, 23], and 3 masks in [153]). The masks are used to derive several new masked S-boxes from each of the 8 DES S-boxes, which are then combined in different ways. In the security discussion conducted in [23] and implicitly assumed in [21, 153], the masks manipulations and the table re-computations are supposed to leak no information about the masks values. Although the different methods proposed in [23] to process the mask values and the masked S-boxes allow to minimize the instantaneous leakages on the mask values, they do not perfectly prevent it. Therefore, according to Definition 8.1, the implementations proposed in [21, 23, 153] are not secure against second-order SCA.

A higher-order masking scheme has been proposed by Yuval Ishai, Amit Sahai, and David Wagner in [126] to protect a logical AND. Based on such a scheme, every circuit can theoretically be secured against d^{th}-order SCA for a given d since every circuit can be decomposed in Boolean functions performing XORs and ANDs. To the best of our knowledge, this is the only scheme that is provably secure against d^{th}-order SCA. However, this scheme does not take glitches into account which makes its practical security an issue for hardware implementations (see [158, 159]). Moreover, it is not suitable for software implementations. Although every S-box computation could be decomposed and secured at the Boolean level, the implied timing overhead of such a decomposition in software would be clearly prohibitive.

The only generic higher-order masking scheme suitable in software (as well as in hardware) as been proposed by Kai Schramm and Christof Paar in [204] for the AES block cipher. Their scheme merely consists of a generalization of the table re-computation method (see Section 9.2.1) to higher orders and it was initially claimed to be secure against d^{th}-order SCA for any chosen parameter d. However, we show in Chapter 13 that the Schramm and Paar's scheme is vulnerable to several third-order attacks which makes it reliable only for $d = 2$ *i.e.* to secure implementations of block ciphers against second-order SCA. The next section describes the Schramm and Paar's scheme for $d = 2$.

10.2.2 Schramm and Paar's scheme

The Schramm and Paar's scheme is a generalization of the table re-computation method (see Section 9.2.1) for higher-order masking. At the second order, the principle is to compute from a 4-tuple of masks (r_1, r_2, s_1, s_2), the look-up table of a masked S-box S^* that satisfies $S^*(x) = S(x \oplus r_1 \oplus r_2) \oplus s_1 \oplus s_2$ for every $x \in \mathbb{F}_2^n$. Then, the masked output $S(x) \oplus s_1 \oplus s_2$ is obtained by simply accessing the value $S^*(\tilde{x})$ in the re-computed look-up table. The tricky part in such a method is the construction of the look-up table for S^* which must never manipulate the sum of masks $r_1 \oplus r_2$ and $s_1 \oplus s_2$ directly (since it would introduce a second-order flaw together with $x \oplus r_1 \oplus r_2$ or with $S(x) \oplus s_1 \oplus s_2$).

In their paper, Kai Schramm and Christof Paar present two schemes. The first one (the generic) involves two table re-computations[16].

Algorithm 7 SecSbox-2O-SP1 (Schramm and Paar's generic scheme)

INPUT: a LUT for S, the masks (r_1, r_2, s_1, s_2), the masked input $\tilde{x} = x \oplus r_1 \oplus r_2$
OUTPUT: the masked output $S(x) \oplus s_1 \oplus s_2$

1. $S_{tmp} \leftarrow$ re-compute(S, r_1, s_1) $\quad [S_{tmp}(\cdot) = S(\cdot \oplus r_1) \oplus s_1]$
2. $S^* \leftarrow$ re-compute(S_{tmp}, r_2, s_2) $\quad [S^*(\cdot) = S(\cdot \oplus r_1 \oplus r_2) \oplus s_1 \oplus s_2]$
3. **return** $S^*(\tilde{x})$

As noticed by Kai Schramm and Christof Paar in [204], the algorithm above is quite costly as it involves two table re-computations for each S-box computation for each round of the cipher. To reduce this overhead, the authors propose an improvement: two successive table re-computations are still preformed to process the first masked S-box in the first round of the cipher but all the other S-box computations are protected with a single table re-computation. Before describing the method, let us assume that the previous S-box computation has been protected with the 4-tuple of masks (r'_1, r'_2, s'_1, s'_2) and with a masked S-box S^*_{prev} (satisfying $S^*_{prev}(y) = S(y \oplus r'_1 \oplus r'_2) \oplus s'_1 \oplus s'_2$ for every $y \in \mathbb{F}_2^n$). To securely compute

[16] We consider that the table re-computations are performed using the straightforward algorithm since, as argued in Section 13.2.3, other proposals of [204] include some flaws.

the new masked output $S(x) \oplus s_1 \oplus s_2$ from the masked input $\tilde{x}$, the masked S-box S^* is derived from S^*_{prev}. Then, the value $S^*(\tilde{x})$ is accessed to get $S(x) \oplus s_1 \oplus s_2$.

Algorithm 8 SecSbox-2O-SP2 (Schramm and Paar's improved scheme)
INPUT: a LUT for S^*_{prev}, the 4-tuples (r_1, r_2, s_1, s_2) and (r'_1, r'_2, s'_1, s'_2), the masked input $\tilde{x} = x \oplus r_1 \oplus r_2$
OUTPUT: the masked output $S(x) \oplus s_1 \oplus s_2$ (and $LUT(S^*)$)

1. icm $\leftarrow (r_1 \oplus r'_1) \oplus r_2 \oplus r'_2$
2. ocm $\leftarrow (s_1 \oplus s'_1) \oplus s_2 \oplus s'_2$
3. $S^* \leftarrow$ re-compute$(S^*_{prev}, \text{icm}, \text{ocm})$
4. **return** $S^*(\tilde{x})$

Schramm and Paar's improved scheme is clearly more efficient than the generic scheme and it seems to be secure against second-order SCA. However, its use as algorithm SecSbox in the scheme described in Section 8.5.2 may potentially introduce a second-order flaw in the implementation (see below).

Complexity analysis. The Schramm and Paar's generic scheme (Algorithm 7) involves two table re-computations, that is 4×2^n XOR operations and $4 \times 2^n (+2)$ memory transfers, plus 1 memory transfer. The Schramm and Paar's improved scheme (Algorithm 8) involves a single table re-computation, that is 2×2^n XOR operations and 2×2^n memory transfers, plus 6 XOR operations and 1 memory transfer. Moreover, both schemes require the allocation of $2 \times 2^n \times m$ bits of RAM. If different S-boxes are involved in the cipher (such as in DES), the improved scheme requires a total of $(1 + N_S) \times 2^n \times m$ bits of RAM (where N_S stands for the number of different S-boxes involved in the cipher).

Security analysis. Although it has never been demonstrated, the generic Schramm and Paar's scheme seems to be secure against second-order SCA. Regarding the improved scheme, its use as algorithm SecSbox in the scheme described in Section 8.5.2 may potentially introduce a second-order flaw. In fact, the proof given in Section 8.5.3 does not apply in this case since Algorithm 8 does not fulfill the definition of SecSbox (see Problem 8.1). Indeed, Algorithm 8 takes additional parameters (r'_1, r'_2, s'_1, s'_2) which may induce a second-order flaw. For instance, let us assume that the linear layer λ operates on a pair of successive S-box outputs $(S(x'), S(x))$ and that it computes a value taking the form $S(x') \oplus S(x)$ (this is the case in AES). When second-order masking is used, this sum of successive S-box outputs takes the form $(S(x') \oplus s'_1 \oplus s'_2) \oplus (S(x) \oplus s_1 \oplus s_2)$ that is $(S(x') \oplus S(x)) \oplus (s'_1 \oplus s'_2 \oplus s_1 \oplus s_2)$. Consequently, if Algorithm 8 is used to implement the S-box computations, then targeting the later sum together with ocm $= s'_1 \oplus s'_2 \oplus s_1 \oplus s_2$ (which is processed during Step 2) reveals some information about the sensitive variable $S(x') \oplus S(x)$. To circumvent such a flaw, additional countermeasures have to be added to the implementation of the linear layer λ. The resulting overhead depends on the definition of λ. In the implementation results given in Section 10.4.2, we do not take this overhead into account for simplicity reasons (namely, the implementations involving Algorithm 8 may have such a flaw). However, we point out that the issue described above must not be neglected when it comes to implement Schramm and Paar's improved scheme.

10.3 Two generic second-order masking schemes for S-boxes

In this section, we first describe two methods to implement any function $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and we prove their security against second-order SCA. Then we propose an improvement that allows us to substantially reduce the complexity of both methods.

10.3.1 First scheme

The following algorithm describes a method to securely process a second-order masked S-box output from a second-order masked input.

Algorithm 9 SecSbox-2O-RDP1

INPUT: a masked value $\tilde{x} = x \oplus r_1 \oplus r_2 \in \mathbb{F}_2^n$, the pair of input masks $(r_1, r_2) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, a pair of output masks $(s_1, s_2) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$, a look-up table for S
OUTPUT: the masked S-box output $S(x) \oplus s_1 \oplus s_2 \in \mathbb{F}^m$

1. $r_3 \leftarrow \mathsf{rand}(n)$
2. $r' \leftarrow (r_1 \oplus r_3) \oplus r_2$
3. **for** $a = 0$ **to** $2^n - 1$ **do**
4. $\quad a' \leftarrow a \oplus r'$
5. $\quad T[a'] \leftarrow (S(\tilde{x} \oplus a) \oplus s_1) \oplus s_2$
6. **return** $T[r_3]$

Remark 10.1. In the description of Step 5, we used brackets to point out that the introduction of the two output masks s_1 and s_2 is done in this very order (otherwise a second-order flaw would occur).

The random value r_3 is used to mask the sum $r_1 \oplus r_2$ in order to avoid any second-order flaw. The value returned at the end of the algorithm satisfies: $T[r_3] = S(\tilde{x} \oplus r_3 \oplus r') \oplus s_1 \oplus s_2 = S(x) \oplus s_1 \oplus s_2$, which proves the correctness of Algorithm 9.

10.3.1.1 Complexity

Algorithm 9 requires the allocation of a table of 2^n m-bit words in RAM. It involves 4×2^n (+2) XOR operations, 2×2^n (+1) memory transfers and the generation of n random bits.

10.3.1.2 Security analysis

We prove hereafter that Algorithm 9 is secure against second-order SCA.

Security Proof. Algorithm 9 involves four primitive random values r_1, r_2, s_1 and s_2. These variables are assumed to be uniformly distributed and mutually independent together with the sensitive variable x.

The intermediate variables of Algorithm 9 are viewed as functions of the loop index a and are denoted by $I_j(a)$. The set $\{I_j(a); 0 \leqslant a \leqslant 2^n - 1\}$ is denoted by I_j. If an intermediate variable $I_j(a)$ does not functionally depend on a, then the set I_j is a singleton. The set $\mathcal{I} = I_1 \cup \cdots \cup I_{15}$ of all the intermediate variables of Algorithm 9 is listed in Table 10.1.

Remark 10.2. In Table 10.1, the step values refer to the lines in the algorithm description (where Step 0 refers to the input parameters manipulation). Note that one step (in the algorithm description) can involve several intermediate variables. However, these ones are separately processed and do not leak information at the same time.

In order to prove that Algorithm 9 is secure against second-order SCA, we need to show that $\mathcal{I} \times \mathcal{I}$ is independent of x. For this purpose, we split $\mathcal{I}$ into the three subsets $E_1 = I_1 \cup \cdots \cup I_9$, $E_2 = I_{10} \cup \cdots \cup I_{14}$ and I_{15}. First, the sets $E_1 \times E_1$, $E_2 \times E_2$ and $I_{15} \times I_{15}$ are shown to be independent of x. Then, we show that $E_1 \times E_2$, $E_1 \times I_{15}$ and $E_2 \times I_{15}$ are also independent of x, thus proving the independence between $\mathcal{I} \times \mathcal{I}$ and x.

The set $E_1 \times E_1$ is independent of x since E_1 is functionally independent of x. Moreover, since $x \oplus r_1 \oplus r_2$ (resp. $S(x) \oplus s_1 \oplus s_2$) is independent of x and since each element in $E_2 \times E_2$ (resp. $I_{15} \times I_{15}$) can be expressed as a function of $x \oplus r_1 \oplus r_2$ (resp. $S(x) \oplus s_1 \oplus s_2$), then Lemma 8.1 implies that $E_2 \times E_2$ (resp. $I_{15} \times I_{15}$) is independent of x.

One can check that E_1 is independent of $r_1 \oplus r_2$ and is functionally independent of x. Hence, we deduce from Lemma 8.2 that $E_1 \times \{x \oplus r_1 \oplus r_2\}$ is independent of x, which implies (from Lemma 8.1) that $E_1 \times E_2$ and x are independent. Similarly, E_1 is independent of $s_1 \oplus s_2$ so that $E_1 \times \{I_{15}\}$ (namely $E_1 \times \{S(x) \oplus s_1 \oplus s_2\}$) is independent of $S(x)$ and hence of x.

Table 10.1: Intermediate variables of Algorithm 9.

j	I_j	Steps
1	r_1	0,2
2	r_2	0,2
3	s_1	0,2
4	s_2	0,2
5	r_3	1,6
6	$r_1 \oplus r_3$	2
7	$r_1 \oplus r_2 \oplus r_3$	2,4
8	a	3,4,5
9	$a \oplus r_1 \oplus r_2 \oplus r_3$	4,5
10	$x \oplus r_1 \oplus r_2$	0,5
11	$x \oplus r_1 \oplus r_2 \oplus a$	5
12	$S(x \oplus r_1 \oplus r_2 \oplus a)$	5
13	$S(x \oplus r_1 \oplus r_2 \oplus a) \oplus s_1$	5
14	$S(x \oplus r_1 \oplus r_2 \oplus a) \oplus s_1 \oplus s_2$	5
15	$S(x) \oplus s_1 \oplus s_2$	6

To prove the independence between $E_2 \times I_{15}$ and x, we split E_2 into two subsets: $I_{10} \cup \cdots \cup I_{13}$ and I_{14}. One can check that $(x \oplus r_1 \oplus r_2, S(x) \oplus s_2)$ is independent of x and that every element of $(I_{10} \cup \cdots \cup I_{13}) \times I_{15}$ can be expressed as a function of this pair. Hence one deduces from Lemma 8.1 that $(I_{10} \cup \cdots \cup I_{13}) \times I_{15}$ is independent of x. In order to prove that $I_{14} \times I_{15}$ is also independent of x, let us denote $u_1 = S(x) \oplus s_1 \oplus s_2$ and $u_2 = S(x \oplus a \oplus r_1 \oplus r_2)$. The variables u_1 and u_2 are uniformly distributed[17], independent and jointly independent of x. Since $I_{14} \times I_{15}$ equals $\{S(x) \oplus u_2 \oplus u_1\} \times \{u_1\}$, we deduce that it is independent of x. □

10.3.2 Second scheme

In this section, we propose an alternative to Algorithm 9 for implementing an S-box securely against second-order SCA. This second solution requires more logical operations but less RAM allocation, which can be of interest for low-cost devices.

The algorithm introduced hereafter assumes the existence of a masked function $\mathsf{compare}_b$ that extends the classical Boolean function (defined by $\mathsf{compare}(x, y) = 0$ iff $x = y$) in the following way:

$$\mathsf{compare}_b(x, y) = \begin{cases} b & \text{if } x = y \\ \bar{b} & \text{if } x \neq y \end{cases} . \tag{10.1}$$

Based on the function above, the second scheme is an adaptation of the first-order scheme of Section 9.3.

Algorithm 10 SecSbox-2O-RDP2

INPUT: a masked value $\tilde{x} = x \oplus r_1 \oplus r_2 \in \mathbb{F}_2^n$, the pair of input masks $(r_1, r_2) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, a pair of output masks $(s_1, s_2) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$, a look-up table for S
OUTPUT: the masked S-box output $S(x) \oplus s_1 \oplus s_2 \in \mathbb{F}^m$

1. $b \leftarrow \mathsf{rand}(1)$
2. **for** $a = 0$ **to** $2^n - 1$ **do**
3. $\quad cmp \leftarrow \mathsf{compare}_b(r_1 \oplus a, r_2)$
4. $\quad R_{cmp} \leftarrow (S(\tilde{x} \oplus a) \oplus s_1) \oplus s_2$
5. **return** R_b

[17] This holds for u_2 if and only if the S-box S is balanced (namely every element in $\mathbb{F}_2^m$ is the image under S of 2^{n-m} elements in $\mathbb{F}_2^n$). As it is always true for cryptographic S-boxes we implicitly make this assumption.

Let *dummy* denote any element in $\mathbb{F}_2^m$. Steps 3 and 4 of Algorithm 10 perform the following operations:

$$\begin{cases} cmp \leftarrow b\ ; & R_b \leftarrow S(x) \oplus s_1 \oplus s_2 & \text{if } a = r_1 \oplus r_2\ , \\ cmp \leftarrow \bar{b}\ ; & R_{\bar{b}} \leftarrow dummy & \text{otherwise.} \end{cases}$$

We thus deduce that the value returned by Algorithm 10 is $S(x) \oplus s_1 \oplus s_2$.

10.3.2.1 An implementation of $\mathsf{compare}_b$

In this section, we describe a way to implement the function $\mathsf{compare}_b$ that prevents any first-order leakage on $\mathsf{compare}(x, y)$.

The method requires a table T of 2^n bits in RAM. The following pre-computation is processed at the beginning of the cipher.

Algorithm 11 Pre-computation for the $\mathsf{compare}_b$ function

INPUT: a bit b
OUTPUT: a random $r_3 \in \mathbb{F}^n$ and a (2^n)-bit table T s.t. $T[x] = b$ if $x = r_3$ and $T[x] = \bar{b}$ otherwise

1. $r_3 \leftarrow \mathsf{rand}(n)$
2. $T \leftarrow \{\bar{b}, \bar{b}, \ldots, \bar{b}\}$
3. $T[r_3] \leftarrow b$

Then, the function $\mathsf{compare}_b$ is implemented by a simple table look-up:

$$\mathsf{compare}_b(x, y) = T[(x \oplus r_3) \oplus y]$$

If $x = y$ then $\mathsf{compare}_b(x, y) = T[r_3]$ otherwise $\mathsf{compare}_b(x, y) = \bar{b}$, which shows the correctness of the solution.

10.3.2.2 Complexity

Algorithm 10 involves 4×2^n XOR operations, 2^n memory transfers and the generation of 1 random bit. Since it also involves 2^n $\mathsf{compare}_b$ operations, the overall complexity relies on the $\mathsf{compare}_b$ implementation. When $\mathsf{compare}_b$ is implemented such as described above, each call to $\mathsf{compare}_b$ involves 2 XOR operations and 1 memory transfer. Then the total complexity of Algorithm 10 is 6×2^n XOR operations, 2×2^n memory transfers. Moreover, it requires some pre-computation (at the beginning of the cipher) that involves the generation of n random bits and the initialization of the (2^n)-bit table T.

Compared to Algorithm 9, this implies a slight timing overhead but it consumes m times less RAM which is of interest for low-cost devices.

10.3.2.3 Security analysis

Let δ_0 denote the Boolean function defined by $\delta_0(z) = 0$ if and only if $z = 0$. For security reasons, $\mathsf{compare}_b(x, y)$ must prevent any first-order leakage on $\delta_0(x \oplus y)$ that is, on the result of the unmasked function $\mathsf{compare}(x, y)$ (and more generally on $x \oplus y$). Otherwise, Step 3 would provide a first-order leakage on $\delta_0(r_1 \oplus r_2 \oplus a)$ and an attacker could target this leakage together with $\tilde{x} \oplus a$ (Step 4) to recover information about x. Indeed, the joint distribution of $\delta_0(r_1 \oplus r_2 \oplus a)$ and $\tilde{x} \oplus a$ depends on x, which can be illustrated by the following observation: $\tilde{x} \oplus a = x$ if and only if $\delta_0(r_1 \oplus r_2 \oplus a) = 0$. In particular, the straightforward implementation $\mathsf{compare}_b(x, y) = \mathsf{compare}(x, y) \oplus b$ would not be valid since it processes $\mathsf{compare}(x, y)$ directly.

When $\mathsf{compare}_b$ is implemented such as described in Section 10.3.2.1, no first-order leakage occurs on $\delta_0(x \oplus y)$ nor on $x \oplus y$. Indeed, all the intermediate variables processed during the table pre-computation and the table look-ups (namely $\{b, \bar{b}, r_3, x \oplus r_3, x \oplus y \oplus r_3, \mathsf{compare}_b(x, y)\}$) are independent of $\mathsf{compare}(x, y)$ (as well as of $x \oplus y$).

Table 10.2: Intermediate variables of Algorithm 10.

j	I_j	Steps
1	r_1	0,3
2	r_2	0,3
3	s_1	0,4
4	s_2	0,4
6	b	1,3
7	a	2-4
8	$r_1 \oplus a$	3
10	$\delta_0(a \oplus r_1 \oplus r_2) \oplus b$	3
11	$x \oplus r_1 \oplus r_2$	0,4
12	$x \oplus r_1 \oplus r_2 \oplus a$	4
13	$S(x \oplus r_1 \oplus r_2 \oplus a)$	4
14	$S(x \oplus r_1 \oplus r_2 \oplus a) \oplus s_1$	4
15	$S(x \oplus r_1 \oplus r_2 \oplus a) \oplus s_1 \oplus s_2$	4
16	$S(x) \oplus s_1 \oplus s_2$	5

Remark 10.3. The previous assertion implicitly assumes that the look-up $T[(x \oplus r_3) \oplus y]$ does not read several bits of T simultaneously. Otherwise a potential flaw would occur. In practice, T could be placed in a bit-addressable memory or each byte of T could be further masked in order to avoid a joint leakage on the different bits resulting from a look-up.

Remark 10.4. The solution described in Section 10.3.2.1 for the $\mathsf{compare}_b$ function implies that the same random values b and r_3 are used in every SecSbox computation of the cipher. It is easy to check that this does not induce any second-order flaw in the overall scheme.

Security Proof. As done in Section 10.3.1, we denote by $\mathcal{I}$ the set of intermediate variables that are processed during an execution of Algorithm 10. Table 10.2 lists these variables. The primitive random values r_1, r_2, s_1, s_2 and b are assumed to be uniformly distributed and mutually independent together with the sensitive variable x. The following security proof is quite similar to the one done in Section 10.3.1.

In order to prove that Algorithm 10 is secure against second-order SCA, we need to show that $\mathcal{I} \times \mathcal{I}$ is independent of x. As in Section 10.3.1 we split $\mathcal{I}$ into three subsets $E_1 = I_1 \cup \cdots \cup I_{10}$, $E_2 = I_{11} \cup \cdots \cup I_{15}$ and I_{16}. First, we show that $E_1 \times E_1$, $E_2 \times E_2$ and $I_{16} \times I_{16}$ are independent of x and then, we show that $E_1 \times E_2$, $E_1 \times I_{16}$ and $E_2 \times I_{16}$ are independent of x (thus proving that $\mathcal{I} \times \mathcal{I}$ is independent of x).

As in Section 10.3.1, $E_1 \times E_1$ is straightforwardly independent of x and the independence between $x \oplus r_1 \oplus r_2$ (resp. $S(x) \oplus s_1 \oplus s_2$) and x implies, by Lemma 8.1, that $E_2 \times E_2$ (resp. $I_{16} \times I_{16}$) is independent of x.

Since E_1 is independent of $r_1 \oplus r_2$ (resp. $s_1 \oplus s_2$) and functionally independent of x, Lemma 8.2 implies that $E_1 \times \{x \oplus r_1 \oplus r_2\}$ (resp. $E_1 \times \{S(x) \oplus s_1 \oplus s_2\}$) is independent of x. Hence, since every element of E_2 (resp. I_{16}) can be written as a function of $x \oplus r_1 \oplus r_2$ (resp. $S(x) \oplus s_1 \oplus s_2$), Lemma 8.1 implies that $E_1 \times E_2$ (resp. $E_1 \times I_{16}$) is independent of x.

Every pair in $(E_2 \backslash I_{15}) \times I_{16}$ can be expressed as a function of $(x \oplus r_1 \oplus r_2, S(x) \oplus s_2)$ which is independent of x. Hence, by Lemma 8.1, $(E_2 \backslash I_{15}) \times I_{16}$ is independent of x. Finally, $I_{15} \times I_{16}$ can be rewritten as $\{S(x) \oplus u_2 \oplus u_1\} \times \{u_1\}$, where u_1 $(= S(x) \oplus s_1 \oplus s_2)$ and u_2 $(= S(x \oplus r_1 \oplus r_2 \oplus a))$ are uniformly distributed, mutually independent and jointly independent of x. This implies that $I_{15} \times I_{16}$ is independent of x. □

10.3.3 Improvement

This section describes an improvement of the two previous methods which can be used when the device architecture allows the storage of 2^w S-box outputs on one q-bit word (namely m, w and q satisfy

$2^w m \leqslant q$). This situation may happen for 8-bit architectures when the S-boxes to implement have small output dimensions (*e.g.* $m = 4$ and $w = 1$) or for q-bit architectures when $q \geqslant 16$ (and $m \leqslant 8$).

In the following, we assume that the S-box is represented by a look-up table having 2^{n-w} elements of bit-length $2^w m$ (instead of 2^n elements of bit-length m). This look-up table can then be seen as the table representation of the $(n-w, 2^w m)$-function S' defined for every $y \in \mathbb{F}^{n-w}$ by: $S'(y) = (S(y,0), S(y,1), \ldots, S(y,2^w-1))$, where each $i = 0, \ldots, 2^w - 1$ must be taken as the integer representation of a w-bit value.

For every $x \in \mathbb{F}_2^n$, let us denote by $x[i]$ the i^{th} most significant bit of x and by x_H (resp. x_L) the vector $(x[1], \ldots, x[n-w])$ (resp. the vector $(x[n-w+1], \ldots, x[n])$). According to these notations, the S-box output $S(x)$ is the m-bit coordinate of $S'(x_H)$ whose index is the integer representation of x_L.

In order to securely compute the masked output $S(x) \oplus s_1 \oplus s_2$ from the 3-tuple $(\tilde{x}, r_1, r_2)$, our improvement consists of the two following steps. In the first step, we securely compute the masked vector $S'(x_H) \oplus z_1 \oplus z_2$ (where z_1 and z_2 are $(2^w m)$-bit random masks). Then, the second step consists in securely extracting $S(x) \oplus s_1 \oplus s_2$ from $S'(x_H) \oplus z_1 \oplus z_2$.

To securely compute the masked vector $S'(x_H) \oplus z_1 \oplus z_2$, we perform Algorithm 9 (or 10) for the pair of dimensions $(n-w, 2^w m)$, with as inputs the 3-tuple $(\tilde{x}_H, r_{1,H}, r_{2,H})$, the pair of output masks (z_1, z_2) and the look-up table for S'. This execution returns the value $S'(x_H) \oplus z_1 \oplus z_2$. Moreover, as proved in Section 10.3.1 (or Section 10.3.2), it is secure against second-order SCA.

At this point, we need to securely extract $S(x) \oplus s_1 \oplus s_2$ from $S'(x_H) \oplus z_1 \oplus z_2$ as well as s_1 and s_2 from z_1 and z_2. Namely, we need to extract the m-bit coordinate of $S'(x_H) \oplus z_1 \oplus z_2$, and of z_1 and z_2 whose index corresponds to the integer representation of x_L. For such a purpose, we propose a process that selects the desired coordinate by dichotomy.

For every word y of even bit-length, let $H_0(y)$ and $H_1(y)$ denote the most and the least significant half part of y. At each iteration our process calls an algorithm select that takes as inputs a dimension l, a second-order masked $(2l)$-bit word $z_0 = z \oplus z_1 \oplus z_2$ (and the corresponding masking words z_1 and z_2) and a second-order masked bit $c_0 = c \oplus c_1 \oplus c_2$ (and the corresponding masking bits c_1 and c_2). This algorithm returns a 3-tuple of l-bit words (z'_0, z'_1, z'_2) that satisfies $z'_0 \oplus z'_1 \oplus z'_2 = H_c(z)$. We detail hereafter the global process which extracts the 3-tuple $(S(x) \oplus s_1 \oplus s_2, s_1, s_2)$ from $(S'(x_H) \oplus z_1 \oplus z_2, z_1, z_2)$.

Algorithm 12 Masked sub-word extraction

INPUT: the masked word $S'(x_H) \oplus z_1 \oplus z_2$, the masks z_1 and z_2
OUTPUT: the masked sub-word $S(x) \oplus s_1 \oplus s_2$, the sub-word masks s_1 and s_2

1. $z_0 \leftarrow S'(x_H) \oplus z_1 \oplus z_2$
2. **for** $i = 1$ **to** w
3. $\quad (c_0, c_1, c_2) \leftarrow (\tilde{x}_L[i], r_{1,L}[i], r_{2,L}[i])$
4. $\quad (z'_0, z'_1, z'_2) \leftarrow \mathsf{select}\big(2^w m/2^i, (z_0, z_1, z_2), (c_0, c_1, c_2)\big)$
5. $\quad (z_0, z_1, z_2) \leftarrow (z'_0, z'_1, z'_2)$
6. **return** (z_0, z_1, z_2)

To be secure against second-order SCA, this process requires that select admits no second-order leakage on z nor on c. A solution for such a secure algorithm is given hereafter (Algorithm 13). It requires three l-bit addressing registers (A_0, A_1), (B_0, B_1) and (C_0, C_1).

Algorithm 13 select

INPUT: a dimension l, a masked word $z_0 = z \oplus z_1 \oplus z_2 \in \mathbb{F}^{2l}$, the pair of masks $(z_1, z_2) \in \mathbb{F}^{2l} \times \mathbb{F}^{2l}$, a masked bit $c_0 = c \oplus c_1 \oplus c_2 \in \mathbb{F}$ and the pair of masking bits $(c_1, c_2) \in \mathbb{F} \times \mathbb{F}$
OUTPUT: a 3-tuple $(z'_0, z'_1, z'_2) \in (\mathbb{F}^l)^3$ that satisfies $z'_0 \oplus z'_1 \oplus z'_2 = H_c(z)$

1. $t_1, t_2 \leftarrow (\mathsf{rand}(l), \mathsf{rand}(l))$
2. $b \leftarrow \mathsf{rand}(1)$
3. $c_3 \leftarrow (c_1 \oplus b) \oplus c_2$
4. $A_{c_3} \leftarrow H_{c_0}(z_0) \oplus t_1$
5. $B_{c_3} \leftarrow H_{c_0}(z_1) \oplus t_2$

6. $C_{c_3} \leftarrow H_{c_0}(z_2) \oplus t_1 \oplus t_2$
7. $A_{\overline{c_3}} \leftarrow H_{\overline{c_0}}(z_0) \oplus t_1$
8. $B_{\overline{c_3}} \leftarrow H_{\overline{c_0}}(z_1) \oplus t_2$
9. $C_{\overline{c_3}} \leftarrow H_{\overline{c_0}}(z_2) \oplus t_1 \oplus t_2$
10. **return** (A_b, B_b, C_b)

One can verify that Algorithm 13 performs the following operations for every value of (c_1, c_2):

$$\begin{cases} (A_b, B_b, C_b) \leftarrow (H_c(z_0) \oplus t_1, H_c(z_1) \oplus t_2, H_c(z_2) \oplus t_1 \oplus t_2) \\ (A_{\overline{b}}, B_{\overline{b}}, C_{\overline{b}}) \leftarrow (H_{\overline{c}}(z_0) \oplus t_1, H_{\overline{c}}(z_1) \oplus t_2, H_{\overline{c}}(z_2) \oplus t_1 \oplus t_2) \end{cases}.$$

Thus the three returned variables satisfy $A_b \oplus B_b \oplus C_b = H_c(z)$.

Remark 10.5. Although it has been described for second-order masking, this improvement could be used for first-order masking and in particular with the method proposed in Chapter 9.

10.3.3.1 Complexity

Algorithm 13 involves 10 XOR operations and the generation of $2l + 1$ random bits.

The improvement allows to divide the execution time of Algorithm 9 (or 10) by approximately 2^w since it performs a loop of 2^{n-w} iterations instead of 2^n. Additionally, the improvement involves w calls to Algorithm 13 which implies an overhead of approximately $10 \times w$ XOR operations and the generation of $2m \times (2^w - 1) + w$ random bits.

*Example.*For an 8×8 S-box on a 16-bit architecture, the improvement applied to Algorithm 9 allows to save 512 XOR operations and 128 memory transfers for an overhead of 10 XOR operations and the generation of 33 random bits (16 more for (z_1, z_2) than for (s_1, s_2) and $16 + 1$ for Algorithm 13).

10.3.3.2 Security analysis

The random values t_1 and t_2 are introduced to avoid any second-order leakage on c. Otherwise, if the algorithm simply returns $(H_c(z_0), H_c(z_1), H_c(z_2))$, an inherent second-order flaw (*i.e.* independent of the algorithm operations) occurs. Indeed, by targeting one of the inputs z_i and one of the outputs $H_c(z_i)$, an attacker may recover some information on c since $\big(z_i, H_c(z_i)\big)$ depends on c (even if z_i is random).

Security Proof. Table 10.3 lists all the intermediate variables that are processed by Algorithm 13. To prove that Algorithm 13 is secure against second-order SCA, we must show that $\mathcal{I} \times \mathcal{I}$ is independent of z and of c.

The independence between $\mathcal{I} \times \mathcal{I}$ and z is straightforward. Indeed, from Table 10.3, one can verify that $z \oplus z_1 \oplus z_2$, z_1 and z_2 are always processed separately.

Now, let us show that $\mathcal{I} \times \mathcal{I}$ is also independent of c. For such a purpose, we split $\mathcal{I}$ into the three subsets $E_1 = I_1 \cup \cdots \cup I_{10}$, $E_2 = I_{11} \cup \cdots \cup I_{26}$ and $E_3 = I_{27} \cup I_{28} \cup I_{29}$. As done in Sections 10.3.1 and 10.3.2, we first show that $E_i \times E_j$ is independent of x for $i = j$ and then for $i < j$.

E_1 is functionally independent of c which implies that $E_1 \times E_1$ is independent of c. Since $c \oplus c_1 \oplus c_2$ is independent of c and since every element of $E_2 \times E_2$ is a function of $c \oplus c_1 \oplus c_2$, Lemma 8.1 implies that $E_2 \times E_2$ is independent of c. On the other hand, each element of $E_3 \times E_3$ can be rewritten $(H_c[u_1] \oplus v_1, H_c[u_1] \oplus v_1)$ or $(H_c[u_1] \oplus v_1, H_c[u_2] \oplus v_2)$ where u_1, u_2 v_1 and v_2 are uniformly distributed random variables that are mutually independent and jointly independent of c. This implies (by Lemma 8.2) that $E_3 \times E_3$ is independent of c.

One can verify that E_1 is independent of $c_1 \oplus c_2$. Moreover E_1 is functionally independent of c. This implies, according to Lemma 8.2, that $E_1 \times \{c \oplus c_1 \oplus c_2\}$ is independent of c. Moreover, since every element in E_2 is a function of $c \oplus c_1 \oplus c_2$ Lemma 8.1 implies that $E_1 \times E_2$ is independent of c.

To prove that $E_1 \times E_3$ is independent of x, we split E_1 into two subsets: $E_1' = E_1 \setminus (I_6 \cup I_7)$ and $I_6 \cup I_7$. The set E_1' is functionally independent of c and is independent of t_1, t_2 and $t_1 \oplus t_2$. Since every element of E_3 is a function of c that is masked either by t_1, or t_2, or $t_1 \oplus t_2$, then, Lemma 8.2 implies

Table 10.3: Intermediate variables of Algorithm 13.

j	I_j	Steps
1	$z \oplus z_1 \oplus z_2$	0-4-7
2	z_1	0,5,8
3	z_2	0,6,9
4	c_1	0,3
5	c_2	0,3
6	t_1	1,4,6,7,9
7	t_2	1,5,6,8,9
8	b	2-3-10
9	$c_1 \oplus b$	3
10	$c_1 \oplus c_2 \oplus b$	3-9
11	$c \oplus c_1 \oplus c_2$	0,4-6
12	$H_{c\oplus c_1\oplus c_2}(z \oplus z_1 \oplus z_2)$	4
13	$H_{c\oplus c_1\oplus c_2}(z \oplus z_1 \oplus z_2) \oplus t_1$	4
14	$H_{c\oplus c_1\oplus c_2}(z_1)$	5
15	$H_{c\oplus c_1\oplus c_2}(z_1) \oplus t_2$	5
16	$H_{c\oplus c_1\oplus c_2}(z_2)$	6
17	$H_{c\oplus c_1\oplus c_2}(z_2) \oplus t_1$	6
18	$H_{c\oplus c_1\oplus c_2}(z_2) \oplus t_1 \oplus t_2$	6
19	$H_{\overline{c\oplus c_1\oplus c_2}}(z \oplus z_1 \oplus z_2)$	7
20	$H_{\overline{c\oplus c_1\oplus c_2}}(z \oplus z_1 \oplus z_2) \oplus t_1$	7
21	$H_{\overline{c\oplus c_1\oplus c_2}}(z_1)$	8
22	$H_{\overline{c\oplus c_1\oplus c_2}}(z_1) \oplus t_2$	8
23	$H_{\overline{c\oplus c_1\oplus c_2}}(z_2)$	9
25	$H_{\overline{c\oplus c_1\oplus c_2}}(z_2) \oplus t_1$	9
26	$H_{\overline{c\oplus c_1\oplus c_2}}(z_2) \oplus t_1 \oplus t_2$	9
27	$H_c(z \oplus z_1 \oplus z_2) \oplus t_1$	10
28	$H_c(z_1) \oplus t_2$	10
29	$H_c(z_2) \oplus t_1 \oplus t_2$	10

that $E_1' \times E_3$ is independent of c. On the other hand, every element of E_3 is a function of $H_c(z_i)$ for $i \in \{0,1,2\}$ (recalling $z_0 = z \oplus z_1 \oplus z_2$). Since z_i is uniformly distributed and independent of c, $H_c(z_i)$ is independent of c. And since $I_6 \cup I_7$ is functionally independent of z, z_1, z_2 and c, we can deduce that $(I_6 \cup I_7) \times E_3$ is independent of c.

To prove that $E_2 \times E_3$ is independent of x, we split E_2 into two subsets: I_{11} and $E_2' = E_2 \backslash I_{11}$. Every element of $\mathcal{I}_{11} \times E_3$ is a pair $(c, H_c(z_i))$ that is masked with an independent and uniformly distributed pair $(c_1 \oplus c_2, t)$ (where t is in $\{t_1, t_2, t_1 \oplus t_2\}$). This implies (by Lemma 8.2) that $I_{11} \times E_3$ is independent of c. On the other hand, every element of $E_2' \times E_3$ is a function of a pair $(H_{c_0}(u_1), H_c(u_2))$ (or $(H_{\overline{c_0}}(u_1), H_c(u_2))$) where c_0 equals $c \oplus c_1 \oplus c_2$ and where u_1 and u_2 are two random variables (possibly equal) both uniformly distributed and independent of c and c_0. Then, since c and c_0 are independent, one can verify that $(H_{c_0}(u_1), H_c(u_2))$ (and $(H_{\overline{c_0}}(u_1), H_c(u_2))$) is independent of c, thus implying, by Lemma 8.1, that $E_2' \times E_3$ is independent of c. □

10.4 Comparison and application

The purpose of this section is twofold. First, we compare the complexity of the Schramm and Paar's scheme with the one of our scheme. Secondly, we present several implementations of the AES protected as described in Section 8.5.2 and where the SecSbox algorithm is implemented either with the Schramm and Paar's schemes or with ours (Algorithm 9). We also implemented the different SecSbox algorithms on 8-bit, 16-bit and 32-bit architectures. To demonstrate the practical interest of the improvement proposed in Section 10.3.3, we implemented the improved version of Algorithm 9 for the 16-bit and 32-bit architectures.

10.4.1 Comparison to Schramm and Paar's scheme

Table 10.4 summarizes the complexities of the Schramm and Paar's schemes and ours with respect to the number of XOR, the number of memory transfers (MT) and the memory size (bits in RAM). N_S stands for the number of different S-boxes involved in the cipher.

Table 10.4: Time & memory complexities of the generic second-order masking schemes for S-boxes.

Method	Pre-computation	S-box computation	RAM
Algo. 7	0	$(4 \times 2^n + 1)\texttt{MT} + 4 \times 2^n\texttt{XOR}$	$2m \times 2^n$
Algo. 8	$2 \times 2^n\texttt{MT} + 2 \times 2^n\texttt{XOR}$	$(2 \times 2^n + 1)\texttt{MT} + (2 \times 2^n + 6)\texttt{XOR}$	$(1 + N_S) \times m \times 2^n$
Algo. 9	0	$(2 \times 2^n + 1)\texttt{MT} + (4 \times 2^n + 2)\texttt{XOR}$	$m \times 2^n$
Algo. 10	$(2 \times 2^n + 1)\texttt{MT}$	$2^n\texttt{MT} + 6 \times 2^n\texttt{XOR}$	2^n

Considering that the execution timings of an XOR and of a memory transfer are equivalent, the Schramm and Paar's generic scheme (Algorithm 7) is clearly less interesting than our schemes. The Schramm and Paar's improved scheme (Algorithm 8) is almost 1.5 times faster than our first scheme and 1.75 times faster than our second scheme (when the latter uses the $\mathsf{compare}_b$ function given in Section 10.3.2.1). However, the Schramm and Paar's improved scheme requires the allocation of $(1+N_S) \times m \times 2^n$ bits of RAM, whereas our schemes respectively require $2^n m$ and 2^n bits of RAM whatever the number of involved S-boxes. RAM memory being a sensitive resource in low-cost devices, the memory gain provided by Algorithms 9 and 10 can often be of great interest even if it is mitigated by a timing overhead. This is especially true when the S-box input dimension is high (namely greater than or equal to 8) and/or when the number of S-box(es) to protect is high (as it is for instance the case for DES).

10.4.2 Application to AES

We compare hereafter several AES implementations protected against second-order DPA (*i.e.* we do not mask the key and we do not protect the key scheduling function). We wrote the codes in assembly

language for an 8051-based 8-bit architecture. The implementations only differ in their approaches to protect the S-box computations. The linear steps of the AES have been implemented in the same way, by following the outlines of the method presented in Section 8.5.2. To secure the S-box computation, we implemented the generic and the improved Schramm and Paar's schemes and the scheme that is the fastest among our two new ones (namely Algorithm 9). Table 10.5 lists the timing and memory performances of each implementation.

Table 10.5: Comparison of AES implementations secure against second-order DPA.

Method	Reference	cycles	RAM (bytes)	ROM (bytes)
AES with Algorithm 7	SP1	10830×10^3	$512 + 86$	2247
AES with Algorithm 8	SP2	5943×10^3	$512 + 90$	2336
AES with Algorithm 9	RDP	6723×10^3	$256 + 86$	2215

As expected, implementations SP2 (Schramm and Paar's improved scheme) and RDP (our scheme) are much more efficient than the SP1 (Schramm and Paar's generic scheme) which performs two table re-computations for each S-box computation. The SP2 implementation is slightly faster than RDP (approximately 1.13 times faster), essentially because it involves less logical operations (as shown in the complexity analysis conducted in Sections 10.3.1 and 10.4.1). However, our scheme (RDP) only requires the allocation of 256 bytes of RAM, which is twice smaller than the RAM memory used by SP1 and SP2. Since RAM memory is a sensitive resource in the area of embedded devices and since the timing performances of SP2 and RDP are close, our tests show that our scheme represents a good alternative to the proposal of Kai Schramm and Christof Paar in applications where memory constraints are strong.

10.4.3 Implementation of the improvement

To demonstrate the practical interest of the improvement proposed in Section 10.3.3, we implemented the generic and the improved Schramm and Paar's schemes and our proposal (improved or not) on a 16-bit architecture with a proprietary assembly language and on a 32-bit ARM architecture.

Table 10.6: Comparison of 8×8 S-box implementations secure against second-order DPA on 8-bit, 16-bit and 32-bit architectures.

Method	Reference	Cycles	RAM (bytes)	ROM (bytes)
8-bit architecture				
Algorithm 7	SP1-8	6703	$512 + 3$	$119 + 256$
Algorithm 8	SP2-8	3638	$512 + 7$	$89 + 256$
Algorithm 9	RDP-8	4142	$256 + 3$	$88 + 256$
16-bit architecture				
Algorithm 7	SP1-16	6418	512	$96 + 512$
Algorithm 8	SP2-16	3090	512	$56 + 256$
Algorithm 9	RDP-16	4125	256	$98 + 512$
Algorithm 9 + Improvement	RDP*-16	2099	256	$260 + 256$
32-bit architecture				
Algorithm 7	SP2-32	3359	512	na.
Algorithm 8	RDP-32	4143	256	na.
Algorithm 9 + Improvement	RDP*-32	1415	256	na.

Remark 10.6. In our implementations of SP1-16 and RDP-16, we represented each element of the 8×8 S-box by a 16-bit word whose LSB is the S-box element and whose MSB is the zero element. This representation multiplies by a factor of two the size of the look-up table in ROM, but avoids the conversions

of 16-bit words into 8-bit words during the loop execution (thus speeding up the entire S-box calculation by approximately 1.25).

In all the cases, the implementation of Algorithm 8 is more efficient than the implementation of Algorithm 9. On average, it is 1.20 times faster. The use of the improvement (Algorithm 13) allows a gain of 50% for the 16-bit architecture and of 65% for the 32-bit architecture. With this improvement our method becomes much faster than SP2. For the implementation on the ARM 32-bit architecture, it may be noticed that the gain is smaller than the one resulting from our theoretical complexity analysis (Section 10.3.3). This is merely due to the fact that the assembly implementation of Algorithm 13 involves costly registers and data pointers manipulations.

Chapter 11

A generic scheme combining higher-order masking and shuffling

Contents

11.1 Introduction

In the previous chapters, we have studied the masking countermeasure for implementations of block ciphers. When a block cipher is implemented in software, another countermeasure is commonly used: operations shuffling. The principle is to perform, as far as possible, the different operations of the computation in a random order. Shuffling can be enhanced by the addition of dummy operations to increase the degree of randomization. As a result of shuffling, a side channel attacker is provided with a set of leakages resulting from several operations and he cannot *a priori* isolate the leakage corresponding to the target operation.

A natural approach to improve the SCA resistance of software implementations of block ciphers is to mix masking and shuffling [124, 228, 229]. This approach seems promising since it enables to get the best of the two techniques. However, the schemes that have been proposed so far [124, 229] only focus on first-order masking which prevents them from reaching high resistance levels. This is all the more serious that advanced DPA attacks have turned out to be quite efficient in breaking them [228, 229]. These advanced DPA techniques have been studied and extended in Chapter 6 of this thesis. Based on this analysis, we design here a new scheme combining higher-order masking and shuffling to protect software implementations of block ciphers. Contrary to the previous chapters, this scheme does not provide perfect security against d^{th}-order SCA for a given order d. Rather it ensures a given resistance level with respect to some advanced DPA techniques. Besides, our scheme is scalable and its parameters can be specified to achieve any desired resistance level. We apply it to protect a software implementation of AES and we show how to choose the scheme parameters to achieve a given security level with the minimum overhead.

The results presented in this chapter have been published in collaboration with Emmanuel Prouff and Julien Doget in the international workshop on *Cryptographic Hardware and Embedded Systems (CHES 2009)* [11].

11.2 Block cipher model

We give hereafter the general model of block ciphers on which our scheme applies. We use the model introduced in Section 8.5.1 with further precisions that shall be useful for the description of our scheme.

We recall that the round transformation is modeled as the composition of a key addition, a non-linear layer γ, and a linear layer λ:

$$\varphi[k](\cdot) = \lambda \circ \gamma(\cdot \oplus k) \ .$$

Compared to the block cipher model of Section 8.5.1, we shall make two further assumptions:

— The non-linear layer applies the same S-box S on N independent n-bit parts p_i of the state: $\gamma(p) = \big(S(p_1), \ldots, S(p_N)\big)$.

— The linear layer λ is composed of L linear operations λ_i that operate on L independent l-bit parts $p_{i(l)}$ of the state: $\lambda(p) = \big(\lambda_1(p_{1(l)}), \ldots, \lambda_L(p_{L(l)})\big)$. We also denote by $l' \leqslant l$ the minimum number of bits of a variable manipulated during the processing of λ_i. For instance, the MixColumns layer of AES applies to columns of $l = 32$ bits but it manipulates some elements of $l' = 8$ bits. We further assume that the λ_i's are sufficiently similar to be implemented by one atomic operation (*i.e.* an operation which has the same execution flow whatever the index i).

Remark 11.1. Linear and non-linear layers may involve different state indexing. In AES for instance, the state is usually represented as a 4×4 matrix of bytes and the non-linear layer usually operates on its elements $p_1, \ldots, p_{16}$ vertically (starting at the top) and from left to right. In this case, the operation λ_1 corresponding to the AES linear layer (that is composed of ShiftRows followed by MixColumns [92]) operates on $p_{1(32)} = (p_1, p_6, p_{11}, p_{16})$.

In the sequel, we shall consider that the key addition and the non-linear layer are merged in a *keyed substitution layer* that adds each key part k_i to the corresponding state part p_i before applying the S-box S.

11.3 The scheme

In this section, we describe a generic scheme to protect a round φ by combining higher-order masking and operations shuffling. Our scheme involves a d^{th}-order masking for an arbitrarily chosen d. Namely, the state p is split into $d+1$ shares $m_0, \ldots, m_d$ satisfying:

$$m_0 \oplus \cdots \oplus m_d = p \ . \tag{11.1}$$

In practice, $m_1, \ldots, m_d$ are random masks and m_0 is the masked state defined according to (11.1). In the sequel, we shall denote by $(m_j)_i$ (resp. $(m_j)_{i(l)}$) the i^{th} n-bit part (resp. the i^{th} l-bit part) of a share m_j. At the beginning of the ciphering the masks are initialized to zero. Then, each time a part of a mask is used during the keyed substitution layer computation, it is refreshed with a new random value (see below). Our scheme uses two different approaches to protect the keyed substitution layer and the linear layer. These are described hereafter.

11.3.1 Protecting the keyed substitution layer

To protect the keyed substitution layer, we use a single d'^{th}-order masked S-box (for some $d' \leqslant d$) to perform all the S-box computations. Whatever the number of masks, a second-order side channel attack targeting two different masked inputs/outputs is always possible (see for instance [176]). To deal with this issue, we make use of a high level of shuffling in order to render such an attack difficult and to keep a homogeneous security level (see Section 11.5).

The input of S is masked with d' masks $r_1, \ldots, r_{d'}$ and its output is masked with d' masks $s_1, \ldots, s_{d'}$. Namely, a masked S-box S^* is computed that is defined for every $x \in \{0,1\}^n$ by:

$$S^*(x) = S\Big(x \oplus \bigoplus_{j=1}^{d'} r_j\Big) \oplus \bigoplus_{j=1}^{d'} s_j \ . \tag{11.2}$$

This masked S-box is then involved to perform all the S-box computations. Namely, when the S-box must be applied to a masked variable $(m_0)_i$, the d masks $(m_j)_i$ of this latter are replaced by the d' masks r_j which enables the application of S^*. The d' masks s_j of the obtained masked output are then switched for d new random masks $(m_j)_i$.

The high level shuffling is ensured by the addition of dummy operations. Namely, the S-box computation is performed t times: N times on a relevant part of the state and $t-N$ times on dummy data. For such a purpose, each share m_j is extended by a dummy part $(m_j)_{N+1}$ that is initialized by a random value at the beginning of the ciphering. The round key k is also extended by such a dummy part k_{N+1}. For each of the t S-box computations, the index i of the parts $(m_j)_i$ to process is read in a table T. This table of size t contains all the indices from 1 to N stored at random positions and its $t-N$ other elements equal $N+1$. Thanks to this table, the S-box computation is performed once on every of the N relevant parts and $t-N$ times on the dummy parts. The following algorithm describes the entire protected keyed substitution layer computation.

Algorithm 14 Protected keyed substitution layer

INPUT: the shares $m_0, \ldots, m_d$ s.t. $\bigoplus m_i = p$, the round key $k = (k_1, \ldots, k_{N+1})$
OUTPUT: the shares $m_0, \ldots, m_d$ s.t. $\bigoplus m_i = \gamma(p \oplus k)$

1. **for** $i_T = 1$ **to** t

// Random index pick-up
2. $i \leftarrow T[i_T]$

// Masks conversion: $(m_0)_i \Leftarrow p_i \bigoplus_j r_j$
3. **for** $j = 1$ **to** d' **do** $(m_0)_i \leftarrow ((m_0)_i \oplus r_j) \oplus (m_j)_i$
4. **for** $j = d'+1$ **to** d **do** $(m_0)_i \leftarrow (m_0)_i \oplus (m_j)_i$

// key addition and S-box computation: $(m_0)_i \Leftarrow S(p_i \oplus k_i) \oplus \bigoplus_j s_j$
5. $(m_0)_i \leftarrow S^*\big((m_0)_i \oplus k_i\big)$

// Masks generation and conversion: $(m_0)_i \Leftarrow S(p_i \oplus k_i) \oplus \bigoplus_j (m_j)_i$
6. **for** $j = 1$ **to** d'
7. $(m_j)_i \leftarrow \mathsf{rand}()$
8. $(m_0)_i \leftarrow ((m_0)_i \oplus (m_j)_i) \oplus s_j$
9. **for** $j = d'+1$ **to** d
10. $(m_j)_i \leftarrow \mathsf{rand}()$
11. $(m_0)_i \leftarrow (m_0)_i \oplus (m_j)_i$

12. **return** $(m_0, \ldots, m_d)$

Remark 11.2. In Steps 3 and 8, we used round brackets to underline the order in which the masks are introduced. A new mask is always introduced before removing an old mask. Respecting this order is mandatory for the scheme security.

Masked S-box computation. The look-up table for S^* is computed dynamically at the beginning of the ciphering by performing d' table re-computations such as proposed in [204]. As shown in Chapter 13, this method is insecure for $d' > 2$, or for $d' > 3$ depending on the table re-computation algorithm (see

Section 9.2.1). We will therefore consider that one can compute a masked S-box S^* with $d' \leqslant 3$ only. The secure computation of a masked S-box with $d' > 3$ is left to further investigations.

Indices table computation. Several solutions exist in the literature to randomly generate indices permutation over a finite set [144, 182, 186]. Most of them can be slightly transformed to design tables T of size $t \geqslant N$ containing all the indices 1 to N in a random order and whose remaining cells are filled with $N + 1$. However, few of those solutions are efficient when implemented in low resources devices. In our case, since t is likely to be much greater than N, we have a straightforward algorithm which tends to be very efficient for $t \gg N$. To generate T, we start by initializing all the cells of T to the value $N + 1$. Then, for every $j \leqslant N$, we randomly generate an index $i < t$ until $T[i] = N + 1$ and we move j into $T[i]$. The process is depicted in the following algorithm.

Algorithm 15 Generation of T
INPUT: state length N and shuffling order t
OUTPUT: indices permutation table T

1. **for** $i \leftarrow 0$ **to** $t - 1$
2. $\quad T[i] \leftarrow N + 1$ // Initialization of T
3. **for** $j \leftarrow 1$ **to** N
4. $\quad$ **while** $(T[i] \neq N + 1)$ **do** $i \leftarrow rand(t)$ // Generate random index $i < t$
5. $\quad T[i] = j$
6. **return** T

11.3.2 Protecting the linear layer

The atomic operations λ_i are applied on each part $(m_j)_{i(l)}$ of each share m_j in a random order. For such a purpose a table T' is constructed at the beginning of the ciphering that is randomly filled with all the pairs of indices $(j, i) \in \{0, \ldots, d\} \times \{1, \ldots, L\}$. The linear layer is then implemented such as described by the following algorithm.

Algorithm 16 Protected linear layer
INPUT: the shares $m_0, \ldots, m_d$ s.t. $\bigoplus_i m_i = p$
OUTPUT: the shares $m_0, \ldots, m_d$ s.t. $\bigoplus_i m_i = \lambda(p)$

1. **for** $i_{T'} = 1$ **to** $(d + 1) \cdot L$
2. $\quad (j, i) \leftarrow T'[i_{T'}]$ // Random index look-up
3. $\quad (m_j)_{i(l)} \leftarrow \lambda_i\big((m_j)_{i(l)}\big)$ // Linear operation
4. **return** $(m_0, \ldots, m_d)$

Indices table computation. To implement the random generation of a permutation T' on $\{0, \ldots, d\} \times \{1, \ldots, L\}$, we followed the outlines of the method proposed in [77]. However, since this method can only be applied to generate permutations on sets with cardinality a power of 2 (which is not *a priori* the case for T'), we slightly modified it. Let 2^q be the smallest power of 2 which is greater than $(d + 1)L$. Our algorithm essentially consists in designing a q-bit random permutation T' from a fixed q-bit permutation π and a family of q random values in $\mathbb{F}_2^q$ (Steps 1 to 6 in Algorithm 17). Then, if $(d+1)L$ is not a power of 2, the table T' is transformed into a permutation over $\{0, \ldots, d\} \times \{1, \ldots, L\}$ by deleting the elements which are strictly greater than $(d + 1)L - 1$. The process is detailed in pseudo-code hereafter.

Algorithm 17 Generation of T'
INPUT: parameters (d, L) and a q-bit permutation π with $q = \lceil \log_2((d+1)L) \rceil$
OUTPUT: indices permutation table T'

```
1.  for i ← 0 to q − 1
2.      do alea_i ← rand(q)                    // Initialization of aleas
3.  for j ← 0 to 2^q − 1
4.       do T'[j] ← π[j]
5.      for i ← 0 to q − 1
6.          do T'[j] ← π[T'[j] ⊕ alea_i]       // Process the i^th index
7.  if q ≠ (d + 1)L
8.       then for j ← 0 to (d + 1)L − 1
9.                do i ← j
10.               while T'[i] ⩾ (d + 1)L
11.                   do i ← i + 1
12.               T'[j] ← T'[i]
13.  return T'
```

With Algorithm 17, it is not possible to generate all the permutations over $\{0, \ldots, d\} \times \{1, \ldots, L\}$. In our context, we assume that this does not introduce any weakness in the scheme.

11.4 Time complexity

In the following we express the time complexity of each stage of our scheme in terms of the parameters (t, d, d', N, L) and of constants a_i that depend on the implementation and the device architecture. Moreover, we provide practical values of these constants (in number of clock cycles) for an AES implementation protected with our scheme and running on an 8051-architecture.

Generation of T. Its complexity $\mathcal{C}_T$ satisfies:

$$\mathcal{C}_T = t \times a_0 + N \times a_1 + f(N,t) \times a_2 \ ,$$

where $f(N,t)$ denotes the expected number of iterations of the loop 3-to-5 in Algorithm 15. As explained in [11, App. A], $f(N,t)$ satisfies:

$$f(N,t) = t \sum_{i=0}^{N-1} \frac{1}{t-i} \ .$$

Moreover, $f(N,t)$ can be approximated by $t \ln\left(\frac{t}{t-N}\right)$ for $t \gg N$.

Example. For our AES implementation, we got $a_0 = 6$, $a_1 = 7$ and $a_2 = 9$.

Generation of T'. Let q denote $\lceil \log_2((d+1)L) \rceil$. The number of iterations of loop 8-to-12 in Algorithm 17 in the worst case is 2^q. The complexity $\mathcal{C}_{T'}$ hence satisfies:

$$\mathcal{C}_{T'} = \begin{cases} q \times a_0 + 2^q \times (a_1 + q \times a_2) \text{ if } q = \log_2((d+1)L), \\ q \times a_0 + 2^q \times (a_1 + q \times a_2) + 2^q \times a_3 \text{ otherwise.} \end{cases}$$

Example. For our AES implementation, we got $a_0 = 3$, $a_1 = 15$ and $a_2 = 14$, $a_3 = 17$.

Generation of the Masked S-box. Its complexity $\mathcal{C}_{MS}$ satisfies:

$$\mathcal{C}_{MS} = d' \times a_0 \ .$$

Example. For our AES implementation, we got $a_0 = 4352$.

Protected keyed Substitution Layer. Its complexity $\mathcal{C}_{SL}$ satisfies:

$$\mathcal{C}_{SL} = t \times (a_0 + d \times a_1 + d' \times a_2) \ .$$

Example. For our AES implementation, we got $a_0 = 55$, $a_1 = 37$ and $a_2 = 18$.

Protected Linear Layer. Its complexity $\mathcal{C}_{LL}$ satisfies:

$$\mathcal{C}_{LL} = (d+1)L \times a_0 \ .$$

Example. For our AES implementation, we got $a_0 = 169$.

11.5 Attack paths and parameters setting

In [228, 229], some advanced DPA attacks have been proposed to bypass countermeasures combining first-order masking and shuffling. In Chapter 6 (see Section 6.5.2), we have extended these attacks to defeat combined higher-order masking and shuffling. In this section, we list the different advanced DPA attacks that may be attempted against our scheme. According to the analysis in Chapter 6 and under the Hamming weight model assumption, each attack is associated with a correlation coefficient which depends on the leakage noise deviation σ, the block cipher parameters (n, N, l', L) and the security parameters (d, d', t). As argued in Chapter 6, this coefficient characterizes the attack efficiency. Therefore, under the Hamming weight model assumption, the overall resistance of our scheme can be quantified according to the maximal of the different attack coefficients.

Remark 11.3. We only consider known plaintext attacks *i.e.* we assume the different sensitive variables uniformly distributed. In a chosen plaintext attack, the adversary would be able to fix the value of some sensitive variables which could yield better attack paths. We do not take such attacks into account and let them for further investigations.

As stated in Proposition 6.8 (see Section 6.5.1), under the Hamming weight model assumption, the correlation coefficient $\rho(n, d, \sigma)$ associated with a higher-order DPA attack involving the normalized product combining satisfies:

$$\rho(n, d, \sigma) = (-1)^d \frac{\sqrt{n}}{(n + 4\sigma^2)^{\frac{d+1}{2}}} \ ,$$

where n denotes the bit-size of the targeted shares, d denotes the masking order and σ denotes the leakage noise standard deviation.

When shuffling is combined with higher-order masking, the manipulation times of the $d+1$ shares constitute a $(d+1)$-combination from a set of different times. Denoting by I the set of the possible combinations, Proposition 6.9, (see Chapter 6, Section 6.5.2) states that the correlation coefficient associated with the resulting combined higher-order and integrated DPA attack satisfies:

$$\frac{1}{\sqrt{\#I}} \rho(n, d, \sigma) \ .$$

11.5.1 Attack paths

Every sensitive variable in the scheme is either (1) masked with d unique masks or (2) masked with d' masks shared with other sensitive variables (during the keyed substitution layer).

(1). In the first case, the $d+1$ shares appear during the keyed substitution layer computation and the linear layer computation. In both cases, their manipulation is shuffled.

(1.1). For the keyed substitution layer (see Algorithm 14), the $d+1$ shares all appear during a single iteration of the loop among t. The attack consists in combining the $d+1$ corresponding signals for

each loop iteration and to sum the t obtained combined signals. This attack can be associated with the following correlation coefficient:

$$\rho_1(t, d) = \frac{1}{\sqrt{t}}\rho(n, d, \sigma) \ . \tag{11.3}$$

(1.2). For the linear layer (see Algorithm 16), the $d + 1$ shares appear among $(d + 1) \cdot L$ possible operations. The attack consists in summing all the combinations of $d + 1$ signals among the $(d + 1) \cdot L$ corresponding signals. This attack can be associated with the following correlation coefficient:

$$\rho_2(L, d) = \frac{1}{\sqrt{\binom{(d+1)\cdot L}{d+1}}}\rho(l', d, \sigma) \ . \tag{11.4}$$

Remark 11.4. In the analysis above, we chose to not consider attacks combining shares processed in the linear layers together with shares processed in the keyed substitution layer. Actually, such an attack would yield to a correlation coefficient upper bounded by the maximum of the two correlations in (11.3) and (11.4).

(2). In the second case, the attack targets a d'^{th}-order masked variable occurring during the keyed substitution layer. Two alternatives are possible.

(2.1). The first one is to simultaneously target the masked variable (that appears in one loop iteration among t) and the d' masks that appear at fixed times (*e.g.* in every loop iteration of Algorithm 14 or during the masked S-box computation). The attack hence consists in summing the t possible combined signals obtained by combining the masked variable signal (t possible times) and the d' masks signals (at fixed times). This leads to a correlation coefficient ρ_3 that satisfies:

$$\rho_3(t, d') = \frac{1}{\sqrt{t}}\rho(n, d', \sigma) \ . \tag{11.5}$$

(2.2). The second alternative is to target two different variables both masked with the same sum of d' masks (for instance two masked S-box inputs or outputs). These variables are shuffled among t variables. The attack hence consists in summing all the possible combinations of the two signals among the t corresponding signals. This leads to a correlation coefficient ρ_4 that satisfies:

$$\rho_4(t) = \frac{1}{\sqrt{t \cdot (t-1)}}\rho(n, 2, \sigma) \ . \tag{11.6}$$

11.5.2 Parameters setting

The security parameters (d, d', t) can be chosen to satisfy an arbitrary resistance level characterized by an upper bound ρ^* on the correlation coefficients corresponding to the different attack paths exhibited in the previous section. That is, the parameters are chosen to satisfy the following inequality:

$$\max(|\rho_1|, |\rho_2|, |\rho_3|, |\rho_4|) \leqslant \rho^* \ . \tag{11.7}$$

Among the 3-tuples (d, d', t) satisfying the relation above, we select one among those that minimize the timing complexity (see Section 11.4).

11.6 Application to AES

We implemented our scheme for AES on an 8051-architecture. According to Remark 11.1, the ShiftRows and the MixColumns were merged in a single linear layer applying four times the same operation (but with different state indexings). The block cipher parameters hence satisfy: $n = 8$, $N = 16$, $l = 32$, $l' = 8$ and $L = 4$.

Remark 11.5. In [124], it is claimed that the manipulations of the different bytes in the MixColumns can be shuffled. However it is not clear how to perform such a shuffling in practice since the processing differs according to the byte index.

Table 11.1: Timings for the different stages of the scheme for an AES implementation on an 8051-architecture.

Stage	Timings
T Generation	$\mathcal{C}_T = 112 + t\left(6 + 9\sum_{i=0}^{15}\frac{1}{t-i}\right)$
T' Generation	$\mathcal{C}_{T'} = 3q + 2^q(15 + 14q) \quad [+17 \times 2^q]$
Masked S-box Generation	$\mathcal{C}_{MS} = 4352d'$
Pre-computations	$\mathcal{C}_T + \mathcal{C}_{T'} + \mathcal{C}_{MS}$
Substitution Layer	$\mathcal{C}_{SL} = t(55 + 37d + 18d')$
Linear Layer	$\mathcal{C}_{LL} = 676(d + 1)$
Protected Round	$\mathcal{C}_{SL} + \mathcal{C}_{LL} = 676(d + 1) + t(55 + 37d + 18d')$
Unprotected Round	432

Table 11.1 summarizes the timings obtained for the different stages of the scheme for our implementation.

Remark 11.6. The unprotected round implementation has been optimized, in particular by only using variables stored in DATA memory. Because of memory constraints and due to the scalability of the code corresponding to the protected round, many variables have been stored in XDATA memory which made the implementation more complex. This explains that, even for $d = d' = 0$ and $t = 16$ (*i.e.* when there is no security), the protected round is more time consuming than the unprotected round.

We give hereafter the optimal security parameters (t, d, d') for our AES implementation according to some illustrative values of the device noise deviation σ and of the correlation bound ρ^*. We consider three noise deviation values: 0, $\sqrt{2}$ and $4\sqrt{2}$. In the Hamming weight model, these values respectively correspond to a signal-to-noise ratio (SNR) of $+\infty$, 1 and $\frac{1}{16}$. We consider four correlation bounds: 10^{-1}, 10^{-2}, 10^{-3}, and 10^{-4}. The security parameters and the corresponding timings for the protected AES implementation are given in Table 11.2. Note that all the rounds have been protected.

Table 11.2: Optimal parameters and timings according to SNR and ρ^*.

	SNR $= +\infty$				SNR $= 1$				SNR $= \frac{1}{16}$			
ρ^*	t	d	d'	timings	t	d	d'	timings	t	d	d'	timings
10^{-1}	16	1	1	3.66×10^4	16	1	1	3.66×10^4	16	1	0	2.94×10^4
10^{-2}	20	3	3	8.57×10^4	20	2	2	6.39×10^4	16	1	1	3.66×10^4
10^{-3}	1954	4	3	5.08×10^6	123	3	3	3.13×10^5	16	2	2	5.75×10^4
10^{-4}	195313	5	3	5.75×10^8	12208	4	3	3.15×10^7	19	3	3	8.35×10^4

When SNR $= +\infty$, the bound $d' \leqslant 3$ implies an intensive use of shuffling in the keyed substitution layer. The resulting parameters for correlation bounds 10^{-3} and 10^{-4} imply timings that quickly become prohibitive. A solution to overcome this drawback would be to design secure table re-computation algorithms for $d' \geqslant 3$. Besides, these timings underline the difficulty in securing block ciphers implementations with pure software countermeasures. When the leakage signals are not very noisy (SNR $= 1$), timings clearly decrease (by a factor from 10 to 20). This illustrates, once again, the soundness of combining masking with noise addition. This is even clearer when the noise is stronger (SNR $= \frac{1}{16}$), where it can be noticed that the addition of dummy operations is almost not required to achieve the desired security level.

Chapter 12

Cryptanalysis of a masking scheme based on the Fourier transform

Contents

12.1 Introduction

At CHES 2006, a generic first-order masking scheme for S-boxes was published that is based on the Fourier transform. In this chapter, we show that this countermeasure has a flaw and that it can be broken by first-order SCA. Moreover, we report successful practical DPA attacks on two different S-box implementations using this countermeasure. Finally, we propose an improvement of the original countermeasure and we prove its security against first-order DPA.

The results presented in this chapter have been published in collaboration with Jean-Sébastien Coron, Christophe Giraud and Emmanuel Prouff in the international workshop on *Cryptographic Hardware and Embedded Systems (CHES 2008)* [1].

12.2 Masked S-box computation based on the Fourier transform

In [192], Emmanuel Prouff, Christophe Giraud and Sébastien Aumônier proposed a new algorithm to perform an S-box computation on masked data. The method is based on the involutivity property of the *Fourier Transform*. Before describing it, let us first recall some basics about the transformation itself.

For every function $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$, the Fourier transform $\widehat{S}$ of S is defined for every $z = (z_0, \ldots, z_{n-1}) \in \mathbb{F}_2^n$ by:

$$\widehat{S}(z) = \sum_{a \in \mathbb{F}_2^n} S(a)(-1)^{a \cdot z} \ , \tag{12.1}$$

where $\cdot$ denotes the scalar product defined by $a \cdot z = \bigoplus_{i=0}^{n-1} a_i z_i$.

It is well known that this transformation is involutive, which means that $\widehat{\widehat{S}} = 2^n S$ or equivalently that:

$$S(z) = \frac{1}{2^n} \sum_{a \in \mathbb{F}_2^n} \widehat{S}(a)(-1)^{a \cdot z},\ z \in \mathbb{F}_2^n \ . \tag{12.2}$$

Let r_1, r_2, r_3 and r_4 be 4 random masks belonging to $\mathbb{F}_2^n$, and let z denotes a sensitive variable. The algorithm proposed in [192] to process $S(z) + r_3 \bmod 2^n$ securely from $\tilde{z} = z \oplus r_1$ and r_1, implements the right-hand side calculus of the following relation (which is a slightly modified version of Relation (12.2)):

$$(-1)^{(\tilde{z} \oplus r_2) \cdot r_1} S(z) + r_3 \bmod 2^n = \left\lfloor \frac{1}{2^n} \left(r' + \sum_{a \in \mathbb{F}_2^n} \widehat{S}(a)(-1)^{a \cdot \tilde{z} \oplus r_1 \cdot (\tilde{z} \oplus a \oplus r_2)} \bmod 2^{2n} \right) \right\rfloor \ , \tag{12.3}$$

where $r' = 2^n r_3 + r_4$.

Let SSP denote the signed scalar product $X, Y \mapsto (-1)^{X \cdot Y}$, let $\boxplus$ denote the addition modulo 2^{2n} and let $\times$ denote the multiplication of two values belonging to $\{-1, 1\}$. We recall hereafter the algorithm proposed in [192] to process the right-hand side of (12.3) securely.

Algorithm 18 Computation of an arithmetically masked S-box output from a Boolean masked input

INPUT: a masked input $\tilde{z} = z \oplus r_1$, the input mask r_1 and a lookup table for $\widehat{S}$

OUTPUT: the 3-tuple $((-1)^{(\tilde{z} \oplus r_2) \cdot r_1} S(z) + r_3 \bmod 2^n, r_3, r_2)$ where r_2 and r_3 are random values.

1. $(r_2, r_3, r_4) \leftarrow \big(\mathsf{rand}(n), \mathsf{rand}(n), \mathsf{rand}(n)\big)$
2. $result \leftarrow 2^n r_3 + r_4$
3. **for** a **from** 0 **to** $2^n - 1$ **do**
4. $\quad T_1 \leftarrow \mathrm{SSP}(a, \tilde{z})$ $\qquad [T_1 = (-1)^{a \cdot \tilde{z}}]$
5. $\quad T_2 \leftarrow \tilde{z} \oplus a$ $\qquad [T_2 = \tilde{z} \oplus a]$
6. $\quad T_2 \leftarrow T_2 \oplus r_2$ $\qquad [T_2 = \tilde{z} \oplus a \oplus r_2]$
7. $\quad T_2 \leftarrow \mathrm{SSP}(r_1, T_2)$ $\qquad [T_2 = (-1)^{r_1 \cdot (\tilde{z} \oplus a \oplus r_2)}]$
8. $\quad T_2 \leftarrow T_1 \times T_2$ $\qquad [T_2 = (-1)^{a \cdot \tilde{z} \oplus r_1 \cdot (\tilde{z} \oplus a \oplus r_2)}]$
9. $\quad T_2 \leftarrow T_2 \times \widehat{S}(a)$ $\qquad [T_2 = \widehat{S}(a)(-1)^{a \cdot \tilde{z} \oplus r_1 \cdot (\tilde{z} \oplus a \oplus r_2)}]$
10. $\quad result \leftarrow result \boxplus T_2$ $\qquad [result = (2^n r_3 + r_4) \boxplus \sum_{i \in \{0,a\}} \widehat{S}(i)(-1)^{i \cdot \tilde{z} \oplus r_1 \cdot (\tilde{z} \oplus i \oplus r_2)}]$
11. $result \leftarrow result \gg n$ $\qquad [result = (-1)^{(\tilde{z} \oplus r_2) \cdot r_1} S(z) + r_3 \bmod 2^n]$
12. **return** $(result, r_3, r_2)$

Finally, it is proposed in [192] to use the method described in [117] in order to transform the arithmetic masking of the output of Algorithm 18 into a Boolean masking.

The authors of [192] had proposed a proof of security *versus* first-order SCA for the countermeasure defined by Algorithm 18, but as we will see in the next section, the proof is flawed and the countermeasure is not secure against first-order SCA.

12.3 First-order attack against the Fourier transform based S-box computation

12.3.1 First-order flaw

Unlike what is claimed in [192], the implementation of Algorithm 18 is not immune against first-order SCA. Indeed, the variable $v = a \cdot \tilde{z} \oplus r_1 \cdot (\tilde{z} \oplus a \oplus r_2)$ processed at Step 8 brings information about the

sensitive variable z (recalling $\tilde{z} = z \oplus r_1$). To exhibit the dependence between v and z, let us first rewrite v as follows:

$$\begin{aligned} v &= a \cdot \tilde{z} \oplus r_1 \cdot (\tilde{z} \oplus a \oplus r_2) \\ &= a \cdot (z \oplus r_1) \oplus r_1 \cdot (\tilde{z} \oplus a \oplus r_2) \\ &= a \cdot z \oplus r_1 \cdot (\tilde{z} \oplus r_2) \ . \end{aligned}$$

The relation above shows that the intermediate variable v equals the sensitive variable $a \cdot z$ (a being a loop index) masked with the scalar product $r_1 \cdot (\tilde{z} \oplus r_2)$. Since r_2 is uniformly distributed and is independent of both z and r_1, then so does the variable $\tilde{z} \oplus r_2$. The flaw of the method proposed in [192] comes from the fact that the scalar product of two uniformly distributed random variables does not output an uniformly distributed random variable. For example, the product $b_1 \cdot b_2$ of two random bits b_1 and b_2 equals 0 with probability 3/4, and equals 1 with probability 1/4. More generally, for n-bit random variables we have the following lemma.

Lemma 12.1. *Let X and Y be two random variables uniformly distributed over $\mathbb{F}_2^n$ and mutually independent. Then the scalar product $X \cdot Y$ satisfies:*

$$\mathrm{P}\left[X \cdot Y = 0\right] = \frac{1}{2} + \frac{1}{2^{n+1}} \ . \tag{12.4}$$

Proof. We have:

$$P[X \cdot Y = 0] = P[X \neq 0] \cdot P[X \cdot Y = 0 | X \neq 0] + P[X = 0] \cdot P[X \cdot Y = 0 | X = 0] \ .$$

Since the Boolean function $y \in \mathbb{F}_2^n \mapsto x \cdot y$ is linear and non-zero for every $x \neq 0$, we have $\#\{x \cdot y = 1\} = \#\{x \cdot y = 0\} = 2^{n-1}$. This, together with the fact that X and Y are independent, implies $P[X \cdot Y = 0 | X \neq 0] = \frac{1}{2}$. Since $P[X \cdot Y = 0 | X = 0] = 1$ and $P[X \neq 0] = \frac{2^n - 1}{2^n}$, we deduce (12.4). □

Remark 12.1. In the security proof conducted in [192], it is stated that the uniform distribution of X and Y implies the uniform distribution of $X \cdot Y$. We show in Lemma 12.1 that this assertion is actually wrong.

Lemma 12.1 implies that the distribution of $r_1 \cdot (\tilde{z} \oplus r_2)$ has a bias $\frac{1}{2^{n+1}}$ with respect to the uniform distribution. Since the sensitive variable $a \cdot z$ is masked with a biased mask, the variable v defined in (12.4) leaks information on $a \cdot z$. This information can be exploited to mount a first-order DPA attack.

12.3.2 DPA attack exploiting a biased masking

We consider a DPA attack targeting a bit b that satisfies:

$$b = f(X, k^*) \oplus R \ , \tag{12.5}$$

where f is a Boolean function and X is a public variable, k^* is a secret key element (that can be exhaustively searched) and R is a random bit. The leakage resulting from this bit is denoted $L(b)$.

If R is uniformly distributed over $\mathbb{F}_2$, then no successful DPA attack is possible. Indeed, in that case b equals 0 (resp. 1) with probability $\frac{1}{2}$ independently of $f(X, k^*)$. Conversely, when the distribution of R is biased compared to the uniform distribution, then the distribution of b depends on $f(X, k^*)$, which renders DPA possible. In the following, we denote by $\varepsilon \neq 0$ the bias such that $P[R = 0] = \frac{1}{2} + \varepsilon$.

The DPA works as described Section 3.5.1. That is, for a given measurements vector $(l_i)_i$ and corresponding input vector $(x_i)_i$, the attacker computes for every key guess k the differential Δ_k defined as[18]:

$$\Delta_k = \frac{\sum_{i=1}^N f(x_i, k) \times l_i}{\sum_{i=1}^N f(x_i, k)} - \frac{\sum_{i=1}^N (1 - f(x_i, k)) \times l_i}{\sum_{i=1}^N (1 - f(x_i, k))} \ . \tag{12.6}$$

[18] Here f is a Boolean function hence the index j is ignored compared to the definition given in Section 3.5.1

When the biais is strictly lower than $\frac{1}{2}$, the randomization provided by R implies that the bit effectively processed equals $f(x_i, k^*)$ with probability $\frac{1}{2} + \varepsilon$. One deduces that, for the correct key hypothesis, a portion $\frac{1}{2} + \varepsilon$ of the predicted bits $f(x_i, k^*)$ effectively correspond to the computed bits while a portion $\frac{1}{2} - \varepsilon$ does not on average. This implies that the expectation of the differential for the correct key hypothesis satisfies:

$$\mathrm{E}\left[\Delta_{k^*}\right] = \left(\frac{1}{2} + \varepsilon\right)\left(\mathrm{E}\left[L(1)\right] - \mathrm{E}\left[L(0)\right]\right) + \left(\frac{1}{2} - \varepsilon\right)\left(\mathrm{E}\left[L(0)\right] - \mathrm{E}\left[L(1)\right]\right) \ ,$$

that is:

$$\mathrm{E}\left[\Delta_{k^*}\right] = 2\varepsilon \times \left(\mathrm{E}\left[L(1)\right] - \mathrm{E}\left[L(0)\right]\right) \ . \tag{12.7}$$

If the key hypothesis k is incorrect then a *ratio* $\alpha \in [0, 1]$ of the x_i's satisfies $f(x_i, k) \neq f(x_i, k^*)$. The expectation of the differential then satisfies:

$$\mathrm{E}\left[\Delta_k\right] = (1 - \alpha)\ \mathrm{E}\left[\Delta_{k^*}\right] - \alpha\ \mathrm{E}\left[\Delta_{k^*}\right] = (1 - 2\alpha)\ \mathrm{E}\left[\Delta_{k^*}\right] \ . \tag{12.8}$$

Since α is usually approximately $\frac{1}{2}$, we have $\mathrm{E}\left[\Delta_{k \neq k^*}\right] \simeq 0$. This implies that, for a sufficiently large N, the correct key hypothesis is such that Δ_k is of maximum amplitude.

Remark 12.2. Depending on the function f, it may happen that the correct key hypothesis is not the single one for which Δ_k is of maximum amplitude. Indeed, a key hypothesis such that $\alpha = 1$ also results in a differential of maximal amplitude. According to (12.6), this differential and the one corresponding to the correct key hypothesis have exactly the same amplitude but have opposite signs. To differentiate them the attacker needs to determine the polarity of $\mathrm{E}\left[L(1)\right] - \mathrm{E}\left[L(0)\right]$.

The effect of the biased masking can be observed from (12.7) and (12.8). We see that for every k, the expectation of Δ_k is divided by a factor $\frac{1}{2\varepsilon}$ compared to an unprotected implementation. This implies, according to the analysis in [74], that the number of required leakage measurements is roughly multiplied by $(\frac{1}{2\varepsilon})^2$.

As a result, Lemma 12.1 implies that a DPA on Algorithm 18 exploiting the flaw exhibited in Section 12.3.1 is expected to require about 2^{2n} times more leakage measurements than a DPA when no masking is used. Since Algorithm 18 is only interesting for a small value of n (*e.g.* $n = 4$), this factor is not prohibitive.

12.3.3 DPA attack on the flaw

In this section, we apply the DPA attack described in Section 12.3.2 in order to exploit the flaw exhibited in Section 12.3.1. More precisely, our attack targets a bit b which is a scalar product $a \cdot z$ masked with a biased mask $R = r_1 \cdot (\tilde{z} \oplus r_2)$, that is:

$$b = a \cdot z \oplus R \ . \tag{12.9}$$

We recall that a refers to a loop index in Algorithm 18 and that its value can be chosen by the attacker among $\{0, \ldots, 2^n - 1\}$ (by choosing the target loop iteration). The sensitive variable z is the sensitive S-box input and it can be written as a function of a public variable X and a piece of secret data k^*. The way our attack is performed depends on this function which can take several forms. In the sequel we consider two usual cases.

The first one is referred as the *linear case* and assumes:

$$z = X \oplus k^* \ .$$

This occurs for instance in AES and in FOX algorithms for the first round S-box computation.

The second case, referred as the *non-linear case*, assumes the existence of a non-linear transformation ϕ such that:

$$z = \phi(X \oplus k^*) \ .$$

This occurs for instance in the AES algorithm implemented using the *composite field method* [178, 192] (see [192, §4.1] for details). In that case, ϕ is the non-linear $(8, 4)$-function which from $a \in \mathbb{F}_{256}$ processes $d \in \mathbb{F}_{16}$ according to the notations of [178, 192].

12.3.3.1 The linear case

We consider here the case where the targeted bit can be expressed as $b = a \cdot (X \oplus k^*) \oplus R$ that is:

$$b = a \cdot X \oplus a \cdot k^* \oplus R \ . \tag{12.10}$$

The bit b in (12.10) only depends on one secret binary value $a \cdot k^*$. Therefore, a DPA on b will provide at most one bit of information on k^*. Hence, recovering the entire secret k^* requires to perform a DPA attack on b for several (say t) different loop indices $a_0, \ldots, a_{t-1}$.

When mounting a DPA attack on b for a particular loop index a, the sequence of guesses can only take one of the two following forms: $(a \cdot x_i)_i$ or $(a \cdot x_i \oplus 1)_i$. According to (12.6), these two sequences result in two differentials that are opposite one to each other. The attacker does not know *a priori* which of these differentials correspond to the correct key hypothesis. Indeed, depending on the device, the polarity $(-1)^s$ of the good differential $\Delta_{a \cdot k^*}$ may be positive or negative. In other terms, the DPA allows the attacker to recover the value of $a \cdot k^* \oplus s$, where k^* and s are unknown.

Since the polarity s is the same for all the loop indices a, then performing t DPA attacks for t different loop indices $a_0, \ldots, a_{t-1}$ provides the attacker with a system of t equations and $n+1$ variables (the polarity bit s and the n bits of k^*). Solving this system requires to have at least $t = n+1$ equations. After choosing n indices a_i having linearly independent vectorial representations in $\mathbb{F}_2^n$ and after defining $a_n = a_0 \oplus a_1$, it can be checked that solving the system allows the attacker to unambiguously determine the value of k^*.

12.3.3.2 The non-linear case

We now consider the case where b satisfies:

$$b = a \cdot \phi(X \oplus k^*) \oplus R \ . \tag{12.11}$$

For a non-linear ϕ, the attack is analogous to a classical DPA on some output bit of *e.g.* a DES or AES S-box [148]. The non-linearity of ϕ ensures that for the correct key hypotheses a peak of maximal amplitude will appear while for most other key hypothesis no peak will appear. This enables to fully recover k^*.

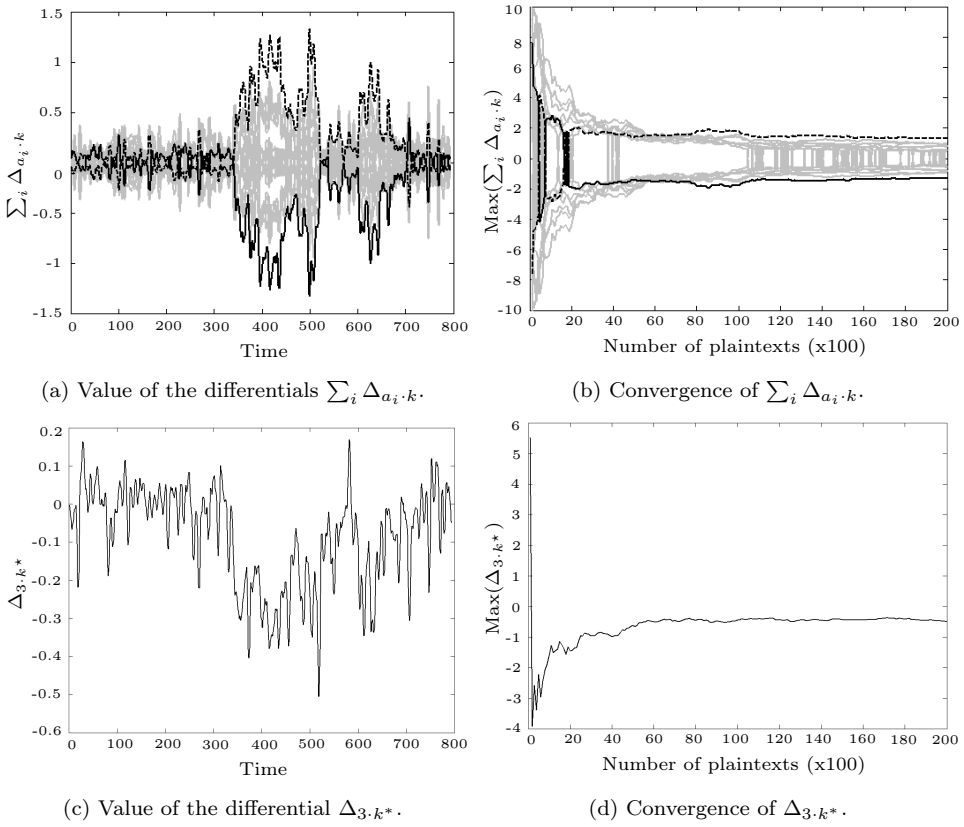
In this section, we have described how to exploit the leakage on a sensitive bit which is masked with a biased random bit. In the linear case, the attack requires to perform $n+1$ DPAs while only one DPA is needed in the non-linear case. In the following section, we present experimental results for these two attacks.

12.4 Experimental results

We put into practice the attacks described in Section 12.3 for two S-box implementations on an 8-bit smart card. Both attacks exploited the power consumption resulting from several S-box computations.

Regarding the linear case, we performed the attack on the S-box computation of FOX algorithm during the first round protected by the method described in [192]. In this case, the sensitive bits we targeted are of the form $a \cdot (X \oplus k^*) \oplus R$, where $a, X, k^* \in \mathbb{F}_2^4$. Following the outlines of the attack described in Section 12.3.3 for the linear case, we have applied $4+1$ DPAs on five different loop iterations of Algorithm 18, namely one DPA for every $a \in \{1, 2, 4, 8, 3\}$.

Figure 12.1.a represents the value of $\sum_{i=0}^{3} \Delta_{a_i \cdot k}$, where $a_i = 2^i$, obtained after 20 000 executions of the algorithm. The full black curve corresponds to the correct subkey value k^* and the dotted black curve corresponds to the complementary of this value. As expected, these two candidates are such that the highest peaks of the differential vectors $\Delta_{a_i \cdot k}$ are either all positive or all negative, hence leading to the highest amplitudes for $\sum_{i=0}^{3} \Delta_{a_i \cdot k}$. As explained in Section 12.3.3, we then computed the differential $\Delta_{a \cdot k^*}$ for $a = a_0 \oplus a_1 = 3$. Figure 12.1.c illustrates this computation. The polarity of the highest peak of $\Delta_{3 \cdot k^*}$ being negative, one deduces that the correct subkey value k^* corresponds to the full black curve in Figure 12.1.a.



(a) Value of the differentials $\sum_i \Delta_{a_i \cdot k}$.

(b) Convergence of $\sum_i \Delta_{a_i \cdot k}$.

(c) Value of the differential $\Delta_{3 \cdot k^*}$.

(d) Convergence of $\Delta_{3 \cdot k^*}$.

Figure 12.1: Practical DPA attack – the linear case.

Figures 12.1.b and 12.1.d represent respectively the convergence of the peak of maximal amplitude for $\sum_{i=0}^{3} \Delta_{a_i \cdot k}$ and for $\Delta_{3 \cdot k^*}$ according to the number of power consumption measurements. By analyzing these curves, we deduce that the value of the 4-bit subkey k^* is recovered by using about 8 000 executions of the algorithm.

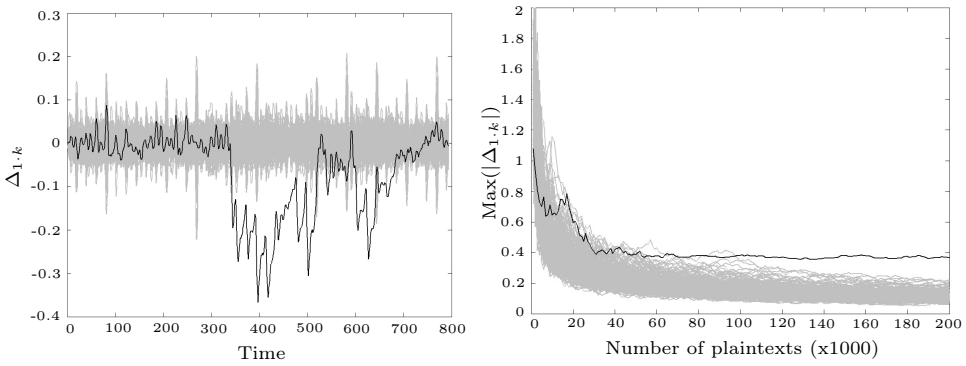
Regarding the non-linear case, we attacked the AES S-box computation using the composite field method in order to perform the inversion in $\mathbb{F}_2^4$ instead of $\mathbb{F}_2^8$ and the method of [192] to protect this inversion (see [192, § 4.1] for more details). In that case, the targeted bit is of the form $a \cdot \phi(X \oplus k^*) \oplus R$ where $X, k^* \in \mathbb{F}_2^8$, $a \in \mathbb{F}_2^4$ and $\phi : \mathbb{F}_2^8 \to \mathbb{F}_2^4$. Figure 12.2.a represents the value of the differentials Δ_k's for $k \in \mathbb{F}_2^8$ and $a = 1$, when 200 000 executions of the algorithm are used. It can be seen that the correct subkey k^* (plotted in black) is easily distinguishable.

Figure 12.2.b represents the convergence of the maximum peak amplitude for the differentials according to the number of power consumption measurements. The analysis of these curves shows us that the value of the 8-bit subkey k^* is recovered after about 100 000 executions of the algorithm.

12.5 An improved version of the Fourier transform based S-box computation

In the following we propose an improvement of Algorithm 18 that allows to circumvent the flaw depicted in Section 12.3.1 and also leads to a more efficient implementation.

The new algorithm is still a secure computation of a Fourier transform but it is based on a slightly modified version of (12.3) which we rewrite in the following form:



(a) Value of the vectors $\Delta_{1\cdot k}$.

(b) Convergence of $|\Delta_{1\cdot k}|$.

Figure 12.2: Practical DPA attack – the non-linear case.

$$(-1)^{r_2}S(z) + r_3 \bmod 2^n = \left\lfloor \frac{1}{2^n}\left(r' + \sum_{a\in\mathbb{F}_2^n} \widehat{S}(a)(-1)^{r_2\oplus a\cdot\tilde{z}\oplus a\cdot r_1} \bmod 2^{2n}\right)\right\rfloor \ , \tag{12.12}$$

where $\tilde{z} = z \oplus r_1$, $r_2 \in \mathbb{F}$, $(r_1, r_3, r_4) \in (\mathbb{F}_2^n)^3$ and $r' = 2^n r_3 + r_4$.

After a brief look at (12.12) (and before a deeper analysis), we can notice that the sensitive variable $a \cdot z$ is now masked with the uniformly distributed random bit r_2. Furthermore, it may be noticed that the exponent in the summation in (12.12) involves less operations than in (12.3).

Let us denote by SP the function $X, Y \mapsto X \cdot Y$ and by SFT the function $X, T \mapsto \widehat{S}(X)(-1)^T$. As we prove in this section, Algorithm 19 implements (12.12) securely.

Algorithm 19 First-order secure S-box computation

INPUT: a masked value $\tilde{z} = z \oplus r_1$ and the mask r_1

OUTPUT: the 3-tuple $((-1)^{r_2}S(z) + r_3 \mod 2^n, r_3, r_2)$, where r_2 and r_3 are random values.

1. Generate a random bit r_2
2. $(r_3, r_4) \leftarrow (\mathsf{rand}(n), \mathsf{rand}(n))$
3. $result \leftarrow 2^n r_3 + r_4$
4. **for** a **from** 0 **to** $2^n - 1$ **do**
5. $\quad T_1 \leftarrow \mathrm{SP}(a, \tilde{z})$ $\quad [T_1 = a \cdot \tilde{z}]$
6. $\quad T_1 \leftarrow T_1 \oplus r_2$ $\quad [T_1 = r_2 \oplus a \cdot \tilde{z}]$
7. $\quad T_2 \leftarrow \mathrm{SP}(a, r_1)$ $\quad [T_2 = a \cdot r_1]$
8. $\quad T_1 \leftarrow T_1 \oplus T_2$ $\quad [T_1 = r_2 \oplus a \cdot z]$
9. $\quad T_1 \leftarrow \mathrm{SFT}(a, T_1)$ $\quad [T_1 = \widehat{S}(a)(-1)^{r_2\oplus a\cdot z}]$
10. $\quad result \leftarrow result \boxplus T_1$ $\quad [result = (2^n r_3 + r_4) \boxplus \sum_{i\in\{0,a\}} \widehat{S}(i)(-1)^{r_2\oplus i\cdot z}]$
11. $result \leftarrow result \gg n$ $\quad [result = (-1)^{r_2}S(z) + r_3 \bmod 2^n]$
12. **return** $(result, r_3, r_2)$

12.5.1 Efficiency analysis

Although Algorithm 19 is more secure than Algorithm 18, it is also faster. For each loop iteration, Algorithm 19 requires two XORs, two calls to the function SP and one call to the lookup table SFT. Therefore, for each loop iteration, Algorithm 18 performs 2 extra multiplications compared to Algorithm 19. Combining this result with the fact that function SP is slightly faster than function SSP, we deduce that our method is faster than the one proposed in [192].

Table 12.1: The different sensitive values manipulated during Algorithm 19.

Step	Instruction			Masked Value	$Mask(s)$
5.1	register	$\leftarrow$	$\tilde{z}$	$\tilde{z}$	r_1
5.2	T_1	$\leftarrow$	$\mathrm{SP}(a, \tilde{z})$	$a \cdot \tilde{z}$	$a \cdot r_1$
6	T_1	$\leftarrow$	$T_1 \oplus r_2$	$r_2 \oplus a \cdot \tilde{z}$	$r_2 \oplus a \cdot r_1$
8	T_1	$\leftarrow$	$T_1 \oplus T_2$	$r_2 \oplus a \cdot z$	r_2
9	T_1	$\leftarrow$	$\mathrm{SFT}(a, T_1)$	$\widehat{S}(a)(-1)^{r_2 \oplus a \cdot z}$	r_2
10	$result$	$\leftarrow$	$result \boxplus T_1$	$(2^n r_3 + r_4) \boxplus \sum_i \widehat{S}(i)(-1)^{r_2 \oplus i \cdot z}$	(r_2, r_3, r_4)
11	$result$	$\leftarrow$	$result \gg n$	$(-1)^{r_2} S(z) + r_3 \bmod 2^n$	r_3

12.5.2 Security analysis

In Table 12.1, we list the intermediate variables of Algorithm 18 that involve a sensitive variable. The values which only depend on the loop counter or on a random value are obviously omitted.

As it can be checked in Table 12.1, the intermediate variables manipulated at Steps 5.1, 6, 8, 9, 10 and 11 are masked with a uniformly distributed random variable (resp. r_1, $r_2 \oplus a \cdot r_1$, r_2, r_2, $r_3||r_4$ and r_3) which is independent of the sensitive variable. Those intermediate variables are therefore independent of the sensitive variable z. The intermediate variable at Step 5.2 can be rewritten $a \cdot z \oplus a \cdot r_1$. When a equals 0, this variable equals 0 whatever z and r_1. Otherwise, for every $a \neq 0$ the variable $a \cdot r_1$ is uniformly distributed and independent of z. We deduce that $a \cdot z \oplus a \cdot r_1$ (and hence $a \cdot \tilde{z}$) is independent of z whatever a.

To summarize, we have proved that all the intermediate variables manipulated during the execution of Algorithm 18 are independent of z. As a consequence, our improved method is secure against first-order DPA.

Chapter 13

Cryptanalysis of a higher-order masking scheme

Contents

13.1 Introduction

At CT-RSA 2006, a higher-order masking scheme has been proposed by Kai Schramm and Christof Paar [204]. It is actually the only higher-order masking scheme for S-boxes that has been published in the literature. The authors claimed that the scheme is resistant against d^{th}-order DPA for any arbitrary chosen order d. In this chapter, we exhibit several third-order SCA attacks that can defeat the Schramm and Paar's scheme for any value of d.

The results presented in this chapter have been published in collaboration with Jean-Sébastien Coron and Emmanuel Prouff in the international workshop on *Cryptographic Hardware and Embedded Systems (CHES 2007)* [2].

13.2 The generic masking scheme

13.2.1 Description

Kai Schramm and Christof Paar propose in [204] a masking scheme for AES which aims to thwart d^{th}-order SCA for any arbitrary chosen d. Every sensitive byte z appearing in the algorithm is never directly

manipulated and is represented by $d+1$ values m_0, m_1, ..., m_d. To ensure the SCA-resistance, the shares $(m_i)_{i\geqslant 1}$ take random values and to ensure completeness, m_0 satisfies:

$$m_0 = z \oplus \bigoplus_{i=1}^{d} m_i \ . \tag{13.1}$$

When a transformation S must be applied to z, $d+1$ new values s_0, s_1, ..., s_d must be processed from the m_i's such that:

$$s_0 = S(z) \oplus \bigoplus_{i=1}^{d} s_i \ . \tag{13.2}$$

As argued in Chapter 8, the critical point of such a method is to deduce the s_i's from the m_i's when S is non-linear, without compromising the security of the scheme against d^{th}-order SCA.

To tackle this issue, Kai Schramm and Christof Paar propose to adapt the table re-computation method which is widely used to protect implementations against first-order SCA (see Section 9.2.1). In their proposal, the d output masks $(s_i)_{i\geqslant 1}$ are randomly generated and a new table S^* is derived from m_1, ..., m_d and s_1, ..., s_d in such a way that S^* satisfies for every x:

$$S^*(x) = S\left(x \oplus \bigoplus_{i=1}^{d} m_i\right) \oplus \bigoplus_{i=1}^{d} s_i \ . \tag{13.3}$$

Then, one lets $s_0 \leftarrow S^*(m_0)$; using (13.1) this gives $s_0 = S(z) \oplus \bigoplus_{i=1}^{d} s_i$ as required.

To ensure that the design of S^* induces no flaw with respect to d^{th}-order SCA, it involves d successive table re-computations from $S_0 = S$ to $S_d = S^*$. For every $j \in \{1, \ldots, d\}$, the j^{th} re-computation produces a new S-box S_j from S_{j-1} such that, for every x:

$$S_j(x) = S_{j-1}(x \oplus m_j) \oplus s_j = S\left(x \oplus \bigoplus_{i=1}^{j} m_i\right) \oplus \bigoplus_{i=1}^{j} s_i \ , \tag{13.4}$$

which for $j = d$ satisfies (13.3).

In the next section, we consider that the table re-computations are performed according to Algorithm 2 (see Section 9.2.1) and we exhibit a third-order flaw. Afterwards, we address the scheme security with respect to other re-computation algorithms.

13.2.2 The third-order flaw

Before describing the flaw, and to simplify the presentation, we will denote $\mathbf{m} = \bigoplus_{i=1}^{d} m_i$ and $\mathbf{s} = \bigoplus_{i=1}^{d} s_i$. During the re-computation of S_d from S_{d-1}, the variables $S_d(0) = S(\mathbf{m}) \oplus \mathbf{s}$ and $S_d(1) = S(\mathbf{m} \oplus 1) \oplus \mathbf{s}$ are respectively manipulated during the first iteration and the second iteration of the loop (see Algorithm 2). The manipulation of these two variables together with m_0 induces a third-order flaw. In fact, recalling that m_0 satisfies $m_0 = z \oplus \mathbf{m}$, we have

$$(m_0, S_d(0), S_d(1)) = (z \oplus \mathbf{m}, S(\mathbf{m}) \oplus \mathbf{s}, S(\mathbf{m} \oplus 1) \oplus \mathbf{s}) \ . \tag{13.5}$$

It can be checked from (13.5) that $(m_0, S_d(0), S_d(1))$ and z are not independent, which implies that a third-order SCA is potentially feasible. Namely, given $S_d(0)$ and $S_d(1)$, one can compute $\Delta = S_d(0) \oplus S_d(1) = S(\mathbf{m}) \oplus S(\mathbf{m} \oplus 1)$. This makes it possible to recover $\mathbf{m}$ with high probability since the number of values z satisfying $\Delta = S(z) \oplus S(z \oplus 1)$ is small when S has good cryptographic properties (*e.g.* this equation admits at most 4 solutions if S is the AES S-box). Then, knowing the value of $\mathbf{m}$ allows to recover z from m_0 since they satisfy $z = m_0 \oplus \mathbf{m}$.

The discussion above demonstrates that the use of Algorithm 2 to perform the table re-computations makes Schramm and Paar's scheme vulnerable to third-order SCA for any value d.

Even if the third-order flaw above has been exhibited for the first and the second loop iterations, the generic scheme admits more generally a flaw $(m_0, S_d(e_1), S_d(e_2))$ for every pair $(e_1, e_2) \in \{0, \ldots, 255\}^2$ of loop indices such that $e_1 \neq e_2$.

The importance of the third-order flaw depends on the amount of information that $(m_0, S_d(e_1), S_d(e_2))$ provides about z. As we show in [2, App. B], this amount depends on the cryptographic properties of S and on the value $e_1 \oplus e_2$. In fact, for every S-box S defined from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$ and for every sub-set $\{e_1, e_2\} \subseteq \mathbb{F}_2^n$, the mutual information $\mathrm{I}(z; (m_0, S_d(e_1), S_d(e_2)))$ between z and $(m_0, S_d(e_1), S_d(e_2))$ satisfies:

$$n - \log(\delta) \leqslant \mathrm{I}(z; (m_0, S_d(e_1), S_d(e_2))) \leqslant n \ , \tag{13.6}$$

where δ denotes $\max_{e \in \mathbb{F}_2^{n*}, z \in \mathbb{F}_2^m} \#\{x \in \mathbb{F}_2^n;\ S(x) \oplus S(x \oplus e) = z\}$.

To resist against differential cryptanalysis [43], the AES S-box ($n = 8$, $m = 8$) has been designed in such a way that $\delta = 4$. Consequently, if S is the AES S-box then (13.6) implies that the mutual information between z and $(m_0, S_d(e_1), S_d(e_2))$ is lower bounded by 6. In fact, this mutual information equals $7 - \frac{1}{64} \approx 6.98$ for every sub-set $\{e_1, e_2\} \subseteq \mathbb{F}_2^n$, which means that knowing the values of m_0, $S_d(e_1)$ and $S_d(e_2)$ reveals almost 7 bits of z (out of 8) on average.

13.2.3 Further re-computation algorithms

In this section, we focus on the different re-computation algorithms given by Kai Schramm and Christof Paar in [204] and we analyze how they impact the security of their scheme.

In [204], a variant of Algorithm 2 is given in which Step 2 is replaced by:

$$S_j(x \oplus m_j) \leftarrow S_{j-1}(x) \oplus s_j \ .$$

If this variant is used in Schramm and Paar's scheme, the third-order flaw presented in the previous section becomes a fourth-order flaw. Indeed, the values stored in memory during the first and the second loop iterations of the d^{th} table re-computation are no longer $S_d(0)$ and $S_d(1)$ but $S_d(m_d)$ and $S_d(m_d \oplus 1)$. The two last variables satisfy:

$$S_d(m_d) = S(\mathbf{m} \oplus m_d) \oplus \mathbf{s} \quad \text{and} \quad S_d(m_d \oplus 1) = S(\mathbf{m} \oplus m_d \oplus 1) \oplus \mathbf{s} \ .$$

Thus, by analogy with Section 13.2.2, knowing the values of these two variables reveals information about $\mathbf{m} \oplus m_d$ (instead of $\mathbf{m}$ in Section 13.2.2). Therefore, in addition to these two variables, an attacker needs to target not only $m_0 = z \oplus \mathbf{m}$ but also m_d in order to unmask z. This results in a fourth-order flaw.

in [204], another re-computation algorithm is recalled which has been introduced in [234]. However, this algorithm is not suitable as its execution time depends on the input mask value. Such a dependence induces a flaw with respect to first-order SCA. Indeed, as the re-computation duration depends on the mask value, the manipulation date of the masked variable after the re-computation also depends on the mask value. This implies that the distribution of the mask given the manipulation date of the masked variable is not uniform. Consequently, a first-order flaw occurs at this date.

Finally, Kai Schramm and Christof Paar propose in [204] a new table re-computation algorithm. This algorithm does not require any RAM allocation for the output table because it modifies the input table itself to compute the new one.

Algorithm 20 Schramm and Paar's re-computation

INPUT: an input mask r, an output mask s, a look-up table $T[\cdot] = S(\cdot)$
OUTPUT: the modified look-up table $T[\cdot] = S(\cdot \oplus r) \oplus s$

1. $l = \lfloor \log_2(r) \rfloor$
2. **for** x_1 **from** 0 **to** 255 **by** 2^{l+1} **do**
3. **for** x_2 **from** 0 **to** $2^l - 1$ **do**
4. $tmp_1 \leftarrow T[x_1 \oplus x_2] \oplus s$
5. $tmp_2 \leftarrow T[x_1 \oplus x_2 \oplus r] \oplus s$
6. $T[x_1 \oplus x_2] \leftarrow tmp_2 \oplus s$
7. $T[x_1 \oplus x_2 \oplus r] \leftarrow tmp_1 \oplus s$

8. **end**
9. **end**

Despite its practical interest, this algorithm cannot be used because it does not take the case $r = 0$ into account. This is problematic since the mask r must be uniformly distributed to ensure the SCA-resistance. Moreover, Algorithm 20 cannot be patched to take this case into account. Indeed, when r equals 0, the re-computation should apply the output mask s to every value in the table: $T[x] \leftarrow T[x] \oplus s$. However, for $r = 0$ and whatever the value of l, it can be checked that Steps 4 to 7 of Algorithm 20 perform twice the operation $T[x_1 \oplus x_2] \leftarrow T[x_1 \oplus x_2] \oplus s$. Thus, when r equals 0, Steps 2 to 9 apply the output mask s only to the half of the table values. Therefore the only solution to patch Algorithm 20 is to perform a particular re-computation when r equals 0. This would induce a dependence between the value of r and the execution time of the re-computation algorithm which, as remarked above, is a flaw with respect to first-order SCA.

13.3 The improved masking scheme

13.3.1 Description

Schramm and Paar's generic Scheme recalled in Section 13.2.1 is quite costly as it involves d table re-computations for each S-box access for each round of the cipher (which implies $160 \times d$ table re-computations for AES).

Therefore, Kai Schramm and Christof Paar propose in [204] an improvement of the method. In the new solution, d successive re-computations are still performed for the computation of the first masked S-box in the first round. Then, each time S must be applied on a new byte $m'_0 = z' \oplus \bigoplus_{i=1}^{d} m'_i$, a new masked S-box S^*_{new}, satisfying $S^*_{new}(x) = S(x \oplus \bigoplus_{i=1}^{d} m'_i) \oplus \bigoplus_{i=1}^{d} s'_i$ for every byte x, is derived from the previous S^* with a single re-computation. This re-computation firstly requires the computation of two values called *chains of masks* in [204] and denoted here by **icm** and **ocm**:

$$\mathbf{icm} = \bigoplus_{i=1}^{d} m_i \oplus \bigoplus_{i=1}^{d} m'_i \ , \tag{13.7}$$

$$\mathbf{ocm} = \bigoplus_{i=1}^{d} s_i \oplus \bigoplus_{i=1}^{d} s'_i \ . \tag{13.8}$$

Once the values of the chains of masks have been computed, the masked S-box S^*_{new} is derived from S^* by performing one single re-computation such that the following relation is satisfied for every x:

$$S^*_{new}(x) = S^*(x \oplus \mathbf{icm}) \oplus \mathbf{ocm} \ . \tag{13.9}$$

To construct an S-box S^*_{new} that satisfies (13.9), a re-computation algorithm may be called with the input parameters $(S^*, \mathbf{icm}, \mathbf{ocm})$. The variable **icm** removes the previous sum of input masks $\bigoplus_{i=1}^{d} m_i$ and adds the new sum of input masks $\bigoplus_{i=1}^{d} m'_i$ while **ocm** removes the previous sum of output masks $\bigoplus_{i=1}^{d} s_i$ and adds the new sum of output masks $\bigoplus_{i=1}^{d} s'_i$.

For the entire AES implementation, this improved scheme replaces the $160 \times d$ table re-computations required in the generic scheme by $d + 159$ table re-computations. For $d \geqslant 2$, this represents a substantial gain.

13.3.2 The third-order flaws

We show hereafter that the computation of the chains of masks induces two third-order flaws. In fact, one obtains from (13.1) and (13.7) that the input chain of masks **icm** satisfies:

$$z \oplus z' = \mathbf{icm} \oplus m_0 \oplus m'_0 \ . \tag{13.10}$$

Since $z \oplus z'$ is a sensitive variable (because it depends on both the plaintext and the secret key), and since the variables **icm**, m_0 and m'_0 are manipulated by the implementation, this immediately gives a third-order flaw.

The second third-order flaw is derived as follows: from (13.2) and (13.8) we deduce that the output chain of masks **ocm** satisfies:

$$S(z) \oplus S(z') \quad = \quad \mathbf{ocm} \oplus s_0 \oplus s'_0 \ . \tag{13.11}$$

This shows that the manipulation of **ocm**, s_0 and s'_0 gives a third-order flaw which leaks information on the sensitive variable $S(z) \oplus S(z')$.

To summarize, we have shown that the improved Schramm and Paar's scheme is vulnerable to third-order SCA for any value of d.

13.4 Attacks simulations

In previous sections, we have shown that an attacker who can obtain the exact values of 3 intermediate variables of the (generic or improved) Schramm and Paar's masking scheme, can recover the value (or a part of the value) of a sensitive variable. This is sufficient to show that the scheme is theoretically vulnerable to third-order SCA. However, the physical leakage of an implementation does not reveal the exact values of the variables manipulated but a noisy function of them. Thus, a leakage model must be considered when SCA attacks are addressed. In this section, we firstly recall two generic d^{th}-order SCA attacks in a classical leakage model. Then we apply each of them against Schramm and Paar's scheme and we present results of attack simulations.

13.4.1 Leakage model

We consider the Hamming weight model with Gaussian noise *i.e.* we assume that the physical leakage L_i resulting from the manipulation of a variable V_i satisfies:

$$L_i = \varepsilon \cdot \mathrm{HW}(V_i) + B_i \ , \tag{13.12}$$

where the B_i's have independent Gaussian distributions $\mathcal{N}\left(0, \sigma^2\right)$.

In the next section, two generic d^{th}-order SCA attacks are recalled. Both of them assume that there exists a d-tuple $(V_1, \ldots, V_d)$ of variables manipulated by the algorithm which is correlated to a sensitive variable $Z = f(X, K)$. The V_i's depend on the sensitive variable Z and on random values generated during the execution of the algorithm. The random values involved in the V_i's are represented by a random variable R which is assumed to be uniformly distributed over $\mathcal{R}$. Thus, the V_i variables considered in the rest of the chapter can be expressed as functions of (Z, R), which will be denoted $V_i(Z, R)$.

Two attacks are described in the next section which aim at recovering the value k^* taken by K on the target implementation.

13.4.2 Higher-order attacks

We recall hereafter two generic higher-order SCA attacks: the higher-order DPA and the higher-order profiled SCA.

13.4.2.1 Higher-order DPA

As explained in Section 3.7.1, a d^{th}-order DPA first applies a combining function $\mathcal{C}$ (*e.g.* the product or the absolute difference) to the d leakage signals $L_1, \ldots, L_d$. Then it estimates the correlation between the combined signal $\mathcal{C}(L_1, \ldots, L_d)$ and a model $\mathsf{M}(X, k) = \hat{\varphi} \circ f(X, k)$ of this signal, according to a guess

k on the value of the secret key part k^*. According to the analysis in Chapter 6, in the Hamming weight model, a good choice for the so-called prediction function $\hat{\varphi}$ is:

$$\hat{\varphi}(z) = \mathrm{E}_R\left[\mathcal{C}\left(\mathrm{HW}\left(V_1\left(z,R\right)\right),\ldots,\mathrm{HW}\left(V_d\left(z,R\right)\right)\right)\right] \ . \tag{13.13}$$

The experiments presented in the next section use a variant of classical correlation attack that evaluates (for every key guess k) the correlation:

$$\rho_k = \hat{\rho}\left[\mathsf{M}(X,k), \mathrm{E}\left[\mathcal{C}\left(L_1,\ldots,L_d\right)|X\right]\right] \ . \tag{13.14}$$

That is, the combined leakage is averaged for every value of X before correlation. For the different attacks we tried both the product combining (3.24) and the generalized absolute difference combining (3.26) (see Section 3.7.1) and we selected the one yielding the most efficient attack.

13.4.2.2 Higher-order profiled SCA

In a higher-order profiled attack (see Section 3.7.3), the attacker has some estimations of the leakage distribution according to the processed variables. In this context, we assume that the attacker knows the exact distributions of the L_i's given the V_i's. The knowledge of these distributions allows him to compute the probability density function $g(.|x,k)$ of $(L_i)_i$ given $X = x$ and $K = k$. As the V_i's satisfy (13.12) for every i, assuming that the B_i's have independent Gaussian distributions, $g(.|x,k)$ satisfies:

$$g(l_1,\ldots,l_d|x,k) = \frac{1}{\#\mathcal{R}} \sum_{r\in\mathcal{R}} \prod_{i=1}^{d} \phi_{0,\sigma^2}\left(l_i - \varepsilon\cdot \mathrm{HW}(V_i(f(x,k),r))\right) \ , \tag{13.15}$$

where ϕ_{0,σ^2} denotes the pdf of the Gaussian distribution $\mathcal{N}(0,\sigma^2)$ (see Section 2.4).

Then, the attack consists in estimating the likelihood λ_k of the key guess k given the observations of the leakage $(l_{j,1},\ldots,l_{j,d})_j$ corresponding to the plaintexts $(x_j)_j$:

$$\lambda_k = \prod_j g(l_{j,1},\ldots,l_{j,d}|x_j,k) \ . \tag{13.16}$$

13.4.3 Results

We launch hereafter the two attacks described in Section 13.4.2 against the Schramm and Paar's generic and improved schemes. Each attack is a third-order SCA targeting three variables V_1, V_2 and V_3 appearing during the computation. The measurements $(l_{j,1}, l_{j,2}, l_{j3})_j$ are simulated according to the noisy Hamming weight model (13.12) with $\varepsilon = 3.72$ and $\sigma = 1.96$[19].

Before presenting the results, we recall that during the first round, every S-box input Z satisfies $Z = X \oplus k^*$, where X is a plaintext byte and k^* is a secret key byte.

13.4.3.1 Attacks on the generic scheme

We have shown in Section 13.2.2 that a third-order flaw results from the manipulation of $V_1 = m_0$, $V_2 = S_d(e_1)$ and $V_3 = S_d(e_2)$. Hereafter, we apply our attacks for $e_1 = 0$ and $e_2 = 1$. In this case, we recall that V_1, V_2 and V_3 satisfy:

$$\begin{aligned} V_1(Z,R) &= Z \oplus \mathbf{m} \ , \\ V_2(Z,R) &= S(\mathbf{m}) \oplus \mathbf{s} \ , \\ V_3(Z,R) &= S(\mathbf{m}\oplus 1) \oplus \mathbf{s} \ , \end{aligned}$$

where $Z = X \oplus k^*$ and where R denotes the pair $(\mathbf{m},\mathbf{s})$ of involved random masks.

Figure 13.1 shows the result of a third-order DPA which uses the product as combining function to exploit the flaw. The different curves represent the different key guesses; the curve corresponding to

[19] These values are the ones used by Kai Schramm and Christof Paar in their experiments [204].

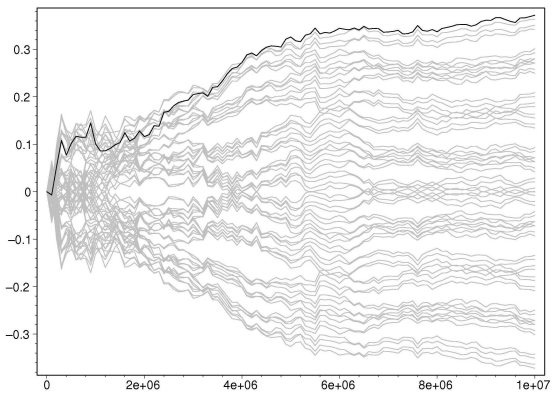


Figure 13.1: **Third-order DPA:** evolution of the correlation (Y-axis) over an increasing number of leakage measurements (X-axis).

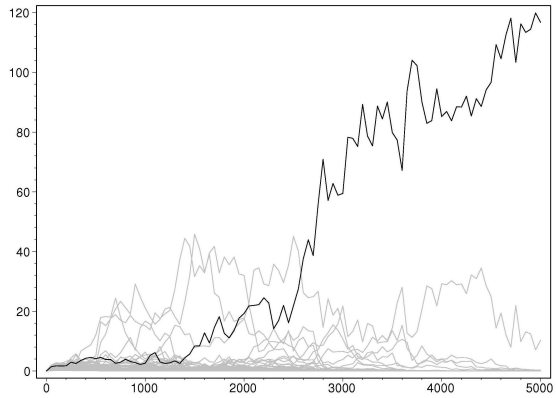


Figure 13.2: **Third-order profiled SCA:** evolution of the likelihood (Y-axis) over an increasing number of leakage measurements (X-axis).

the correct key guess is plotted in black. We noticed that this curve also corresponds to three other wrong key hypotheses (additionally, four wrong key hypotheses result in correlation peaks with equal magnitude and opposite sign). It can be observed that the correlation for the correct key guess comes out after about 4.10^6 measurements. This implies that several millions of measurements are required to recover the secret key byte. However this assertion must be mitigated. Indeed, we noticed that the correlation curve corresponding to the correct key guess is quickly among the top curves, which implies a significant loss of entropy for the secret key value.

Figure 13.2 shows the results of a third-order profiled SCA. The likelihood of the correct key guess is clearly remarkable after 2800 measurements which shows that the third-order profiled SCA is much more efficient than the third-order DPA.

13.4.3.2 Attacks on the improved scheme

As argued in Section 13.3.2, a third-order flaw results from the manipulation of $V_1 = \mathbf{icm}$, $V_2 = m_0$ and $V_3 = m'_0$. We recall that these 3 variables satisfy:

$$\begin{aligned} V_1(Z,R) &= Z \oplus m_0 \oplus m'_0 \ , \\ V_2(Z,R) &= m_0 \ , \\ V_3(Z,R) &= m'_0 \ , \end{aligned}$$

where $Z = X \oplus k^* \oplus X' \oplus k^{*\prime}$ and where R denotes the pair (m_0, m'_0) of involved random masks. An attack targeting this flaw therefore aims at recovering the secret value $k^* \oplus k^{*\prime}$.

Remark 13.1. The flaw above corresponds to a "standard" third-order flaw since the sensitive variable Z is masked with two random masks (m_0 and m'_0).

Figure 13.3 shows the result of a third-order DPA which uses the absolute difference as combining function and Figure 13.4 shows the result of a third-order profiled SCA. The third-order DPA allows to recover the targeted secret key part with 2.10^5 measurements, whereas the third-order profiled SCA only requires 600 measurements.

13.4.3.3 Results analysis

We performed each attack 100 times and we recorded the obtained success rates. Table 13.1 summarizes the number of measurements required to reach a success rate equal to 50%. We list hereafter our observations:

— The most efficient of our third-order DPA requires a number of measurements which is only 10 times larger than for a first-order DPA against an unprotected implementation.

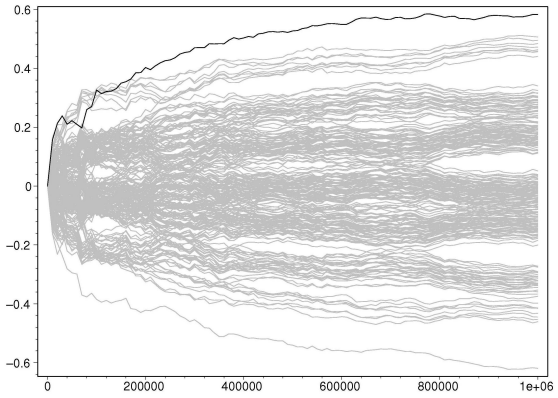


Figure 13.3: **Third-order DPA:** evolution of the correlation (Y-axis) over an increasing number of leakage measurements (X-axis).

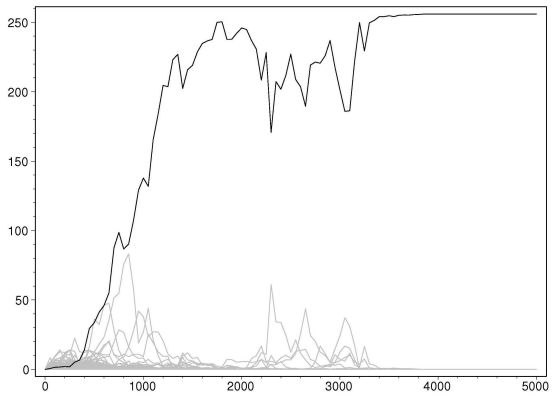


Figure 13.4: **Third-order profiled SCA:** evolution of the likelihood (Y-axis) over an increasing number of leakage measurements (X-axis).

Attack	Implementation	Measurements
No countermeasure	DPA	100
S&P generic scheme	3O DPA	6.10^6
S&P generic scheme	3O profiled SCA	2.10^3
S&P improved scheme	3O DPA	10^5
S&P improved scheme	3O profiled SCA	10^3

Table 13.1: Number of measurements required to achieve a success rate of 50%.

— The third-order profiled SCA is much more efficient than the third-order DPA. This result was predictable: the third-order profiled SCA exploits all the information provided by the 3 leakage signals to derive the likelihood of a key candidate, whereas combining the 3 leakage signals in a single signal implies a significant loss of information whatever the combining function. However, the adversary model of third-order profiled SCA is very strong and in such a model, an attacker may break an implementation without exploiting the kind of flaws exhibited in this chapter.

— The third-order profiled SCA requires a quite small number of measurements. This shows the practicability of such an attack when the attacker owns a profile that matches well the real leakage of the implementation.

— The third-order DPA is fairly efficient against the improved scheme but is less suitable against the generic scheme. This is not surprising: combining techniques have been especially designed to attack Boolean masking and the flaw in the improved scheme involves a doubly masked variable and two Boolean masks. The flaw in the generic scheme has not this particularity and the combining techniques involved in this chapter are less appropriate to exploit it.

Chapter 14

Conclusions and perspectives

In this second part, we have investigated one of the most widely used countermeasures to side channel analysis: data masking. Our investigations focused on masking schemes for block ciphers composed of linear transformations and non-linear S-boxes. First, we introduced a model to prove the security of masking schemes with respect to side channel analysis of a given order. In this security model, we have shown that the problem of securing a block cipher could be reduced to the problem of securing a single S-box computation. This allowed us to concentrate our investigations on masking schemes for S-boxes.

At first, we proposed new designs of generic masking schemes for S-boxes with provable security against first-order and second-order SCA. On the one hand, we proposed a first-order masking scheme which performs an S-box computation on masked data on-the-fly and which does not require any RAM allocation. Such an asset comes at the price of a significant overhead in computation time compared to the usual table re-computation method. Our scheme is therefore merely interesting for low cost implementations without strong timing constraints. On the other hand, we proposed two second-order masking schemes. Proving their security was trickier than for a first-order scheme. For this purpose, we introduced a methodology which could be used to prove the security of future proposals. We also described an improvement of our methods that takes advantage of the device architecture to speed the computation up. Our schemes provide worthwhile alternatives to the other existing methods and they enable different time-memory tradeoffs. Moreover, for a software implementation on an architecture with large bit-size (*e.g.* 16, 32), our improved solutions are the most efficient in the literature.

Then, we have designed a generic scheme combining higher-order masking and operation shuffling for implementations of block ciphers. This design followed a different approach compared to our previous works about masking. Instead of aiming at absolute security with respect to a given order, the latter scheme rather aims at practical security. In fact, some advanced DPA attacks have been shown especially effective in breaking masking and shuffling schemes. Our scheme advantageously makes use of higher-order masking and shuffling to ensure a chosen resistance level against these attacks.

Our investigations on masking schemes also include the cryptanalysis of two masking schemes proposed in 2006 at the conferences CT-RSA and CHES respectively. For both attacks, the underlying scheme does not satisfy the claimed security order. First, we exhibited a flaw in the first-order masking scheme based on the Fourier transform proposed at CHES 2006. We explained how to exploit such a flaw to mount a first-order DPA, and we reported successful practical attacks on two different S-box implementations using this scheme. We then proposed an improvement of the original countermeasure and we prove its security against first-order SCA. Secondly, we attacked the only higher-order masking scheme for block ciphers proposed in the literature which was published at CT-RSA 2006. This scheme merely consists in a generalization of the table re-computation method to the d^{th}-order for a chosen parameter d. We have invalidated this approach by exhibiting several third-order attacks breaking the scheme irrespective of the value of d. Our attacks have been validated by simulations under common assumptions about the leakage.

For future works, masking schemes for block ciphers with provable security against higher-order SCA needs more investigations. A secure higher-order masking scheme for a logical AND operation has been proposed in [126]. Based on this scheme, every circuit can be secured against d^{th}-order SCA for any

chosen d since every circuit can be decomposed in Boolean functions performing XORs and ANDs. However, this solution induces a significant size-overhead for hardware implementations and it is clearly impractical for software implementations. Besides, masking schemes for block ciphers only exist at the second order and have an important timing complexity. The research for alternative methods with better performances as well as higher security orders is therefore an important open issue. Another line for further research would be the investigation of other kinds of randomization techniques in a view of improving the practical resistance to higher-order side channel analysis. For instance, the recently proposed permutation table countermeasure [77, 193] may have a better practical resistance to higher-order SCA than Boolean masking.

Part III

Fault analysis

Chapter 15

Introduction to fault analysis

Contents

15.1 Introduction

Fault analysis is a cryptanalytic technique that takes advantage of errors occurring in cryptographic computations. Such errors can be induced in a device by physical means such as the variation of the power supply voltage, the increase in the clock frequency or an intensive lighting of the circuit. The erroneous results of the cryptographic computations can then be exploited in order to retrieve some information about the secret key.

We give in this chapter a general introduction to fault analysis. After a brief history, we discuss the existing techniques to inject faults and the expected effects on target devices. Then, we give an overview of fault attacks against block ciphers and against the RSA cryptosystem. Finally, we review common countermeasures to thwart these attacks.

15.2 Brief History

Several years ago, it was noticed that cosmic rays, which are energetic particles originating from outer space, could affect the behavior of electronic devices present in aircrafts and space vehicles [250]. Notably,

such particles induce some faults in devices memories which may result in the corruption of the different program executions. For several years, this problem was studied in the context of aviation and space equipment without imagining that it could be relevant in other situations. But what if such a fault occurs during the execution of a security protocol and, in particular, during the execution of a cryptographic algorithm? The firsts to raise this question were Dan Boneh, Richard DeMillo and Richard Lipton in 1996. The answer was devastating attacks against RSA and other public key signature and authentication schemes [50, 51]. Soon, the approach was extended to symmetric cryptography by Eli Biham and Adi Shamir that described an attack on DES [44]. These new kinds of attacks aroused the interest of the cryptographers and hardware designers. In subsequent years, several reports and papers were published that investigated physical means of injecting faults [30, 119, 149, 211] and extended fault analysis to other cryptosystems [41, 42, 107, 188].

Today, fault analysis is an important branch of embedded cryptography. An international annual workshop named *Fault Diagnosis and Tolerance in Cryptography (FDTC)* has been dedicated to its study since 2004. Fault analysis is also considered as an important threat in the smart card industry. It is taken into account in the development of secure embedded devices by including some countermeasures to their design. In addition, laboratories performing security evaluations experiment fault attacks to check the resistance of smart cards products.

15.3 Fault injection techniques

Several techniques are able to disrupt the execution of a microprocessor. The most commonly used are *power and clock glitches* [30, 32] and *pulses of light* [32, 211].

A power glitch is an instantaneous sizeable variation of the power supply voltage. Smart cards are supplied at a voltage constant to 5 volts and usually tolerate variations of $\pm 10\%$. A more significant variation over a short period may provoke the switching of the voltage of several gates in the circuit thereby disrupting the current execution. Similarly, a temporary increase in the clock frequency may result in an early start for the execution of the subsequent instruction, which induces a skipping of the current instruction. Glitches attacks are very simple to put into practice and they do not require any particular equipment. Fortunately, glitches can be avoided by the addition of voltage threshold sensors and high frequency detectors to the chip. Modern smart cards usually include such hardware mechanisms which protect them against glitches attacks.

Electronic circuits are also sensitive to light. A brief and intensive pulse of light may induce faults in the exposed gates. To perform a light attack on a smart card, a part of its package must be removed in order to expose the chip (see Section 1.4.4). The first light attack was described in [211] by Sergei Skorobogatov and Ross Anderson. In this paper, the authors show how to disrupt the execution of a depakaged microchip using a camera flash amplified through a probing station. Such equipment is quite cheap, which makes the attack relatively easy to put into practice (note however that the depackaging is a difficult task for a novice). Light attacks can be enhanced by the use of a laser (see for instance [32]). Using a laser gives two significant advantages. First, a camera flash is easily detected by light sensors (which are often included in modern chips) while a laser beam can escape such detection. Moreover, a laser beam enables a precise and localized fault injection which is a requirement for several attacks. Note, however, that a laser is much more expensive than a simple camera flash with a probing station.

Many other ways exist to inject faults in electronic devices (see [180] for an overview). In particular cosmic rays [119], electromagnetic fields [149, 195], infrared radiations and heat spikes [119] have been successfully experimented.

15.4 Fault effects

Different effects may be observed following a fault injection. As explained above, a laser-based fault injection disrupts a local area of the chip (*e.g.* a few RAM cells, a memory bus, a few CPU registers or a cryptographic coprocessor).

The effect of a disruption of the RAM is difficult to predict in general. Indeed, RAM is usually a few

kilobytes of data and the attacker does not know *a priori* the content of the different areas. This makes it difficult to target a precise variable and to anticipate the effect of an induced fault. The disruption of a memory bus on the other hand has a more predictable effect. Two typical cases are observed in practice: (1) The bus is disrupted while reading/writing data; this has the effect of corrupting the current intermediate variable, (2) the bus is disrupted during an instruction fetch while looking up the codop of the next instruction. In the latter case, a wrong instruction is subsequently executed which may have several effects: (i) simple corruption of the computation (*e.g.* a XOR is performed rather than an ADD), (ii) skipping of a function execution (if the subsequent instruction was a branch), (iii) random branch in the software code (if the induced codop is a branch) and so on. It is worth noting that certain chips use different buses for the different memories and an attacker may choose the target bus according to the desired effect. Disrupting the CPU may also provoke similar effects. If working registers are corrupted then intermediate results are likely to be erroneous. If some special purpose registers are corrupted then some branches in the code may be induced. For instance, a typical target is the *program counter* register. Disrupting its value amounts to performing a jump in the code. Another sensitive register is the *stack pointer*. If it is corrupted then a wrong memory area is taken as the stack which may in particular induce a bad branch at the end of the function being executed. Finally, attacking a cryptographic coprocessor directly leads to the disruption of the cryptographic computation. Cryptographic coprocessors are therefore common targets of fault attacks.

Contrary to a laser, a camera flash does not make it possible to target a local area of the chip and it may induce some (or many) of the effects described above. Likewise, power glitches may affect any part of the chip. On the other hand, clock glitches have a predictable effect which is instruction skipping.

Remark 15.1. The kinds of faults described above are *transient* since they only affect one execution of the device. It is worth noting that some faults may be induced whose effect is *permanent* if the content of the EEPROM or the ROM is modified. The ROM may be overwritten using laser cutting [26] and the EEPROM can be modified using microprobing needles [26]. However such fault injections are very chancy and they are only possible at the cost of high-tech microelectronic equipment. In addition, to obtain any desired fault effect an attacker must precisely know the content of the memory which seems unlikely.

Remark 15.2. Fault attacks do not only target cryptographic computations. Instruction skipping attacks are widely used to overcome security mechanisms in general. A classical example is the skipping of the PIN verification enabling unauthorized users to execute commands.

15.5 Fault models

The different fault injection techniques and effects presented above enable the corruption of cryptographic computations performed on an unprotected microchip. The obtained erroneous results can then be analyzed to break the underlying cryptosystems. Some fault attacks are very generic: they only require that the computation be corrupted in any way and within a large time interval during the execution. This is typically the case of the *Bellcore attack* on RSA with CRT (see Section 1.3.4.2). However, most fault attacks require more specific assumptions about the fault effect. They are based on a so-called *fault model* that formally describes the effect of the fault.

A fault model typically specifies what data have been corrupted by the fault and how the fault affects these data. Common models consider the corruption of a bit, a byte or a data word (whose length depends on the device architecture). When the fault affects an intermediate variable composed of several bits, bytes or words, its position may be either fixed (chosen or not) or randomly spread over the whole variable.

Two possible effects on the data are usually considered. First, the fault may induce a random switching of the corrupted data. In that case, the faulty data is modelled as a uniformly distributed random variable. On the other hand, the fault may erase the data by a given fixed value (usually the all-0 word or the all-1 word). This fault model is known as *stuck-at fault model*. It is mostly relevant in the context of light attacks due to their physical effect.

Finally, the time of the fault injection may not be precisely controlled by the attacker due to dynamic

variations in the execution time (*e.g.* desynchronization mechanisms). For this reason, some fault models consider a set of intermediate variables among which the corrupted one is randomly picked up.

15.6 Fault analysis against block ciphers

Fault analysis against block ciphers can be divided into two mains categories: *differential fault analysis* and *ineffective fault analysis*.

15.6.1 Differential fault analysis

Differential fault analysis (DFA) was introduced by Eli Biham and Adi Shamir in [44]. The principle of DFA is to infer information about the secret key by exploiting the differences between the ciphertexts obtained from correct and faulty computations. To a large extent it is based on *differential cryptanalysis*, a cryptanalytic technique against block ciphers also introduced by Eli Biham and Adi Shamir a few years before [43]. We recall hereafter the basic principle of DFA.

Let us consider a block cipher ending by a non-linear layer composed of several S-boxes $(\mathsf{S}_i)_i$ followed by the bitwise addition of a key $\mathbf{k}$. The principle of DFA is to inject a fault in the computation before the final S-boxes, which results in a faulty ciphertext $\widetilde{C}$. Then, based on the pair of correct-faulty ciphertexts $(C, \widetilde{C})$, the attacker infers some information on $\mathbf{k}$. For such a purpose, the attacker assumes a fault model from which he deduces a set $\mathcal{D}_i$ of possible bitwise differences between the correct input and the faulty input of S_i. For example, the fault model could be a simple bit switch in input of S_i, which would imply $\mathcal{D}_i = \{2^i; i \leqslant n\}$ (assuming n-bit S-box inputs). The attacker then makes a guess k on the value of the subkey $\mathbf{k}_i$ which is added to the i^{th} S-box output. For a given pair $(C, \widetilde{C})$, this guess leads to an expected differential $\delta_i(k)$ in input of S_i:

$$\delta_i(k) = \mathsf{S}_i^{-1}(C_i \oplus k) \oplus \mathsf{S}_i^{-1}(\widetilde{C_i} \oplus k) \ .$$

If the differential $\delta_i(k)$ is not included in the set of possible input differentials $\mathcal{D}_i$, then the guess k is incorrect (since by definition $\delta_i(\mathbf{k}_i) \in \mathcal{D}_i$). In this way, several wrong key guesses are discarded. After a few pairs $(C, \widetilde{C})$, only the correct guess remains. The lower the cardinal of $\mathcal{D}_i$, the higher the number of wrong guesses discarded per pair $(C, \widetilde{C})$. Besides if $\#\mathcal{D}_i$ is maximal, no key guess is discarded and the attack fails. Consequently, the fault model is usually chosen to minimize $\#\mathcal{D}_i$. This particularly implies that (i) DFA targets last rounds of the cipher (otherwise the error propagation implies that $\#\mathcal{D}_i$ is high/maximal) (ii) the more accurate the fault model, the more efficient the attack.

Note that for a Feistel cipher (*e.g.* DES, see Section 1.2.3.1), the attack is slightly different: the attacker knows the correct/faulty inputs of the last Feistel function but ignore the outputs. Based on a key guess he predicts the differential of an S-box output which is expected to match a certain set (depending on the fault model).

The original attack of [44] targets DES. It was subsequently applied to other block ciphers such as AES [48, 67, 89, 107, 142, 188, 227], IDEA [75] and CLEFIA [68, 226]. These different attacks more or less follow the above principle. They differ in the considered fault models as well as in the way to discriminate key guesses. DFA was also extended to target early rounds of DES in [123]. The proposed attack aims to generate correct-faulty collisions *i.e.* pairs of plaintexts (P, P') such that their correct and faulty encryptions yield $C = \widetilde{C'}$. The pair (P, P') can then be used to infer information about the first round key just as $(C, \widetilde{C})$ for the last round key in classical DFA.

15.6.2 Ineffective fault analysis

Contrary to DFA, ineffective fault analysis (IFA) [48, 73, 198] does not exploit the faulty outputs but the simple information of whether or not the injected fault yields an erroneous ciphertext. IFA assumes a stuck-at fault model: the attacker is able to set an intermediate variable to a fixed value. This value is either known to the attacker or it is guessable (in general one assumes this value to be zero). If the result is erroneous or if a fault is detected, the attacker knows that the intermediate variable was different from the induced value. Obtaining this information for several encryptions enables key-recovery.

Remark 15.3. Fault attacks exploiting the behavior of faulty computations have been previously introduced against modular exponentiations that are known as *safe-error attacks* (see Section 15.7.3). Contrary to IFA, safe-error attack do not particularly require a stuck-at fault model.

15.7 Fault analysis against RSA

Fault analysis on RSA can also be divided into two categories: the attacks exploiting the faulty outputs and the *safe-error attacks*. Among the former, one counts the *Bellcore attack* against an RSA-CRT implementation and other attacks against an implementation in standard mode (*i.e.* without CRT).

15.7.1 The Bellcore attack

The first fault attacks against RSA were introduced in the founding paper of Dan Boneh, Richard DeMillo and Richard Lipton [50]. The most powerful, that is usually referred to as the Bellcore attack [50], targets an RSA-CRT implementation (see Section 1.3.4.2). It consists in corrupting one of the two CRT exponentiations, *e.g.* the one modulo p. The RSA computation thus results in a faulty signature $\tilde{s}$ that is correct modulo q (*i.e.* $\tilde{s} \equiv s \bmod q$) and corrupted modulo p (*i.e.* $\tilde{s} \not\equiv s \bmod p$). This implies that the difference $\tilde{s} - s$ is a multiple of q but is not a multiple of p, and hence we have

$$\gcd(\tilde{s} - s, N) = q \ .$$

Therefore, a pair signature-faulty signature provides a way to factorize N and consequently to fully break RSA. In fact, a pair message-faulty signature is sufficient to mount the attack since we have [132]:

$$\gcd(\tilde{s}^e - m, N) = q \ .$$

In this way, RSA can be broken with a single faulty computation.

15.7.2 Fault analysis against standard RSA

RSA implemented in standard mode (*i.e.* without CRT) is also vulnerable to fault analysis. The first attack on RSA in standard mode was proposed by Dan Boneh *et al.* in the original paper [50]. Their attack assumes a single bit flip in an intermediate variable of a binary exponentiation algorithm. Similar attacks were subsequently proposed in [31, 180]. In [52], an attack is described which assumes a random fault in a chosen intermediate result. These different attacks enable the recovery of the secret exponent. An attack based on a corruption of the exponent was described in [31] and generalized in [134]. It assumes the corruption of few bits of the secret exponent which can then be recovered. The first fault attack targeting the public modulus was published in [205] which was then improved in [172]. These attacks do not enable the recovery of an RSA private key but they aim to fool a signature verification by the target device in order that it accepts a wrong signature. The first fault attack fully breaking an RSA in standard mode based on the corruption of the modulus was presented in [56]. This attack assumes a random modification of the modulus before the exponentiation. Improved attacks targeting RSA public modulus were then published in [37, 39]. These attacks are based on the random corruption of one byte of the modulus at a chosen step during the exponentiation. Finally, an attack assuming a skipping of a multiplication of the RSA computation has been published in [202].

The different attacks reviewed above all require several faulty signatures to fully recover the key. This requirement varies depending on the attack and the fault model and it is sometimes quite huge. However, these attacks still constitute a practical threat and the implementation of RSA in standard mode must also be secured against fault analysis.

15.7.3 Safe-error attacks

Safe-error attacks were introduced by Sung-Ming Yen and Marc Joye in [246]. Their principle differs from traditional fault attacks that exploit erroneous results. Here, the attack exploits the behavior of a

corrupted computation, *i.e.* it checks whether or not the induced fault effectively provokes an erroneous result. Depending on the algorithm, a fault injection may have no effect for some secret key values and may cause a corruption for others. In that case, simply observing whether the computation was corrupted or not reveals information on the secret key. Such attacks are especially threatening since they bypass classical fault analysis countermeasures that return an error in case of fault detection. Among these attacks, two categories can be distinguished: the *C-safe-error attacks* [248] that target dummy operations and the *M-safe-error attacks* that target registers copies [143, 246].

15.8 Countermeasures to fault analysis

We have seen in the previous sections that unprotected implementations of cryptographic algorithms are highly vulnerable to fault attacks. As a matter of fact, some countermeasures exist to thwart them. Some of these countermeasures are generic and can be applied irrespective of the algorithm and the device. Others are included in the hardware during the design of the chip and are independent of the cryptographic algorithm. Finally some are dedicated to protect a given (class of) cryptographic algorithm (*e.g.* block ciphers, RSA).

15.8.1 Generic countermeasures

15.8.1.1 Computation doubling or inversion

A straightforward way to protect any algorithm against fault analysis is simply to perform the computation twice and to check that the same result is obtained. In case of an inconsistency, an error message is returned, thus preventing the exposure of the faulty result. A variant consists in verifying an encryption by means of a decryption (or *vice versa*). Such a countermeasure can be easily implemented in hardware and in software at the cost of a doubling of the execution time (or of the circuit size). These countermeasures are suitable for fast algorithms such as block ciphers, but when a public key cryptosystem such as RSA must be implemented, a doubling of the execution time (or of the circuit size) often becomes prohibitive.

15.8.1.2 Desynchronization

Desynchronization of the computation is widely used as a side channel attack countermeasure that can be included in hardware or in software (see Section 3.6). Desynchronization also complicates fault attacks that have strong synchronization constraints *e.g.* requiring the corruption of a precise intermediate variable or the skipping of a precise instruction (see for instance [24]).

15.8.2 Hardware countermeasures

15.8.2.1 Physical sensors

Some sensors are usually included in modern chips in order to detect important voltage and frequency variations. Such detectors usually prevent fault attacks based on power glitches and clock glitches. Light sensors are also included to detect light attacks. Such sensors are usually very effective in detecting a general light induction such as provoked by a camera flash. However, a laser beam may still be induced at some precise locations over the chip leading to undetected faults. An attacker can then program its laser to perform an automatic scan of the chip in order to detect unprotected locations. Nevertheless, modern smart cards are often programmed to erase their memory when a certain number of light attacks have been detected, thus rendering such a scan useless.

15.8.2.2 Active shield

Another common hardware protection is to include an *active shield* that is a metal mesh covering the chip in which all paths are continuously monitored [149]. Any disconnection while the chip is powered is

instantaneously detected. Such shields are initially devoted to thwarting probing attacks but they also render light attacks more complicated by making the depackaging of the chip harder.

15.8.2.3 Error detection/correction units

A natural countermeasure to fault attacks is the inclusion of error detection units or error correction units in the circuit. *Coding theory* has been investigated for several years to find some way to detect and correct random errors occurring in digital communications. It is therefore natural to turn to such codes in the context of fault analysis resistance [98,125]. Some modern smart cards include error detection/correction units in order to protect the integrity of their memories. However, the security offered by such a solution is only partial since there is still a small chance that errors will go undetected. While keeping an implementation overhead lower than the one of computation doubling, the non-detection probability is usually non-negligible. Nevertheless, such a protection renders harder the task of the attacker that must significantly increase the number of attempts to obtain one faulty execution.

15.8.2.4 Bus encryption

Bus encryption is widely used in secure embedded microchip [59, 90, 115]. The principle is to encrypt the data transferred on the bus *via* hardware encryption mechanisms. The purpose of bus encryption is essentially to prevent probing attacks and not fault attacks. However it does prevent fault attacks based on the corruption of a limited number of bits of a memory transfer.

15.8.2.5 Dual-rail logic

Dual-rail logic is usually involved in protecting circuits against side channel analysis (see Section 3.6) but it may also be useful to detect faults [206]. In a dual-rail circuit, each wire is replaced by a pair of wires carrying complementary electrical signals in order to reduce the side channel leakage. A fault injection is likely to destroy this complementarity. Therefore, checking that the signals of the different pairs are in fact complementary makes it possible (with some probability) to detect fault injections.

15.8.3 Countermeasures dedicated to block ciphers

15.8.3.1 Error detection schemes

To design block ciphers implementation resistant to DFA one must include some error detection mechanisms. This may be done by including redundancy and coherence checking at the operation level. Many such schemes have been proposed for hardware implementations of block ciphers and in particular for AES [130, 137–139, 199]. A detection scheme for AES suitable for software implementation was recently proposed in [100]. None of these schemes provide a perfect security against fault analysis and their detection probabilities highly depend on the considered fault model. In general the offered complexity-security ratio is of the same order as the one for computation/circuit doubling [155].

15.8.3.2 Round doubling

As mentioned above, a straightforward way to protect any algorithm against DFA is computation doubling (or inversion). In case of block ciphers, the doubling of the execution time is conceivable. A more efficient solution is to double a limited number of rounds. In fact, most DFA techniques target the last few rounds of the block cipher. To thwart these attacks, one only needs to double the computation of these last few rounds thus saving computation time.

However, a question remains: how many rounds should be protected to obtain a good security level with respect to DFA? To answer this question, DFA on middle rounds of the cipher must be investigated. This issue has been addressed in [185] by Raphael Phan and Sung-Ming Yen for the AES block cipher. They apply block cipher cryptanalysis techniques to improve DFA on AES and exhibit some attacks against rounds 7, 6 and 5. Concerning DES, the original work by Eli Biham and Adi Shamir [44] described an attack that exploits a fault corrupting either round 16, 15 or 14 (and equivalently the end

of round 15, 14 or 13). In his PhD thesis, Mehdi-Laurent Akkar addresses this issue and describes some attacks on DES middle rounds [19]. However, most of the considered fault models are not realistic and it is not clear what an attacker could expect in practice.

Issue 15.1. *Assuming standard fault models, which rounds of DES are sensitive to differential fault analysis?*

This issue is investigated in Chapter 16 where we present an improved attack on DES middle rounds. In addition, the effectiveness of the presented attack is analyzed in different standard fault models.

15.8.3.3 Preventing ineffective fault analysis

Contrary to DFA, error detection schemes at the algorithmic level do not thwart IFA. Indeed, even an error detection leaks the desired information to the attacker: the fault injection effectively produced an error. A simple way to thwart IFA is to employ data masking. This countermeasure is often applied to protect implementations of block ciphers against side channel analysis (see Part II). Indeed, masking ensures that no single intermediate variable of the computation provides sensitive information. It is worth noting that masking does not ensure result integrity and is hence ineffective against DFA (see for instance [24, 54]).

15.8.4 Countermeasures dedicated to RSA

15.8.4.1 Public verification

A simple way to protect RSA against fault analysis is by verifying the signature s before returning it, namely by performing the following check:

$$m \stackrel{?}{=} s^e \bmod N \ .$$

This method offers a perfect security against fault analysis (provided that the check cannot be skipped) since a faulty signature is systematically detected. This countermeasure is efficient as long as e is small, but in the opposite case, it implies performing two exponentiations which doubles the time complexity of RSA. This overhead is clearly prohibitive in the context of low resource devices. Moreover, depending on the situation, the public exponent e may not be available (*e.g.* the Javacard API for RSA signature [222]). That is why many research works in the last decade have been dedicated to the search for alternative solutions. We review hereafter the main proposals that can be divided into two families: the *extended modulus based countermeasures* and the *self-secure exponentiations*.

15.8.4.2 Extended modulus based countermeasures

We present hereafter different countermeasures that all rely on the use of an extended modulus in order to add redundancy in the computation.

Shamir's trick and variants. A first solution to protect RSA with CRT was proposed by Adi Shamir [208]. It consists in performing the two CRT exponentiations with extended moduli $p \cdot t$ and $q \cdot t$ where t is a small random integer (typically of 32 to 80 bits depending on the desired security). Namely, one computes:

$$s_p^* = m^{d \bmod \varphi(p \cdot t)} \bmod p \cdot t$$

and

$$s_q^* = m^{d \bmod \varphi(q \cdot t)} \bmod q \cdot t \ .$$

The consistency of the computation is then checked by verifying that $s_p^* \bmod t$ equals $s_q^* \bmod t$. If no error is detected, the algorithm returns $\mathsf{CRT}(s_p^* \bmod p, s_q^* \bmod q)$. In its simplest form, this countermeasure does not protect the CRT recombination which enables a successful fault attack [30]. Several works have proposed variants of Shamir's countermeasure in order to deal with this issue [30, 47, 72].

Vigilant's scheme. In [242], David Vigilant proposed another countermeasure based on a modulus extension. The modulus is multiplied by $t = r^2$ for a small random number r. The message is then formatted as follows:

$$\hat{m} = \alpha m + \beta \cdot (1 + r) \bmod Nt$$

where (α, β) is the unique solution in $\{1, \ldots, Nt\}^2$ of the system:

$$\alpha \equiv \begin{cases} 1 \bmod N \\ 0 \bmod t \end{cases} \quad \text{and} \quad \beta \equiv \begin{cases} 0 \bmod N \\ 1 \bmod t \end{cases} .$$

Then, the exponentiation $s_r = \hat{m}^d \bmod Nt$ is performed. As shown in [242], s_r satisfies:

$$s_r = \alpha m^d + \beta \cdot (1 + dr) \bmod Nt .$$

Therefore, the signature can be recovered from s_r since we have $s = s_r \bmod N$ and the consistency of the computation can be verified by checking: $s_r \equiv 1 + dr \bmod t$. This method can be extended to protect RSA-CRT (see [242] for details).

Security considerations. The security of an extended modulus based countermeasure is not perfect. For instance, if a faulty message $\widetilde{m}$ satisfies $\widetilde{m} \equiv m \bmod t$ and $\widetilde{m} \not\equiv m \bmod N$, then the exponentiation of this message results in a faulty signature that is not detected. The non-detection probability of an extended modulus based countermeasure is roughly about 2^{-k} where k denotes the bit-length of the modulus extension t. Therefore, the greater k, the more secure the countermeasure. However, the greater k, the slower the exponentiation. This kind of countermeasure therefore offers a time-security tradeoff. A common choice for k is 64 bits which provides a fairly good degree of security. However, depending on the application, one may choose $k = 32$ (low security, more efficient exponentiation) or $k = 80$ (high security, less efficient exponentiation).

Remark 15.4. The security of the Vigilant scheme such as described in [242] is actually only of $k/2$ bits since a corruption of the exponent is detected with a probability close to $2^{-k/2}$ (see [242, Sect. 4.1]). This can be fixed by the on-the-fly computation of a cyclic redundancy code during the exponentiation in order to check the exponent integrity.

15.8.4.3 Self-secure exponentiations

For the countermeasures presented hereafter, the redundancy is no longer included in the modular operations but at the exponentiation level. Namely, the exponentiation algorithm provides a direct way to check the consistency of the computation.

Giraud's scheme. In [108, 109], Christophe Giraud proposed a fault analysis countermeasure for RSA which is based on the use of the Montgomery powering ladder. The Giraud's scheme takes advantage of the fact that this exponentiation algorithm works with a pair of intermediate variables (a_0, a_1) storing values of the form $(m^\alpha, m^{\alpha+1})$. At the end of the exponentiation the pair (a_0, a_1) equals (m^{d-1}, m^d) and the consistency of the computation can be verified by checking whether $a_0 \cdot m$ equals a_1. If a fault is injected during the computation, the coherence between a_0 and a_1 is lost and the fault is detected by the final check.

Boscher et al.'s scheme. The scheme by Arnaud Boscher, Robert Naciri and Emmanuel Prouff [55] is based on the *right-to-left square-and-multiply-always* algorithm [76] which was originally devoted to thwart simple power analysis (see Section 3.1). In [55], the authors observe that this algorithm computes a triplet (a_0, a_1, a_2) that equals $(m^d, m^{2^l - d - 1}, m^{2^l})$ at the end of the algorithm, where l denotes the bit-length of d. The principle of their countermeasure is hence to check that $a_0 \cdot a_1 \cdot m$ equals a_2 at the end of the exponentiation. Once again, in case of a fault injection, the relation between the a_i's is broken and the fault is detected by the final check.

The main drawback of the two previous schemes is that they both require the use of an exponentiation algorithm that performs 2 modular multiplications per bit of the exponent while other exponentiation algorithms require an average of 1.5 multiplications per bit of the exponent (and sometimes less).

Issue 15.2. *Is it possible to design a self-secure exponentiation algorithm that performs faster than* 2 *modular multiplications per bit of the exponent?*

In Chapter 17, we propose a new self-secure exponentiation that requires approximately 1.65 multiplications per bit of the exponent on average. Our solution thus represents an efficient alternative to the existing countermeasures.

15.8.4.4 Randomization techniques

The randomization techniques used to protect RSA against side channel analysis (see Section 3.6.3.2) may also be useful to counteract fault analysis against RSA in standard mode. In particular all the existing attacks against RSA in standard mode (see Section 15.7.2 can be prevented by randomizing the secret exponent. However, other fault attacks may exist that overcome the exponent randomization. It is therefore recommended to use a sound error detection scheme to protect both CRT and standard RSA implementations.

15.8.4.5 Preventing safe-error attacks

As explained in Section 15.7.3, safe-error attacks are based on the observation of whether or not an induced fault in fact provokes an erroneous result. As argued in [246], such attacks overcome standard fault analysis countermeasures involving checking procedures.

To prevent C-safe-error attacks [248] one must ensure that no dummy operation is conditionally performed depending on the secret key value. If for some reason this condition cannot be satisfied, C-safe-error attacks may be avoided by randomizing the exponent (see Section 3.6). To prevent M-safe-error attacks [246] one can either randomize the exponent or randomize the indices of the registers that are addressed by some exponent bits (see Section 17.4.2 for an example).

15.8.5 Implementation of coherence checks

As explained in Section 15.4, many kinds of fault injections can lead to instruction skipping. On the other hand, most of the presented countermeasures against fault analysis are based on the addition of redundancy in the computation in order to check its coherence afterwards. Such a check is hence a key point of the security of fault analysis countermeasures in practice. It is therefore important to implement coherence checks in such a way that they cannot be skipped.

Issue 15.3. *How can coherence checks be implemented in such a way that they cannot be skipped?*

In Chapter 18 we propose a solution to this issue that stops attackers from bypassing a coherence check by skipping some instructions. Together with a sound error detection scheme for the implemented algorithm, our solution thwarts any *second-order fault attack* which corrupts the computation and attempts to skip the coherence check.

Chapter 16

Differential fault analysis on DES middle rounds

Contents

16.1 Introduction

In his PhD thesis [19], Mehdi-Laurent Akkar investigates the application of differential cryptanalysis techniques to attack earlier rounds of DES. In a first place, the considered attacker is assumed to be able to induce a differential of its choice in the DES internal value at the end of some round. The last round key is recovered by guessing every 6-bit parts independently and by selecting, for each subkey, the candidate that produces the expected differential at the S-box output the more frequently. The obtained attacks are quite efficient but, as mentioned by the author, the fault model is not realistic. Mehdi-Laurent Akkar then applies this attack under two more realistic fault models: a single bit switch at a fixed position (in the left part of the DES internal state) and a single bit switch at a random position (in the right part of the DES internal state). For the fixed position bit error model, the attack needs a few hundred fault injections at the end of round 11 and it fails on round 9 (the attack on round 10 is not considered). For the random position bit error model, the attack needs a few dozen fault injections at the end of round 12 and it fails on round 11.

In this chapter, we generalize and improve the attack described by Mehdi-Laurent Akkar in [19]. We consider various realistic fault models for an error induced in the left part of the DES internal state, including the bit error model and the byte error model with chosen error position or random error position. As we will argue, disrupting the left part leads to better attacks than disrupting the right part. Moreover, we use more accurate distinguishers than the one proposed in [19]. In the common (chosen position) byte error model, our attack recovers the entire last round key with a 99% success rate using 9 faults on round 12, 210 faults on round 11 and 13400 faults on round 10. In the (chosen position) bit

error model, these numbers are reduced to 7, 11 and 290, respectively. These results allow us to exhibit some lower bounds on the number of rounds of DES that should be protected against fault analysis.

The results presented in this chapter have been published in the international workshop on *Cryptographic Hardware and Embedded Systems (CHES 2009)* [8].

16.2 Notations and fault models

16.2.1 Notations

The reader is referred to Section 1.2.3.1 for a description of the DES block cipher. We shall use hereafter the same notation as in Section 1.2.3.1: IP for the DES initial permutation, FP for the DES final permutation, F for the round function and f for the round *internal* function. We shall further denote the expansion layer by E, the eight S-boxes by $(\mathsf{S}_i)_{1\leqslant i\leqslant 8}$, the round permutation by P, the secret key by $\mathbf{k}$ and the r^{th} round key by $\mathbf{k}_r$. A ciphertext C is then computed from a plaintext P according to:

$$C = \mathsf{FP} \circ \left(\bigcirc_{r=1}^{16} F_{\mathbf{k}_r}\right) \circ \mathsf{IP}(P) \ ,$$

where:

$$F_{\mathbf{k}_r} : (L, R) \mapsto (R, L \oplus f_{\mathbf{k}_r}(R)) \ .$$

The output block of the r^{th} round shall be denoted as (L_r, R_r). Defining $(L_0, R_0) = \mathsf{IP}(P)$, we have $(L_r, R_r) = F_{\mathbf{k}_r}(L_{r-1}, R_{r-1})$ for every $r \leqslant 16$ and $C = \mathsf{FP}(L_{16}, R_{16})$.

Finally, E_i and P_i^{-1} denote the i^{th} 6-bit coordinate of the expansion layer E and the i^{th} 4-bit coordinate of the bit-permutation P^{-1}, respectively. Similarly, $\mathbf{k}_{r,i}$ shall denote the i^{th} 6-bit part of a round key $\mathbf{k}_r$. We hence have the equality:

$$\mathsf{P}_i^{-1}(f_{\mathbf{k}_r}(\cdot)) = \mathsf{S}_i(\mathsf{E}_i(\cdot) \oplus \mathbf{k}_{r,i}) \ . \tag{16.1}$$

16.2.2 Fault models

Our attack consists in corrupting some bits of the left part of the DES internal state at the end of the r^{th} round with $r \in \{9, 10, 11, 12\}$. We shall consider different fault models depending on the statistical distribution of the induced error. We first consider the *bit error model*: one and one single bit of the left part is switched. We also consider the *byte error model*: one byte of the left part is switched to a random and uniformly distributed value. Furthermore, the fault position may be either *chosen* by the attacker or *random* among the 32 bit-positions or the 4 byte-positions of the left part.

In the sequel, $\widetilde{L}_i$ and $\widetilde{R}_i$ will respectively denote the corrupted value of the left part L_i and the right part R_i at the end of round i and $\widetilde{C} = \mathsf{FP}(\widetilde{L}_{16}, \widetilde{R}_{16})$ will denote the faulty ciphertext. We shall further denote by ε the induced error that is defined as $\varepsilon = L_r \oplus \widetilde{L}_r$.

16.3 Attack description

16.3.1 General principle

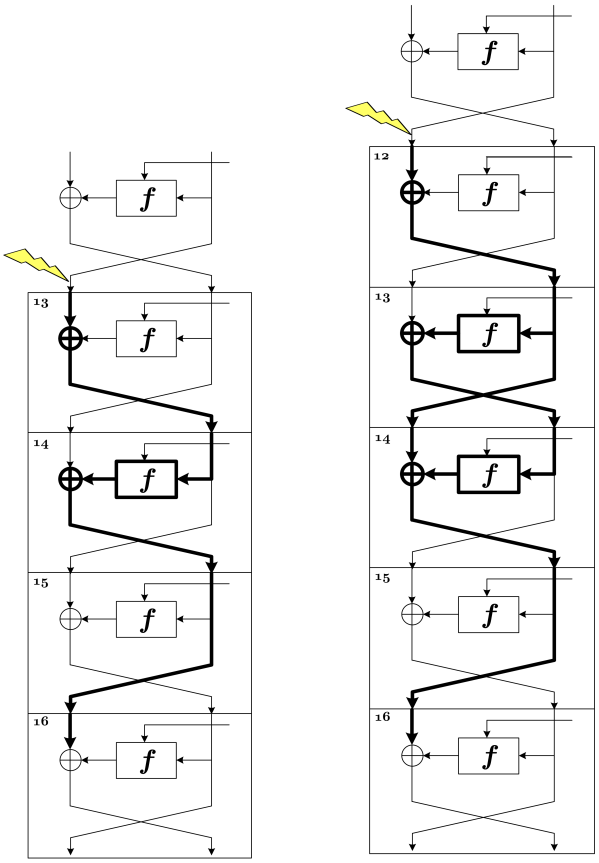
Let us denote by Δ the bitwise difference between the correct value and the corrupted value of the left part at the end of the fifteenth round: $\Delta = L_{15} \oplus \widetilde{L}_{15}$. Due to the Feistel scheme, we have the following relation:

$$R_{16} \oplus \widetilde{R}_{16} = f_{\mathbf{k}_{16}}(L_{16}) \oplus f_{\mathbf{k}_{16}}(\widetilde{L}_{16}) \oplus \Delta \ . \tag{16.2}$$

Based on (16.2), an adversary that knows Δ can mount a key recovery attack. The principle is to make a guess on the value of the round key $\mathbf{k}_{16}$. Then, given a pair of ciphertexts $(C, \widetilde{C})$, the attacker checks whether (16.2) is consistent for this guess. If not, the guess is discarded. In this way, $\mathbf{k}_{16}$ is non-ambiguously determined using a few pairs of ciphertexts. Due to the structure of f (see (16.1)), the attacker does not need to guess the entire round key $\mathbf{k}_{16}$ but he can guess and check each subkey $\mathbf{k}_{16,i}$ independently. When an error is induced in the final rounds, the differential Δ (or at least a part of it)

can be predicted according to the pair $(C, \widetilde{C})$ which enables the attack [44]. This is no more the case for an error induced in a middle round; in that case the attack must be extended.

As noted in [19], if an error ε is induced in the left part at the end of the thirteenth round then Δ equals ε. Therefore, an attacker that is able to induce a chosen (or at least known) error in L_{13} can apply the previous attack. For a fault induced in the left part during an earlier round, the equality $\Delta = \varepsilon$ does not hold anymore. However the statistical distribution of Δ may be significantly biased (depending on the fault model and the round number). Indeed, as illustrated in Figure 16.1, a fault injected in the left part skips one round before propagating through the function f. Besides, the error propagation path from L_r to L_{15} sticks through the function f only once for $r = 12$, twice for $r = 11$, etc. This is quite low considering the slow diffusion of the function f. As a result, a fault induced in L_r may produce a differential Δ with a distribution that is significantly biased. As described hereafter, this bias enables a key recovery attack based on a statistical distinguisher.



(a) From L_{12} to L_{15}. (b) From L_{11} to L_{15}.

Figure 16.1: Error propagation paths.

Remark 16.1. From Figure 16.1, it can be noticed that the injection of an error ε in L_r is equivalent to the injection of ε in R_{r+1}. This demonstrates the relevance of attacking the left part rather than the right one. Besides, this explains why the attack on the right part described in [19] is inefficient compared to the one on the left part on the same round.

Let us define, for every $i \in \{1, \ldots, 8\}$, the function g_i as the prediction of the i^{th} 4-bit coordinate of

$\mathsf{P}^{-1}(\Delta)$ according to a pair $(C, \widetilde{C})$ and to a guess k on the value of $\mathbf{k}_{16,i}$:

$$g_i(C, \widetilde{C}, k) = \mathsf{S}_i\big(\mathsf{E}_i(L_{16}) \oplus k\big) \oplus \mathsf{S}_i\big(\mathsf{E}_i(\widetilde{L}_{16}) \oplus k\big) \oplus \mathsf{P}_i^{-1}\big(R_{16} \oplus \widetilde{R}_{16}\big) \ .$$

From (16.1) and (16.2), it can be checked that, for the correct key guess, $g_i(C, \widetilde{C}, k)$ equals $\mathsf{P}_i^{-1}(\Delta)$. On the other hand, for a wrong key guess, $g_i(C, \widetilde{C}, k)$ can be assumed to have a uniform distribution. This is a classical assumption in block cipher cryptanalysis known as the *wrong-key assumption*.

Let us define, for every $i \in \{1, \ldots, 8\}$ and for every $\delta \in \{0, \ldots, 15\}$, the probability $p_i(\delta)$ as:

$$p_i(\delta) = \mathrm{P}\left[\mathsf{P}_i^{-1}(\Delta) = \delta\right] \ .$$

To summarize, according to the wrong-key assumption, we have:

$$\mathrm{P}\left[g_i(C, \widetilde{C}, k) = \delta\right] = \begin{cases} p_i(\delta) & \text{if } k = \mathbf{k}_{16,i} \\ \frac{1}{16} & \text{otherwise} \end{cases} \tag{16.3}$$

Provided that the distribution $p_i(\cdot)$ is significantly biased, (16.3) clearly exhibits a wrong-key distinguisher for $\mathbf{k}_{16,i}$.

16.3.2 Wrong-key distinguishers

We define hereafter two possible distinguishers $d(k)$ for a key candidate k which are expected to be maximal for the correct key candidate $k = \mathbf{k}_{16,i}$. These distinguishers take as input a set of N pairs $(C_n, \widetilde{C}_n)$, $1 \leqslant n \leqslant N$. The choice of the distinguisher to use depends on the attacker's knowledge of the fault model.

Likelihood distinguisher. The attacker is assumed to have an exact knowledge of the fault model, namely he knows the distribution of ε. In that case, he can compute (or at least estimate) the distribution $p_i(\cdot)$ in order to use a maximum likelihood approach[20]. The likelihood of a key candidate k is defined as the product of the probabilities $p_i\big(g_i(C_n, \widetilde{C}_n, k)\big)$ for $n = 1, \ldots, N$. For practical reasons, we make the classical choice to use the logarithm of the likelihood, namely $d(k)$ is defined as:

$$d(k) = \sum_{n=1}^{N} \log\left(p_i\big(g_i(C_n, \widetilde{C}_n, k)\big)\right) \ .$$

Squared Euclidean imbalance (SEI) distinguisher. The attacker does not have a precise knowledge of the fault model and is hence not able to estimate the distribution $p_i(\cdot)$. In that case, an alternative strategy is to look for the strongest bias in the distribution of $g_i(C_n, \widetilde{C}_n, k)$. This is done by computing the squared Euclidean distance to the uniform distribution (known as *squared Euclidean imbalance*), namely $d(k)$ is defined as:

$$d(k) = \sum_{\delta=0}^{15} \left(\frac{\#\{n; g_i(C_n, \widetilde{C}_n, k) = \delta\}}{N} - \frac{1}{16}\right)^2 \ .$$

16.3.3 Chosen error position strategies

In a chosen error position fault model scenario, we further have to define a strategy to choose the positions where to induce the errors.

Bit error model. In the bit error model, ε has a single bit to 1 which implies that the function f in round $r + 2$ has one or two *active S-boxes*. That is, the correct output and the corrupted output of f only differ for one or two S-boxes. Indeed, as shown in Table 16.1, the expansion layer sends every input bit of f in one or two S-boxes. In order to maximize the bias in the distribution of Δ, the bit-positions

[20] Note that the distribution $p_i(\cdot)$ is independent of the secret key and only depends on the error (considering random plaintexts).

Table 16.1: Destination S-boxes for the input bits of f.

bits	1,32	2,3	4,5	6,7	8,9	10,11	12,13	14,15
S-boxes	1,8	1	1,2	2	2,3	3	3,4	4
bits	16,17	18,19	20,21	22,23	24,25	26,27	28,29	30,31
S-boxes	4,5	5	5,6	6	6,7	7	7,8	8

Table 16.2: Destination S-boxes for the input bytes of f.

bytes	1	2	3	4
S-boxes	8,1,2,3	2,3,4,5	4,5,6,7	6,7,8,1

should be chosen among the ones entering in a single S-box hence slowing the error propagation. Our strategy is simply to first choose a bit-position entering in S-box 1 only, then in S-box 2 only, and so on until S-box 8 and start over with S-box 1, etc.

Byte error model. Concerning the byte error model, every input byte of f is spread over four S-boxes. This can be checked from Table 16.2 which gives the destination S-boxes of every input byte of f. As a result, a byte error in L_r always implies four active S-boxes in the output differential of f in round $r+2$. For the attacks in the chosen position byte error model, the four byte-positions are hence equivalently chosen since they all induce the corruption of exactly four S-boxes in round $r+2$.

Remark 16.2. In a chosen error position attack, several fault models are involved hence, for a given i, different distributions $p_i(\cdot)$ are induced. Consequently, the SEI distinguisher shall not be directly applied but the SEI of $p_i(\cdot)$ shall be estimated for every error position independently. The SEI distinguisher is then defined as the sum of the SEIs for the different error positions.

Remark 16.3. In our attack simulations, we tried more specific strategies taking into account the bias in the $(p_i(\cdot))_i$ distributions resulting from the different bit-error positions. These strategies did not yield substantial improvements.

16.4 Attack simulations

This section presents some experimental results. We performed attack simulations for each of the fault models introduced in Section 16.2 with a fault induced at the end of round 12, 11, 10 or 9. For every round number and every fault model, we applied the likelihood distinguisher and the SEI distinguisher (see Section 16.3.2). For the likelihood distinguisher, we empirically computed the distributions $(p_i(\cdot))_i$ based on several[21] ciphertexts pairs, each obtained from the correct and faulty encryptions of a random plaintext[22].

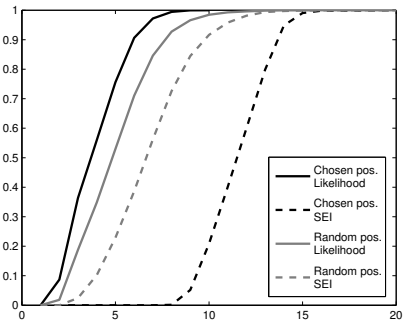
In what follows, we consider an attack successful when the entire last round key is determined with a 99% success rate. This strong requirement is motivated by the fact that, for a triple DES, too many key bits remain to perform an exhaustive search once the last round key has been recovered. Therefore, one shall fully determine the sixteenth round key before reiterating the attack on the fifteenth and so on. Every subsequent attack on a previous round key can be performed by using the same set of ciphertexts pairs and is expected to be substantially more efficient since the error propagates on fewer rounds. This way, if the last round key is recovered with a 99% success rate then the cipher can be considered fully broken.

Figure 16.2 shows the success rate (over 1000 simulations) of the different attacks (chosen/random position bit/byte error, likelihood/SEI distinguishers) on rounds 12, 11 and 10. Figure 16.3 shows the

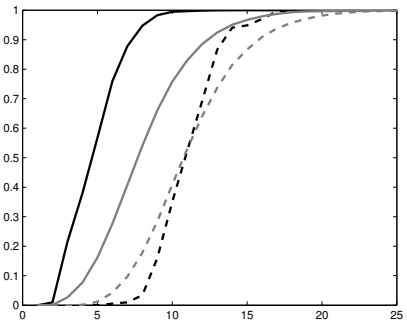
[21] 10^7 for bit errors models and 10^8 for byte errors models.

[22] Note that the value of the key does not change the $(p_i(\cdot))_i$ distributions.

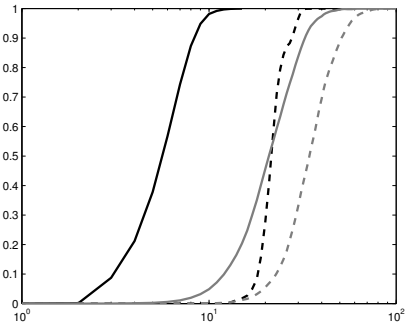
success rate (over 10 to 100 simulations) for the attacks on round 9 in the bit error model. Attacks on round 9 in the byte error model all required more than 10^8 faults. The numbers of faults required for a 99% success rate are summarized in Table 16.3.



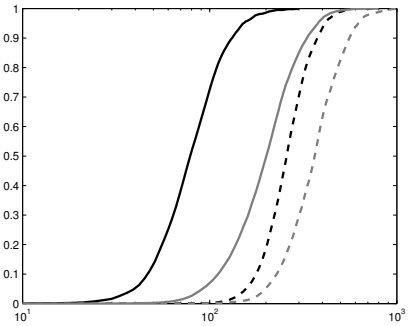
Attack on round 12 – Bit error model.



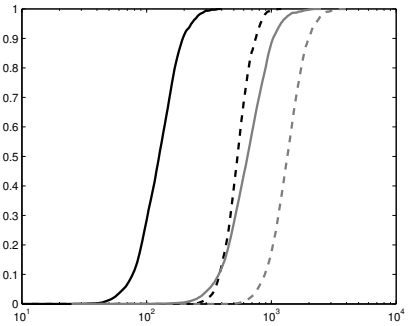
Attack on round 12 – Byte error model.



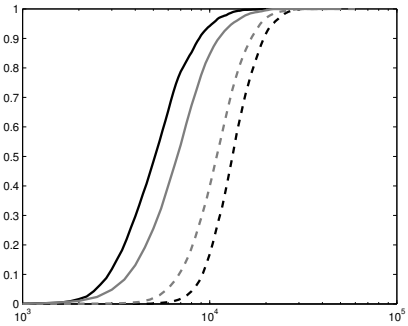
Attack on round 11 – Bit error model.



Attack on round 11 – Byte error model.



Attack on round 10 – Bit error model.



Attack on round 10 – Byte error model.

Figure 16.2: Attacks on rounds 10, 11 and 12: success rate w.r.t. number of faults.

Attack efficiency *vs.* **round number.** The attacks on rounds 11 and 12 are very efficient: less than 25 faults are sufficient on round 12 while, on round 11, less than 100 faults are sufficient in a bit error model and less than 1000 faults are sufficient in a byte error model. On round 10, the attacks are still fairly efficient: the best attack (chosen position bit error model, likelihood distinguisher) requires 290

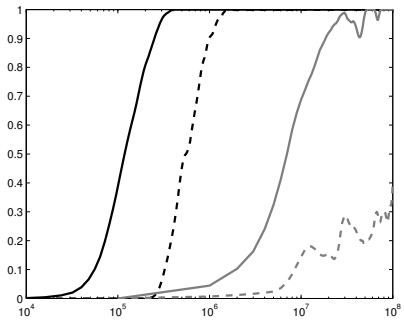


Figure 16.3: Attacks on rounds 9, bit error model: success rate w.r.t. number of faults.

Table 16.3: Number of faults to recover the 16^{th} round key with a 99% success rate.

		bit error		byte error	
round	distinguisher	chosen pos.	random pos.	chosen pos.	random pos.
12	Likelihood	7	11	9	17
	SEI	14	12	17	21
11	Likelihood	11	44	210	460
	SEI	30	71	500	820
10	Likelihood	290	1500	13400	18500
	SEI	940	2700	26400	23400
9	Likelihood	$3.4 \cdot 10^5$	$2.2 \cdot 10^7$	$> 10^8$	$> 10^8$
	SEI	$1.4 \cdot 10^6$	$> 10^8$	$> 10^8$	$> 10^8$

faults whereas the least efficient attack (chosen position byte error model, SEI distinguisher) requires 26400 faults. It is on round 9 that the attacks become quite costly since the most efficient attack in the bit error model (chosen position, likelihood distinguisher) requires approximately $3.4 \cdot 10^5$ faults and all the attacks in the byte error model require more than 10^8 faults[23].

Attack efficiency *vs.* **fault model.** As expected, we observe that, for a given setting (random/chosen position, likelihood/SEI distinguisher), a bit error model always leads to more efficient attacks than a byte error model. Similarly, a chosen position usually leads to more efficient attacks than a random position. Some exceptions are observed for the SEI distinguisher for which a random position sometimes leads to more efficient attacks than a chosen position. The reason of this phenomenon may be that in a chosen position bit (resp. byte) error model, 8 (resp. 4) different SEIs are estimated based on 8 (resp. 4) times less faults than in the random position model where a single SEI is estimated (see Remark 16.2). As a result, these estimations are less precise which may render the attack less efficient than in the random position model. In these cases, the attacker can compute a single SEI based on all the faults, which amounts to perform the attack in the random position model.

To summarize, we naturally have that a bit error is better than a byte error and a chosen position is better than a random position. What was not *a priori* straightforward is the superiority of the random position bit error model compared to the chosen position byte error model. Except on round 12 where

[23]The most efficient one (chosen position byte error model, likelihood distinguisher) yielded a 0% success rate (over 10 attack simulations) for 10^8 faults.

both cases are almost equivalent, our results show that the attacks in the random position bit error model are significantly more efficient than the ones in the chosen position byte error model.

Another interesting observation is that, in the bit error model, the ability to choose the error position is more advantageous than in the byte error model. This phenomenon results from the strategy for the choice of the bit-positions (see Section 16.3.3) which selects 8 positions over 32 leading to more important bias in the distributions $p_i(\cdot)$ than the average case, whereas, in the chosen position byte error model, the 4 byte-positions are equivalently used.

Likelihood *vs.* **SEI.** As expected, the likelihood distinguisher always leads to more efficient attacks than the SEI distinguisher. It is interesting to note that this difference of efficiency is always greater in a chosen position model than in a random position model. Once again, this phenomenon results from the fact that, for a chosen position model, several different SEIs are estimated based on 4 or 8 times less faults compared to the random position model where a single SEI is estimated.

16.5 How many rounds to protect?

The question of the number of rounds to protect does not have a unique answer. Indeed, the answer to this question depends on the ability of an attacker to induce faults and on the number of correct and faulty ciphertexts pairs that he can collect. Moreover, more efficient attacks that those described above may exist.

What provides our study is some lower bounds on the number of rounds to protect. We have shown that in a realistic fault model, efficient DFA attacks can be performed by inducing some faults until round 10. It seems therefore reasonable to protect at least the last seven rounds of the cipher. However, this may not suffice while considering a strong adversary model. We have shown that in a chosen position bit error model, $3.4 \cdot 10^5$ faults induced at the end of round 9 are sufficient to recover the last round key with a 99% confidence. Consequently, in order to thwart an adversary able to induce a single bit fault at a chosen position and to gather about 10^5 ciphertexts pairs, one shall at least protect the last eight rounds.

Attacks on initial rounds. As noted in [110], if an attacker has access to a decryption oracle then any DFA attack can be transposed on the initial rounds of the cipher. In fact, the attacker may obtain a faulty ciphertext $\widetilde{C}$ from a plaintext P by inducing a fault at the end of the first round. The plaintext P can then be viewed as the faulty result of a decryption of $\widetilde{C}$ for which a fault has been induced at the beginning of the last round. The attacker then asks for the decryption of $\widetilde{C}$ which provides him with a plaintext $\widetilde{P}$. The pair $(\widetilde{P}, P)$ thus constitutes a pair of correct and faulty results of the decryption algorithm with respect to an error induced at the beginning of the last round. According to this principle, any fault attack on an initial round of an encryption can be transposed to a fault attack on a final round of a decryption, provided that the attacker has access to a decryption oracle. In that case, the same number of rounds should be protected at the beginning and at the end of the cipher in order to obtain an homogenous security level. For a simple DES, based on our study, we recommend to protect the entire cipher. For a triple DES, one can only protect some rounds at the beginning of the first DES computation and some rounds at the end of the last DES computation; the number of protected rounds being at least seven according to our study.

Chapter 17

Securing RSA by double addition chain exponentiation

Contents

17.1 Introduction

This chapter describes a countermeasure against fault analysis for exponentiation and RSA. It consists of a *self-secure* exponentiation algorithm, namely an exponentiation algorithm that provides a direct way to check the result coherence. An RSA implemented with our solution hence avoids the use of an extended modulus (which slows down the computation) as in several other countermeasures. Moreover, our exponentiation algorithm involves 1.65 multiplications per bit of the exponent which is significantly less than the 2 required by other self-secure exponentiations.

The results presented in this chapter have been published in the *Cryptographer's Track at the RSA conference (CT-RSA 2009)* [9]. The proposed countermeasure has also been patented in [13].

17.2 A self-secure exponentiation based on double addition chains

17.2.1 Basic principle

In the following, we shall call *double exponentiation* an algorithm taking as inputs an element m and a pair of exponents (a, b), and computing the pair of powers (m^a, m^b).

The core idea of our method is to process a double exponentiation to compute the pair $(m^d, m^{\varphi(N)-d})$ modulo N. Then, the consistency of the computation is verified by performing the following check:

$$m^d \cdot m^{\varphi(N)-d} \stackrel{?}{\equiv} 1 \bmod N \ . \tag{17.1}$$

If no error occurs during the computation then, due to Euler's Theorem, this check is positive. In that case, the algorithm returns $m^d \bmod N$. On the other hand, if the computation is corrupted, then the result of this check is negative with high probability. In that case, the algorithm returns an error message.

In order to construct a self-secure exponentiation based on aforementioned principle, we need a double exponentiation algorithm. We propose hereafter such an algorithm that is well suited for implementation constrained in memory. Our solution is based on the building of an *addition chain*. This notion, as well as the ensued notion of *addition chain exponentiation* are briefly introduced in the next section (see [144] for more details).

17.2.2 Addition chain exponentiations

At first, we give the definition of an addition chain.

Definition 17.1. *An* addition chain *for an integer a is a sequence $x_0, x_1, \ldots, x_n$ with $x_0 = 1$ and $x_n = a$ that satisfies the following property: for every k there exist indices $i, j < k$ such that $x_k = x_i + x_j$.*

An addition chain $(x_i)_i$ for an integer a provides a way to evaluate any element m to the power a. Let $m_0 = m$. For k from 1 to n, one computes $m_k = m_i \cdot m_j$ where $i, j < k$ are such that $x_k = x_i + x_j$. By induction, the sequence $(m_k)_k$ satisfies: $m_k = m^{x_k}$ for every $k \leqslant n$ which leads to $m_n = m^{x_n} = m^a$. Such an addition chain exponentiation may require an important amount of memory to store the intermediate powers required for the computation of subsequent powers. This can make the exponentiation unpractical, especially in the context of low resource devices. Therefore, the minimum number of variables required to store the intermediate powers is an important parameter of the addition chain exponentiation. This parameter that directly results from the addition chain will be called the *memory depth* of the chain in the following.

An addition chain $x_0, x_1, \ldots, x_n$ with $(x_{n-1}, x_n) = (a, b)$ will be called a *double addition chain* for the pair (a, b). A double addition chain for a pair (a, b) provides a way to perform the double exponentiation $m \mapsto (m^a, m^b)$ for any element m.

Remark 17.1. What we call here double exponentiation shall not be confused with multi-exponentiations (also known as simultaneous exponentiations) that compute a product of powers $\prod_i m_i^{a_i}$ (see for instance [168]). What we call double addition chain is also called *addition sequence* in the general case where possibly more than two powers must be computed [53, 116]. Addition sequences have not been so much investigated. In [53], the authors propose some heuristics but these are not suitable for implementations constrained in memory.

17.2.3 A heuristic for double addition chains

In this section, we propose a heuristic to compute a double addition chain with a memory depth of 3 for any pair of natural integers (a, b). This provides us with a double exponentiation algorithm that is well suited for implementations constrained in memory.

Without loss of generality, we assume $a \leqslant b$. The chain involves a pair of intermediate results (a_i, b_i) that is initialized to $(0, 1)$ and that equals (a, b) once all the additions have been performed. In order to have a memory depth of 3, one single additional variable is used that holds the value 1 (this amounts to

hold the element m in a register for the resulting exponentiation). Therefore, at the i^{th} step of the chain, one can either increment a_i or b_i by 1, double a_i or b_i, or add a_i and b_i together.

To construct such a chain, we start from the pair (a, b) and go down to the pair $(0, 1)$ by applying the inverse operations. Namely, we define a sequence $(\alpha_i, \beta_i)_i$ such that $(\alpha_0, \beta_0) = (a, b)$ and $(\alpha_n, \beta_n) = (0, 1)$ for some $n \in \mathbb{N}$, and where, for every i, the pair $(\alpha_{i+1}, \beta_{i+1})$ is obtained from (α_i, β_i) by decrementing, by dividing by two and/or by subtracting an element to the other one. In order to limit the memory required to the storage of the chain, we have to restrict the set of possible operations. Our heuristic is the following one:

$$(\alpha_{i+1}, \beta_{i+1}) = \begin{cases} (\alpha_i, \beta_i/2) & \text{if } \alpha_i \leqslant \beta_i/2 \text{ and } \beta_i \bmod 2 = 0 \\ (\alpha_i, (\beta_i - 1)/2) & \text{if } \alpha_i \leqslant \beta_i/2 \text{ and } \beta_i \bmod 2 = 1 \\ (\beta_i - \alpha_i, \alpha_i) & \text{if } \alpha_i > \beta_i/2 \end{cases} \tag{17.2}$$

Proposition 17.1. *If $\alpha_0, \beta_0 \in \mathbb{N}^*$ are such that $\alpha_0 \leqslant \beta_0$ then the sequence $(\alpha_i, \beta_i)_i$ satisfies the following properties:*

1. *For every i, we have $\alpha_i \leqslant \beta_i$.*
2. *There exists $n \in \mathbb{N}$ such that $(\alpha_n, \beta_n) = (0, 1)$.*

Proof. The first property is straightforward: it is true for $i = 0$ and it is preserved by every step. The second one is demonstrated as follows. For every i such that $\alpha_i > 0$, we have $\alpha_{i+1} \leqslant \beta_{i+1} \leqslant \beta_i$ and $\alpha_{i+1} + \beta_{i+1} < \alpha_i + \beta_i$. This implies that there exists $n' \in \mathbb{N}$ such that $\alpha_{n'} > 0$ and $\alpha_{n'+1} \leqslant 0$. From (17.2), one deduces $\alpha_{n'} = \beta_{n'} > 0$ and $\alpha_{n'+1} = 0$. Denoting x the natural integer such that $(\alpha_{n'+1}, \beta_{n'+1}) = (0, x)$, we finally get $(\alpha_{n'+\lceil \log x \rceil}, \beta_{n'+\lceil \log x \rceil}) = (0, 1)$. □

At this point, we need a binary representation for the sequence of additions to perform for the processing of the sequence $(a_i, b_i)_i$. Let us denote by n the natural integer satisfying $(\alpha_n, \beta_n) = (0, 1)$. We define τ and ν as the n-bit vectors whose coordinates satisfy:

$$\tau_i = \begin{cases} 0 & \text{if } \alpha_{n-i} \leqslant \beta_{n-i}/2 \\ 1 & \text{if } \alpha_{n-i} > \beta_{n-i}/2 \end{cases} \tag{17.3}$$

and

$$\nu_i = \beta_{n-i} \bmod 2 \ . \tag{17.4}$$

The sequence $(a_i, b_i)_i$ can be computed from τ and ν by initializing (a_0, b_0) to $(0, 1)$ and by iterating:

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i, 2b_i) & \text{if } \tau_{i+1} = 0 \text{ and } \nu_{i+1} = 0 \\ (a_i, 2b_i + 1) & \text{if } \tau_{i+1} = 0 \text{ and } \nu_{i+1} = 1 \\ (b_i, a_i + b_i) & \text{if } \tau_{i+1} = 1 \end{cases}$$

One can verify that $(a_i, b_i) = (\alpha_{n-i}, \beta_{n-i})$ holds for every i which yields $(a_n, b_n) = (a, b)$.

Let us remark that the entire sequence ν is not necessary for processing this addition chain (and the resulting exponentiation). Indeed, only the bits ν_i for which τ_i equals 0 are required. Therefore, the exponentiation algorithm shall make use of a single compressed sequence ω in order to avoid memory loss. We simply define ω as the sequence obtained from τ by inserting every bit ν_i for which $\tau_i = 0$ between τ_i and τ_{i+1}. In the sequel, we shall denote by n^* the bit-length of ω. Moreover, when we will need to make appear the relationship between the pair (a, b) and ω, we will use the notation $\omega(a, b)$.

The sequence $\omega(a, b)$ thus constitutes the binary representation of the double addition chain for the pair of exponents (a, b). To process the corresponding double exponentiation one must pre-compute ω. This is done by computing the pair (α_i, β_i) for every $i \in \{1, \ldots, n\}$. The following algorithm details such a computation. It makes use of two registers R_0 and R_1 that store the intermediate results α_i and β_i. It makes also use of a Boolean variable γ such that α_i is stored in $R_{\gamma \oplus 1}$ and β_i is stored in R_γ.

Algorithm 21 Double addition chain computation – ChainCompute

INPUT: a pair of natural integers (a, b) s.t. $a \leqslant b$
OUTPUT: the chain $\omega(a, b)$

1. $R_0 \leftarrow a$; $R_1 \leftarrow b$; $\gamma \leftarrow 1$; $j \leftarrow n^*$
2. **while** $(R_{\gamma\oplus1}, R_\gamma) \neq (0, 1)$ **do**
3. **if** $(R_\gamma/2 > R_{\gamma\oplus1})$
4. **then** $\omega_{j-1} \leftarrow 0$; $\omega_j \leftarrow R_\gamma \bmod 2$; $R_\gamma \leftarrow R_\gamma/2$; $j \leftarrow j - 2$
5. **else** $\omega_j \leftarrow 1$; $R_\gamma \leftarrow R_\gamma - R_{\gamma\oplus1}$; $\gamma \leftarrow \gamma \oplus 1$; $j \leftarrow j - 1$
6. **end while**
7. **return** ω

Remark 17.2. The length n^* is *a priori* unknown before the computation of the chain. However, as shown in Section 17.6.2, it is upper bounded by $2.2\lceil \log b \rceil$ (with high probability). For a practical implementation of Algorithm 21, one may use a buffer of $2.2\lceil \log b \rceil$ bits to store ω.

The following algorithm describes the resulting double modular exponentiation algorithm. It makes use of two registers R_0 and R_1 that store the intermediate results m^{a_i} and m^{b_i} and one more register to hold m. It makes also use of a Boolean variable γ such that m^{a_i} is stored in $R_{\gamma\oplus1}$ and m^{b_i} is stored in R_γ.

Algorithm 22 Double modular exponentiation – DoubleExp

INPUT: an element $m \in \mathbb{Z}_N$, a chain $\omega(a, b)$ s.t. $a \leqslant b$, a modulus N
OUTPUT: the pair of modular powers $(m^a \bmod N, m^b \bmod N)$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow m$; $\gamma \leftarrow 1$
2. **for** $i = 1$ **to** n^* **do**
3. **if** $(\omega_i = 0)$ **then**
4. $R_\gamma \leftarrow R_\gamma^2 \bmod N$; $i \leftarrow i + 1$
5. **if** $(\omega_i = 1)$ **then** $R_\gamma \leftarrow R_\gamma \cdot m \bmod N$
6. **else**
7. $R_{\gamma\oplus1} \leftarrow R_{\gamma\oplus1} \cdot R_\gamma \bmod N$; $\gamma \leftarrow \gamma \oplus 1$
8. **end for**
9. **return** $(R_{\gamma\oplus1}, R_\gamma)$

17.2.4 The secure exponentiation algorithm

Following the principle described in Section 17.2.1, Algorithm 22 provides a way to perform a modular exponentiation secure against fault analysis. The resulting secure modular exponentiation is depicted in the following algorithm.

Algorithm 23 Secure modular exponentiation

INPUT: a message m, a secret exponent d, a modulus N and its Euler's totient $\varphi(N)$
OUTPUT: the modular power $m^d \bmod N$

1. $\omega \leftarrow \mathsf{ChainCompute}\big(d,\ 2\varphi(N) - d\big)$
2. $(s, c) \leftarrow \mathsf{DoubleExp}\big(m,\ \omega,\ N\big)$
3. **if** $s \cdot c \bmod N \neq 1$ **then return** "error"; **else return** s

Remark 17.3. For the chain computation (Step 1), $\varphi(N) - d$ is replaced by $2\varphi(N) - d$ in order to fit the constraint $a \leqslant b$ imposed by the chain computation algorithm. This does not affect the result of the double exponentiation in Step 2 since we have $m^{\varphi(N)-d} \equiv m^{2\varphi(N)-d} \bmod N$.

17.3 A secure RSA-CRT algorithm

For an RSA computation, the secure modular exponentiation proposed above can be extended to be performed in CRT mode. Two double exponentiations are performed separately in order to compute the pairs (s_p, c_p) and (s_q, c_q) where:

$$c_p = m^{p-1-d_p} \bmod p$$

and

$$c_q = m^{q-1-d_q} \bmod q \ .$$

Then the signature s is recovered from s_p and s_q by CRT recombination and its value is checked modulo p (resp. q) using c_p (resp. c_q) according to (17.1).

Algorithm 24 Secure RSA-CRT

INPUT: a message m, the secret exponents d_p and d_q, the secret primes p and q
OUTPUT: the modular power $m^d \bmod p \cdot q$

1. $\omega_p \leftarrow \mathsf{ChainCompute}\big(d_p,\ 2(p-1) - d_p\big)$
2. $(s_p, c_p) \leftarrow \mathsf{DoubleExp}(m \bmod p,\ \omega_p,\ p)$
3. $\omega_q \leftarrow \mathsf{ChainCompute}\big(d_q,\ 2(q-1) - d_q\big)$
4. $(s_q, c_q) \leftarrow \mathsf{DoubleExp}(m \bmod q,\ \omega_q,\ q)$
5. $s \leftarrow \mathsf{CRT}(s_p, s_q)$
6. **if** $(s \cdot c_p \bmod p \neq 1$ **or** $s \cdot c_q \bmod q \neq 1)$ **then return** "error" **else return** s

Remark 17.4. We assume that $m \bmod p$ (resp. $m \bmod q$) cannot be corrupted before the beginning of the double exponentiation. This is mandatory for the security of Algorithm 24, since such a corruption would not be detected and would enable the Bellcore attack. In practice, this can be ensured by computing a cyclic redundancy code for $m \bmod p$ (resp. $m \bmod q$) at the beginning of the RSA-CRT algorithm. Then, at the beginning of the double exponentiation algorithm, $m \bmod p$ (resp. $m \bmod q$) is recomputed from m and its integrity is checked once it has been loaded in two different registers (m and R_1 in Algorithm 22). Any corruption occurring after this check shall be detected by the final check.

Remark 17.5. The chains ω_p and ω_q can be either computed on-the-fly as depicted in Algorithm 24 (Steps 1 and 3) or pre-computed and stored in non-volatile memory. The first solution has the advantages of preserving the classical RSA-CRT parameters and of enabling the exponent blinding countermeasure (see Section 17.5.2). The second solution has the advantage of avoiding the timing and memory overhead induced by the chain computations.

17.4 Security against fault analysis

In this section, we analyze the security of our method against fault analysis. We start with a few remarks of practical purpose, then we investigate the detection probability of a fault injection and finally we address safe-error attacks.

Remark 17.6. We assume that the Boolean γ cannot be modified at the end of Algorithm 22. This is mandatory for the security of the solution since such a modification would result in a swapping of the two registers which would not be detected. In practice, this can be ensured by doubling the variable γ.

Remark 17.7. We assume that one cannot switch the last bit(s) of the chain ω (resp. ω_p, ω_q) from 1 to 00 (or *vice versa*). This would provoke an undetected error. Such a switch can be prevented in practice by checking that the loop index i matches the chain length n^* at the end of Algorithm 22.

Remark 17.8. In Algorithms 23 and 24, we assume that the integrity of the chain computation parameters is checked before executing the chain computation algorithm. This avoids any attack that would corrupt d (resp. d_p, d_q) before the computation of $2\varphi(N) - d$ (resp. $2(p-1) - d_p$, $2(q-1) - d_q$).

Remark 17.9. Some papers claim that coherence checks using conditional branches should be avoided to strengthen fault analysis security [72, 249]. The argument behind this assertion is that the coherence check could be easily skipped by corrupting the status register. In the next chapter, a simple solution is proposed that performs a coherence check in a way that is secure against operations skipping. We suggest to use this solution for the coherence checks performed in Algorithm 23 (Step 3) and Algorithm 24 (Step 6).

17.4.1 Fault detection

We analyze hereafter the different fault attacks that can be attempted on our secure exponentiation algorithm and we investigate the corresponding detection probability. We only focus on transient faults, namely faults whose effect lasts for one computation. Permanent fault attacks are easily thwarted by the addition of some cyclic redundancy codes to check the parameters integrity.

We use the generic notation M to denote the involved modulus that may equal N (for a standard RSA), p or q (for an RSA-CRT) and we denote by $\mathrm{ord}_M(m)$ the order of an element m in $\mathbb{Z}_M^*$. When the fault causes the corruption of an intermediate variable v, we denote the corrupted variable by $\widetilde{v}$ and the error by ε such that $\widetilde{v} = v + \varepsilon$. We analyze here the condition about ε for a non-detection and we bound the probability $\mathcal{P}$ of non-detection in the *uniform fault model i.e.* assuming that ε is uniformly distributed.

For our analysis, the following lemma shall be useful.

Lemma 17.1. *Let M be an integer greater than 30. Let m be a random variable uniformly distributed over $\mathbb{Z}_M^*$ and let u be a random variable uniformly distributed over $\{1, \ldots, \varphi(M)\}$ and independent of m. We have:*

$$\mathrm{P}\left[\mathrm{ord}_M(m)|u\right] < \frac{2}{M^{1/3}} \ . \tag{17.5}$$

Proof. By the law of total probability, we have:

$$\mathrm{P}\left[\mathrm{ord}_M(m)|u\right] = \sum_{\lambda \in \mathcal{D}(\varphi(M))} \mathrm{P}\left[\lambda|u\right] \mathrm{P}\left[\mathrm{ord}_M(m) = \lambda\right] \ , \tag{17.6}$$

where $\mathcal{D}$ is the function mapping a natural integer to the set of its divisors. On the one hand, the probability $\mathrm{P}\left[\lambda|u\right]$ equals $1/\lambda$. On the other hand, for every $\lambda \in \mathcal{D}(\varphi(M))$, there are $\varphi(\lambda)$ elements of order λ in $\mathbb{Z}_M^*$ which leads to $\mathrm{P}\left[\mathrm{ord}_M(m) = \lambda\right] = \varphi(\lambda)/\varphi(M)$. On the whole, (17.6) can be rewritten as:

$$\mathrm{P}\left[\mathrm{ord}_M(m)|u\right] = \frac{1}{\varphi(M)} \sum_{\lambda \in \mathcal{D}(\varphi(M))} \frac{\varphi(\lambda)}{\lambda} \ . \tag{17.7}$$

Since $\varphi(\lambda)/\lambda$ is strictly lower than or equal to 1, we have $\mathrm{P}\left[\mathrm{ord}_M(m)|u\right] \leqslant d(\varphi(M))/\varphi(M)$ where $d(\cdot)$ denotes the divisor function (*i.e.* the function that maps a natural integer to the quantity of its distinct divisors). It is well known that the divisor function satisfies $d(x) < 2\sqrt{x}$ for every x [167] which implies $\mathrm{P}\left[\mathrm{ord}_M(m)|u\right] < 2/\sqrt{\varphi(M)}$. Since we have $\varphi(M) > n^{2/3}$ for every $M > 30$ [167], we get (17.5). □

For the sake of simplicity, we approximate hereafter a uniform distribution over $\mathbb{Z}_M$ by a uniform distribution over $\mathbb{Z}_M^*$. This approximation is sound in our context since M is a large prime or an RSA modulus.

Corruption of one of the two exponents. Among the two exponents a and b, one equals d and the other one equals $\varphi(M) - d$. On the one hand, if $\varphi(M) - d$ is corrupted, then the result of the exponentiation remains correct (*i.e.* it equals $m^d \bmod M$) and the attack failed whatever the result of the final check (which is however very likely to detect the fault). On the other hand, if d is corrupted, we show hereafter that the final check will detect the error with high probability.

In fact, the error is not detected if and only if we have $m^{\widetilde{d}} \cdot m^{\varphi(M)-d} \equiv 1 \bmod M$ that is $m^{\varepsilon} \equiv 1 \bmod M$. This occurs if and only if ε is a multiple of the order of m. Therefore, the probability of non-detection can be expressed as $\mathcal{P} = \mathrm{P}\left[\mathrm{ord}_M(m)|\varepsilon\right]$ (the lower the order of m, the higher the probability of non-detection).

Since a potential attacker does not know $\varphi(M)$, he cannot chose m in a way that affects its order. For this reason, m can be considered uniformly distributed over $\mathbb{Z}_M$. Therefore, in the uniform fault model, Lemma 17.1 implies $\mathcal{P} < 2/M^{1/3}$.

Remark 17.10. The bound provided by Lemma 17.1 is not tight at all but it is sufficient to show that $\mathcal{P}$ is negligible. For instance, if M satisfies $\log M \geqslant 244$, which is necessary (but not sufficient) for the security of RSA (even for RSA-CRT where $\log N = 2 \log M$), $\mathcal{P}$ is strictly lower than 2^{-80} which is negligible.

Corruption of the message or an intermediate power. From the definition of the double addition chain given in Section 17.2.3, one can see that for every $i \in \{1, \ldots, n\}$, the pair (a_n, b_n) can be expressed as a linear transformation of the triplet $(a_i, b_i, 1)$. Let us denote by α_i^a, β_i^a, δ_i^a the three coefficients of the expression of a_n, namely $a_n = \alpha_i^a a_i + \beta_i^a b_i + \delta_i^a$. By analogy, we denote by α_i^b, β_i^b, δ_i^b the coefficients in the expression of b_n.

If the message m is corrupted at the i^{th} step of the exponentiation, the latter returns the following pair of powers: $\left(m^a(m^{-1} \cdot \widetilde{m})^{\delta_i^a}, m^b(m^{-1} \cdot \widetilde{m})^{\delta_i^b}\right)$ modulo M. The error is not detected if and only if we have $(m^{-1} \cdot \widetilde{m})^{\delta_i^a + \delta_i^b} \equiv 1 \bmod M$, that is $(1 + \varepsilon \cdot m^{-1})^{\delta_i^a + \delta_i^b} \equiv 1 \bmod M$. This occurs if and only if the order of $m' = 1 + \varepsilon \cdot m^{-1}$ divides $\delta_i^a + \delta_i^b$. Therefore, the probability of non-detection can be expressed as $\mathcal{P} = \mathrm{P}\left[\mathrm{ord}_M(m') | \delta_i^a + \delta_i^b\right]$. Following the same reasoning, a corruption of the intermediate power m^{a_i} (resp. m^{b_i}) is not detected with a probability $\mathcal{P} = \mathrm{P}\left[\mathrm{ord}_M(m') | \alpha_i^a + \alpha_i^b\right]$ where $m' = 1 + e \cdot m^{-a_i}$ (resp. $\mathcal{P} = \mathrm{P}\left[\mathrm{ord}_M(m') | \beta_i^a + \beta_i^b\right]$ where $m' = 1 + \varepsilon \cdot m^{-b_i}$).

Since a and b are unknown to the attacker, this one cannot chose the value of $\delta_i^a + \delta_i^b$, $\alpha_i^a + \alpha_i^b$ or $\beta_i^a + \beta_i^b$ as these directly ensue from a and b. That is why, we make the heuristic assumption that $\mathcal{P}$ equals $\mathrm{P}\left[\mathrm{ord}_M(m') | u\right]$ where u is uniformly distributed over $\{1, \ldots, \varphi(M)\}$. In the uniform fault model, we have the uniformity of m' that holds from the one-to-one relationship between ε and m' for every $m \neq 0$. Consequently, Lemma 17.1 implies $\mathcal{P} < 2/M^{1/3}$ and $\mathcal{P}$ is negligible.

Corruption of the chain. A faulty chain $\widetilde{w}$ results in a faulty pair of powers $(m^{\widetilde{a}}, m^{\widetilde{b}})$. The error is not detected if and only if the order of m divides $\widetilde{a} + \widetilde{b}$, hence the non-detection probability can be expressed as $\mathcal{P} = \mathrm{P}\left[\mathrm{ord}_M(m) | \widetilde{a} + \widetilde{b}\right]$.

As shown in Section 17.6.2, the expected bit-length of the chain ω yielding a pair of l-bit exponents (a, b) is $2l$. This suggests an almost bijective relationship between the chains space and the exponents pairs space. In the uniform fault model, we can therefore consider that $\widetilde{a}$ and $\widetilde{b}$ are uniformly distributed which, by Lemma 17.1, implies $\mathcal{P} < 2/M^{1/3}$.

Corruption of the modulus. If the modulus M is corrupted at the i^{th} step of the exponentiation, then the latter results in the two following powers: $m_1^{\alpha_i^a} \cdot m_2^{\beta_i^a} \cdot m^{\delta_i^a} \bmod \widetilde{M}$ and $m_1^{\alpha_i^b} \cdot m_2^{\beta_i^b} \cdot m^{\delta_i^b} \bmod \widetilde{M}$ where $m_1 = m^{a_i} \bmod M$ and $m_2 = m^{b_i} \bmod M$. Therefore, the error is not detected if and only if we have $m_1^{\alpha_i^a + \alpha_i^b} \cdot m_2^{\beta_i^a + \beta_i^b} \cdot m^{\delta_i^a + \delta_i^b} \bmod \widetilde{M} = 1$.

In the uniform fault model, the faulty modulus $\widetilde{M}$ is uniformly distributed over $[0, 2^l[$ where l denotes the bit-length of M. Therefore, the probability of non-detection $\mathcal{P}$ is close to $\mathrm{P}\left[u_1 \bmod u_2 = 1\right]$ where u_1 and u_2 are uniform (and independent) random variables over $[0, 2^l[$. This probability equals $2^{-l} \sum_{i=1}^{2^l - 1} (1/i)$ which is strictly lower than 2^{-80} for every $l \geqslant 86$. The probability of non-detection $\mathcal{P}$ is hence negligible in our context.

17.4.2 Safe-error attacks

As explained in Section 15.7.3, safe-error attacks divide into two categories: C-safe-error attacks [248] and M-safe-error attacks [143,246]. To prevent C-safe-error attacks one must ensure that no dummy operation is conditionally performed depending on the secret key. Our secure exponentiation does not perform any dummy operation and is hence secure against C-safe-error attacks. When the chain is computed on-the-fly, it must be done in an atomic way in order to thwart SPA (see Section 17.5.1). The atomic version of the chain computation algorithm makes use of dummy operations and is hence vulnerable to C-safe-error

attacks. In that case, one must use the exponent blinding countermeasure to thwart them (see Section 17.5.2). In addition, the exponent blinding countermeasure also thwarts M-safe-error attacks. On the other hand, if the chain is pre-computed, the exponent cannot be randomized and M-safe-error must be prevented in another way (while C-safe-error attacks do not apply since no dummy operations are involved anymore). In that case, we suggest to randomize the indices of the registers that are addressed by some exponent bits (or chain bits in our context). Namely, the registers used to store the different variables are randomly chosen at each execution among the different available registers. For instance, in Algorithm 21, a random bit r is picked up so that the registers R_0 and R_1 are switched if r equals 1. In the description of Algorithm 21 this amounts to replace R_γ by $R_{\gamma \oplus r}$. In this way, an M-safe-error attack will imply a faulty output once out of two, independently of the performed operation.

17.5 Toward side channel analysis resistance

In this section, we address the resistance of our exponentiation algorithm against simple power analysis (SPA) and differential power analysis (DPA).

17.5.1 Simple power analysis

As explained in Section 3.1, SPA [148] exploits the fact that the operation flow of a cryptographic algorithm may depend on the secret key. Different operations may induce different patterns in the side channel leakage which provides secret information to any attacker able to eavesdrop this leakage. To thwart SPA, an algorithm must be atomic [71], namely, it must have the same operation flow whatever the secret key.

The chain computation algorithm (Algorithm 22) and the double exponentiation algorithm (Algorithm 21) may be vulnerable to SPA. To circumvent this weakness, we provide hereafter atomic versions of these algorithms.

Atomic chain computation. Looking at the chain computation algorithm, we observe that the main operations (namely operations on large registers) performed at each loop iteration are a division by two and possibly a substraction (depending on the value of τ_i). To render the algorithm atomic both operations must be performed at each loop iteration. The following algorithm describes the atomic version of the chain computation. It makes use of three registers: R_0, R_1 and R_2 which are used to store the values of α_i and β_i as well as a temporary value. It also uses three indices $i_\alpha, i_\beta, i_{tmp} \in \{0,1,2\}$ such that α_i is stored in R_{i_α}, β_i is stored in R_{i_β} and the temporary value is stored in $R_{i_{tmp}}$.

Algorithm 25 Atomic double addition chain computation
INPUT: a pair of natural integer (a, b) s.t. $a \leqslant b$
OUTPUT: the chain $\omega(a, b)$

1. $R_{i_\alpha} \leftarrow a$; $R_{i_\beta} \leftarrow b$; $j \leftarrow n^*$
2. **while** $(R_{i_\alpha}, R_{i_\beta}) \neq (0, 1)$ **do**
3. $\quad R_{i_{tmp}} \leftarrow R_{i_\beta} - R_{i_\alpha}$
4. $\quad v \leftarrow R_{i_\beta} \bmod 2$
5. $\quad R_{i_\beta} \leftarrow R_{i_\beta}/2$
6. $\quad t \leftarrow (R_{i_\beta} \leqslant R_{i_\alpha})$
7. $\quad \omega_{j-1} \leftarrow t$; $\omega_j \leftarrow t \vee v$
8. $\quad (i_\alpha, i_\beta, i_{tmp}) \leftarrow \big(t \wedge (i_{tmp}, i_\alpha, i_\beta)\big) \vee \big((t \oplus 1) \wedge (i_\alpha, i_\beta, i_{tmp})\big)$
9. $\quad j \leftarrow j - 1 - (t \oplus 1)$
10. **end while**
11. **return** ω

Notations. In Step 6, the notation $t \leftarrow (R_{i_\beta} \leqslant R_{i_\alpha})$ is used to denote the operation that compares the two values in R_{i_β} and R_{i_α} and that returns the binary value t satisfying $t = 1$ if $R_{i_\beta} \leqslant R_{i_\alpha}$ and $t = 0$

otherwise. In Step 8, the logical AND is extended to the $\{0,1\} \times \{0,1\}^n \to \{0,1\}^n$ operator performing a logical AND between the left argument and each coordinate of the right argument.

Looking at Algorithm 25, we see that, at each loop iteration, the Boolean values t and v represent the values of τ_i and ν_i. One can verify that if $t = 0$ then these values are stored in (ω_{j-1}, ω_j) and j is decremented by two while if $t = 1$ then t is stored in ω_j and j is decremented by one. Moreover, if $t = 0$ then Step 8 has no effect while if $t = 1$ then it ensures that the indices of the different registers are permuted so that (α_i, β_i) is correctly updated.

Although Algorithm 25 requires three l-bit registers and a $(2.2\ l)$-bit buffer to store ω (see Section 17.6), its memory consumption can be reduced to $4.2\ l$ bits using the following trick. During the computation of the $1.2\ l$ high order bits of ω, the l low order bits allocated for ω are used as one of the three necessary l-bit registers. Once the $1.2\ l$ high order bits of ω have been computed, the intermediate values α_i and β_i have a bit-length lower than $l/2$. Therefore, the three registers can be allocated on less than $2l$ bits and the low order part of the buffer for ω can be freed.

Atomic double exponentiation. The following algorithm describes the atomic version of the double modular exponentiation. It makes use of two registers $R_{(0,0)}$ and $R_{(0,1)}$ that are used to store the intermediate results m^{a_i} and m^{b_i} and one more register $R_{(1,0)}$ to store m. It makes also use of two Boolean variables γ and μ. The Boolean γ indicates that m^{a_i} is stored in $R_{(0,\gamma\oplus 1)}$ and that m^{b_i} is stored in $R_{(0,\gamma)}$. And the Boolean μ indicates wether the next modular multiplication is a multiplication by m ($\mu = 0$) or not ($\mu = 1$).

Algorithm 26 Atomic double modular exponentiation

INPUT: an element $m \in \mathbb{Z}_N$, a chain $\omega(a, b)$ s.t. $a \leqslant b$, a modulus N
OUTPUT: the pair of modular power $(m^a \bmod N, m^b \bmod N)$

1. $R_{(0,0)} \leftarrow 1$; $R_{(0,1)} \leftarrow m$; $R_{(1,0)} \leftarrow m$
2. $\gamma \leftarrow 1$; $\mu \leftarrow 1$; $i \leftarrow 0$
3. **while** $i < n$ **do**
4. $\quad t \leftarrow \omega_i \wedge \mu$; $v \leftarrow \omega_{i+1} \wedge \mu$
5. $\quad R_{(0,\gamma\oplus t)} \leftarrow R_{(0,\gamma\oplus t)} \cdot R_{((\mu\oplus 1),\gamma\wedge\mu)} \bmod N$
6. $\quad \mu \leftarrow t \vee (v \oplus 1)$; $\gamma \leftarrow \gamma \oplus t$
7. $\quad i \leftarrow i + \mu + \mu \wedge (t \oplus 1)$
8. **end while**
9. **return** $(R_{\gamma\oplus 1}, R_\gamma)$

While $\mu = 1$, the Boolean t is evaluated to τ_i and, if $\tau_i = 1$, the Boolean v is evaluated to ν_i. Then, while $t = 1$ or $v = 0$ each loop iteration corresponds to a step performing one single multiplication which is done in Step 5. If $t = 0$ and $\nu = 1$, the step must perform two multiplications: $R_{(0,\gamma)}$ by $R_{(0,\gamma)}$ and $R_{(0,\gamma)}$ by $R_{(1,0)}$. The first one is performed in Step 5 afterwards the Boolean μ is evaluated to 0 thus indicating that the next loop must perform the multiplication by $R_{(1,0)}$. In that case, i is not incremented and the next loop iteration performs the desired multiplication before evaluating μ to 1 and normally carrying on the computation.

17.5.2 Differential power analysis

As explained in Section 3.1 (and developed in Chapter 3), DPA [148] exploits the fact that the side channel leakage reveals information about some key-dependent intermediate variables of the computation. Since its first publication, several improvements of DPA have been proposed, in particular to attack modular exponentiation [25, 95, 127, 166]. In order to thwart DPA, one usually makes use of randomization techniques. The message randomization as well as the modulus randomization are common countermeasures (see Section 3.6.3.2) that can be straightforwardly combined with our method. The exponent is usually randomized using the blinding technique that consists in performing the exponentiation to the power

$d' = d + r \cdot \varphi(N)$ for a small random number r [76, 146, 166]. This technique cannot be straightforwardly applied while using our secure exponentiation algorithm since we have $d' > \varphi(N)$ for every $r > 0$. Therefore, we propose the following simple adaptation: in Step 1 of Algorithm 23, the exponent a is set to $d + r_1 \cdot \varphi(N)$ and the exponent b is set to $r_2 \cdot \varphi(N) - d$ where r_1 and r_2 are two small random numbers with $r_2 \geqslant r_1 + 2$. Then the rest of the secure exponentiation algorithm does not change. Since $m^{d+r_1 \cdot \varphi(N)} \equiv m^d \bmod N$, the desired signature is computed and since $m^{d+r_1 \cdot \varphi(N)} \cdot m^{r_2 \cdot \varphi(N)-d} \equiv 1 \bmod N$, the final check is correctly carried out.

Remark 17.11. If the chain ω is pre-computed, the exponent blinding cannot be used. In that case, another kind of randomization (message, modulus) shall be used. However, these do not prevent a DPA targeting the chain itself (as for instance the SEMD attack of [166] or the address-bit DPA [127]). To deal with this issue, we suggest to use a Boolean masking such as proposed in [128].

17.6 Complexity analysis

In this section we analyze the time complexity and the memory complexity[24] of our proposal. In the sequel, we shall denote by l the bit-length of the exponentiation inputs. Namely for a standard RSA we have $l = \lceil \log N \rceil$ and for an RSA-CRT we have $l = \lceil \log N/2 \rceil$.

17.6.1 Time complexity

Our secure exponentiation is mainly composed of the chain computation and the double exponentiation. The chain computation loop is shorter than the exponentiation loop and it involves simple operations (*e.g.* substraction, division by 2) whose time complexities are negligible compared to a modular multiplication. Therefore, the time complexity of our proposal mainly depends on the number of multiplications performed by the double exponentiation algorithm (all the more so as the chain may be pre-computed). We shall denote this number by m and we shall define the *multiplications-per-bit ratio* as the coefficient θ satisfying $m = \theta l$.

Some practical values for the expectation and the standard deviation of θ are given in Table 17.6.1 that were obtained by simulations. For $l \in \{512, \ldots, 1024\}$, the expected multiplications-per-bit ratio is approximately 1.65. Compared to the classical *square-and-multiply* algorithm, our exponentiation hence requires 10% more multiplications, implying a 10% overhead on average, which is a fair cost for fault analysis resistance. Moreover, it can be checked that the time complexity of our exponentiation is steadier than the one of the square-and-multiply for which the standard deviation $\sigma[\theta]$ equals $1/(2\sqrt{l})$.

Table 17.1: Expectation and standard deviation of the double exponentiation multiplications-per-bit ratio.

	$l = 512$	$l = 640$	$l = 768$	$l = 896$	$l = 1024$
$\mathrm{E}[\theta]$	1.65	1.66	1.66	1.66	1.66
$\sigma[\theta]$	0.020	0.017	0.017	0.016	0.014

17.6.2 Memory complexity

Our double exponentiation algorithm requires three l-bit registers to store the message and the pair of powers. If the chain ω is computed on-the-fly then an additional buffer is necessary to store it.

We performed simulations to derive the practical values of the expectation and the standard deviation of the chain length n^*. For the expectation, we obtained $\mathrm{E}[n^*] \approx 2.03\ l$ for $l \in \{512, \ldots, 1024\}$. For the standard deviation, the obtained values are summarized in Table 17.2. Approximating the distribution

[24] We only focus on RAM consumption without taking into account the modulus and the exponent (which are usually stored in non-volatile memory).

of n^* by a Gaussian, we get $\mathrm{P}\left[n^* > \mathrm{E}\left[n^*\right] + k\sigma\left[n^*\right]\right] = \left(1 - \mathrm{erf}\left(k/\sqrt{2}\right)\right)/2$ where $\mathrm{erf}(\cdot)$ denotes the error function. For $k = 10$ and for $l \in \{512, \ldots, 1024\}$, this probability is lower than 2^{-80}. Consequently, for $l \in \{512, \ldots, 1024\}$, the probability to have $n^* > 2.2\ l$ is negligible in practice, hence ω can be stored in a $(2.2\ l)$-bit buffer.

Table 17.2: Standard deviation of the chain bit-length.

	$l = 512$	$l = 640$	$l = 768$	$l = 896$	$l = 1024$
$\sigma\left[n^*\right]$	$0.015\ l$	$0.013\ l$	$0.011\ l$	$0.010\ l$	$0.010\ l$

On the whole, our secure exponentiation requires $5.2\ l$ bits of memory when the chain is computed on-the-fly and it requires $3\ l$ bits of memory when the chain is pre-computed.

For our secure RSA-CRT (see Algorithm 24), the peak of memory consumption is reached in the second exponentiation while s_p and c_p must be kept in memory. This makes a total memory consumption of $7.2\ l$ bits with on-the-fly chain computation and of $5\ l$ bits with pre-computed chain.

17.6.3 Comparison with previous solutions

We analyze hereafter the complexity of previous countermeasures in the literature. As explained in Section 15.8.4, these can be divided into two categories: the extended modulus based countermeasures and the self-secure exponentiations.

Extended modulus based countermeasures. The time complexity of an extended modulus based countermeasure (such as the Shamir's trick or the Vigilant's scheme) is around the complexity of the main exponentiation loop(s) since the additional computations are negligible. However, such countermeasures are not free in terms of timing since the use of an extended modulus slows down the exponentiation. In fact, the time complexity of a modular multiplication can be written as $l^2 t_0$ where t_0 denotes a constant time that depends on the device architecture. Denoting by k the bit-length of the modulus extension, an extended modulus exponentiation has a time complexity of $m(l+k)^2 t_0$ while a normal exponentiation has a time complexity of $ml^2 t_0$. Besides, the modulus extension implies an increase in the exponentiation execution time by a factor $(1 + k/l)^2$. As an illustration, Table 17.6.3 gives several values of the induced overhead according to the modulus length and to the extension length. For instance,

Table 17.3: Time overhead (in %) for an extended modulus based modular exponentiation.

	$l = 512$	$l = 768$	$l = 1024$
$k = 32$ (low security)	13	9	6
$k = 64$ (fair security)	27	17	13
$k = 80$ (high security)	34	22	16

an RSA 1024 implemented in CRT ($l = 512$) with extended modulus providing a fair level of security ($k = 64$) is about 27% slower than an unprotected one. This time overhead is sizeable; in particular it is significantly greater than the 10% overhead induced by our countermeasure. However, extended modulus based countermeasures enables the use of exponentiation algorithms faster that the square-and-multiply such as the q-ary or the sliding windows methods (see for instance [161]). Roughly, a q-ary exponentiation has a multiplications-per-bit ratio of $1 + (2^q - 1)/(q2^q)$ which is lower than or equal to 1.5, but it has a higher memory complexity since it requires $2^{q-1} + 1$ registers. The use of a sliding window allows to slightly improve the time complexity of a q-ary method [145].

The memory complexity of an exponentiation with modulus extension is of $n_r(l+k)$ where n_r denotes the number of registers required by the exponentiation algorithm. For an RSA-CRT, the memory com-

plexity depends on the used countermeasure. For the Vigilant's scheme, the memory consumption peak occurs during the second exponentiation while the values S'_p, i_{qr}, r, R_3 and R_4 must hold in memory (see [242]). This results in a memory consumption of $n_r(l+k)+(l+k)+3.5\ k = (n_r+1)\cdot l+(n_r+4.5)\cdot k$ bits.

Remark 17.12. We do not detail the memory complexity of the other extended modulus based countermeasures since, for most of them, it is close to the memory complexity of the Vigilant's scheme.

Previous self-secure exponentiations. The Giraud's scheme and the Boscher *et al.* 's scheme both have a multiplications-per-bit ratio constant to 2. This implies an average time overhead of 33% compared to the square-and-multiply algorithm and of 21% compared to our exponentiation. However, both of these schemes do not require additional computations contrary to the extended modulus based countermeasures or to our scheme when the chain is computed on-the-fly. Although these additional computations are theoretically negligible, they may induce an important overhead for a practical implementation depending on the device architecture.

In terms of memory, we shall focus on the Giraud's scheme since it is less consuming than the Boscher *et al.* 's scheme. The secure exponentiation requires two l-bit registers. For the RSA-CRT, the peak of memory consumption is reached during the two recombinations. For instance, the first recombination requires (at least) $3l$ bits of memory while m, S_p and S_q must hold in memory (see [109]) which makes a total complexity of $7l$ bits.

Comparison with our solution. Table 17.4 provides a comparison between the Giraud's scheme, the Vigilant's scheme and ours for an RSA 1024 with CRT (*i.e.* $l = 512$). For the Vigilant's scheme, we assume a modulus extension of $\{64, 80\}$ bits and a q-ary sliding window exponentiation for $q = 1, 2$ or 3 [161]. The results given in Table 17.4 show that our countermeasure is currently one of the most competitive solution to thwart fault analysis for an RSA 1024 with CRT.

Table 17.4: Memory and time complexities of different fault analysis countermeasures for an RSA 1024 with CRT.

Countermeasure	Time ($10^6 \cdot t_0$)	Memory (Kb)
Vigilant [242] ($q = 1$)	$\{484, 511\}$	$\{2.3, 2.4\}$
Vigilant [242] ($q = 2$)	$\{444, 468\}$	$\{2.5, 2.6\}$
Vigilant [242] ($q = 3$)	$\{417, 440\}$	$\{3.6, 3.7\}$
Giraud [109]	537	3.5
Ours	443	2.5 (+1.1)

Remark 17.13. The time complexity for the Vigilant's scheme with sliding widow is computed as follows. A q-ary exponentiation performs an average of $l \cdot \big(1 + (2^q - 1)/(q2^q)\big)$ multiplications [161] and the use of a sliding window yields an improvement of about 5% for $l = 512$ [145]. Therefore, the time complexity of one exponentiation is estimated to $0.95 \cdot (l + k)^2 t_0 \cdot l \cdot \big(1 + (2^q - 1)/(q2^q)\big)$. Concerning the memory complexity, the sliding window method requires a total of $n_r = 2^{q-1} + 1$ registers.

Chapter 18

How to implement coherence checks?

Contents

18.1 Introduction

In this chapter, we address the problem of implementing a coherence check in a secure way with respect to fault attacks. As we have seen earlier, most countermeasures against fault analysis are based on the addition of redundancy in the computation in order to check its coherence afterwards. Such a check is hence a key point of the security of fault analysis countermeasures in practice. It is therefore important to implement coherence checks in such a way that they cannot be skipped by a fault injection.

The results presented in this chapter have been published in collaboration with Emmanuelle Dottax, Christophe Giraud and Yannick Sierra in the international workshop in *Information Security Theory and Practices (WISTP 2009)* [3]. The proposed method has also been patented in [12].

18.2 Infective procedures *vs.* checking procedures

Most fault analysis countermeasures can be summarized according to the following principle: some redundancy is included in the computation in order to provide an error detection code c that is expected to take a certain value $c^\star$. At the end of the computation, the code is involved to check the coherence: if the expected value is obtained then the computation is considered correct. In that case, the result of the computation is well returned otherwise an error message is returned. This avoids the exposure of faulty results and hence prevents fault attacks. This general principle is illustrated in Figure 18.1. For example, the redundancy can be a simple doubling of the computation. In that case, the error detection code c is the second result and its expected value $c^\star$ is the first one. Depending on the cryptographic algorithm, more efficient solutions exist; the countermeasure for RSA presented in the previous chapter is an example.

The natural way to implement a coherence check is to perform a simple comparison followed by a conditional branch:

$$\textbf{if } (c = c^\star) \textbf{ then return } \mathit{result} \textbf{ else return } \text{“error”}$$

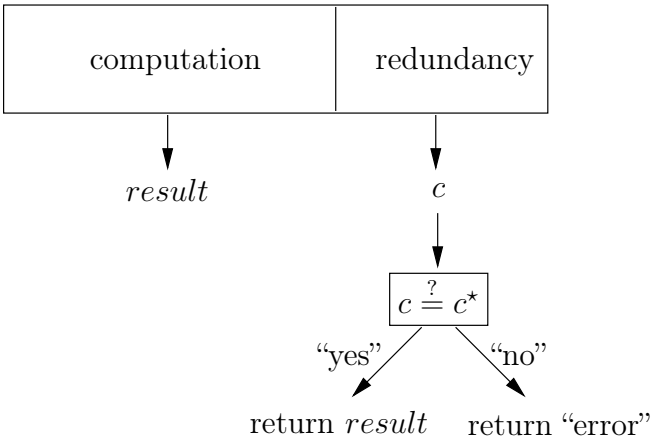


Figure 18.1: General principle of fault analysis countermeasures.

However some authors argued that such simple *checking procedures* could be easily overcome by fault injection [249]. In fact, when a conditional branch is executed, the CPU tests the value of some bit of the status register which depends on the result of the previous instructions. According to the value of this bit the program either executes the subsequent instructions or it jumps at a given address in the program code. If a fault is induced in the status register during a conditional branch, the value of the tested bit may be flipped. In that case, the wrong code section is executed. This way, an attacker may corrupt the cryptographic computation based on a first fault injection and then, based on a second fault injection, fool the coherence check such that the erroneous result is returned instead of the error message. Such an attack is called a *second-order fault attack* since it requires the injection of two different faults in two different parts of the computation (see Section 18.3).

In order to prevent such attacks, the authors of [249] introduced the concept of *infective computation*. Rather than checking the coherence of the computation, the algorithm is designed in such a way that any fault corrupting the computation yields a faulty result that is not exploitable by an attacker *i.e.* that does not provide any information about the secret key. Such an infective countermeasure for RSA-CRT was proposed in [249] but it has been shown insecure in [247]. The concept of infective computation was then applied to classical error detection schemes as replacement of checking procedures. Instead of comparing the error detection code c to its expected value $c^\star$, an *infective procedure* is involved that ensures the infection of the result when c differs from $c^\star$. Namely a function $f(c, c^\star, result)$ is always returned that satisfies:

$$f(c, c^\star, result) = \begin{cases} result & \text{if } c = c^\star, \\ dummy & \text{otherwise,} \end{cases}$$

where *dummy* denotes a dummy value useless for an attacker.

Two schemes for RSA-CRT based on the Shamir's trick (see Section 15.8.4) have been proposed that make use of infective procedures: the BOS scheme [47] and the Ciet and Joye scheme [72]. Unfortunately, these two schemes have been broken due to flaws in the infective procedures [38, 243]. In both cases, the faults are effectively detected but the infected results $f(c, c^\star, result)$ leak some information enabling key recovery. Special care must hence be taken while designing an infective procedure so that infected results do not leak any information. Moreover, it must be noticed that even a secure infective procedure (in the sense that infected results do not leak information) may be skipped just as a checking procedure. Namely, face to a secure infective procedure a second-order fault analysis may still be attempted.

18.3 Second-order fault analysis

An algorithm is said to be first-order resistant when it contains some countermeasure which ensures that any single error occurring in its execution is not exploitable by an attacker. When a single fault

is injected and corrupts the cryptographic computation, it is systematically detected (or at least with high probability) and, based on a checking/infective procedure, the faulty result is either destroyed or infected.

However, an attacker may defeat such a countermeasure by using a second-order fault analysis, namely by injecting two faults during the execution. In that case, one of these faults must be dedicated to the corruption of the cryptographic computation in order to produce an exploitable faulty result. The second fault is then used to render the countermeasure ineffective. For such a purpose, two approaches are possible:

— In the first approach, the attacker tries to fool the error detection mechanism. Namely, the second fault aims at covering the effects of the first fault in such a way that the error detection mechanism does not detect it while the result of the cryptographic computation remains faulty. To do so, the attacker needs to precisely control the two fault injections effects. This implies a strong and not very practical adversary model.

— The second approach consists in directly skipping the coherence verification (or the infection procedure). Since several experiments have demonstrated the practicability of skipping the execution of one or more operations (see for instance [32, 140, 149]), this approach corresponds to a weaker model of adversary and it is then more natural from a practical point of view.

In the following, the fault model corresponding to the latter approach will be referred to as the *corrupt-and-skip* fault model. We formalize it hereafter.

Fault model (corrupt-and-skip). *Two faults are injected. The first fault can be of any type (instructions skip, memory modification, etc.), provided it corrupts the cryptographic computation and produces a faulty result. The second fault allows the attacker to skip any operation or set of contiguous operations in order to circumvent the coherence check (or the infective procedure).*

The literature contains several practical examples of fault attacks. However, to the best of our knowledge, the paper [140] is the first one that reports a successful experiment of second-order fault attacks. In this paper, Chong Hee Kim and Jean-Jacques Quisquater explain how they practically mounted some corrupt-and-skip fault attacks against the first-order resistant CRT-RSA implementations proposed in [72] and [108]. Their work also includes improvements of the latter first-order countermeasures to achieve a secure implementation against corrupt-and-skip attacks. Later, the same authors proposed in [141] an implementation also meant to resist this kind of attacks. Unfortunately, we have shown in [3] that the proposed countermeasures are actually not intrinsically resistant in the corrupt-and-skip model and that they may be successfully attacked. In order to circumvent the lack of secure solutions, we have proposed in [3] a generic way to implement a coherence check that is secure against any attack in the corrupt-and-skip model. This method is presented in the next section.

18.4 A generic method to check coherence

In this section, we describe a generic method to counteract corrupt-and-skip second-order fault attacks based on an error detection scheme secure against first-order fault analysis.

18.4.1 Description

As described in Section 18.2, we assume that the error detection scheme computes some redundant value c which is involved in the checking (or infective) procedure. Thus c is expected to take a given value, denoted $c^\star$, otherwise the checking (resp. infective) procedure returns an error (resp. infects the result).

Our countermeasure is based on a simple mechanism that advantageously replaces the checking (resp. infective) procedure: given c and its expected value $c^\star$, we perform the check $c \stackrel{?}{=} c^\star$ twice while inserting in between a simple but pivotal statement. The whole procedure is described in the following algorithm.

Algorithm 27 Lock
INPUT: $result$, c and $c^\star$
OUTPUT: $result$ if $c = c^\star$ and 0 otherwise

1. **if** $(c \neq c^\star)$ **then** $result \leftarrow 0$
2. $tmp \leftarrow result$
3. **if** $(c \neq c^\star)$ **then** $tmp \leftarrow 0$
4. **return** tmp

Remark 18.1. The conditional branches in Steps 1 and 3 can be avoided (although not necessary to thwart corrupt-and-skip second-order fault attacks). Instead, a function $f(c, c^\star, \cdot)$ such that:

$$f(c, c^\star, x) = \begin{cases} x & \text{if } c = c^\star \\ 0 & \text{otherwise} \end{cases}$$

can be implemented using basic instructions. Such an implementation is detailed in [3].

The whole solution based on the Lock procedure follows the series of steps hereafter. First, the buffer $result$ is initialized at 0. Then the fault analysis resistant cryptographic algorithm is executed: from a message m and a key k it computes the result of the cryptographic algorithm and a couple of values $(c, c^\star)$ depending on the fault analysis countermeasure. Afterwards, the Lock procedure described in Algorithm 27 is executed and $result$ is eventually returned. If $c = c^\star$ then Lock returns the content $result$ otherwise it returns 0. The overall cryptographic implementation resistant against corrupt-and-skip fault attacks is described in Algorithm 28.

Algorithm 28 KQ-attack resistant RSA
INPUT: m, $\mathcal{P}$
OUTPUT: $m^d \bmod N$

1. $result \leftarrow 0$
2. $(result, c, c^\star) \leftarrow$ FA-Res-CryptoAlgo(m, k)
3. **return** Lock$(result, c, c^\star)$

This generic solution can be applied to any fault analysis countermeasure based on an error detection scheme and it is resistant to any corrupt-and-skip attack as shown in the next section.

18.4.2 Security analysis

In the corrupt-and-skip fault model, a first fault is dedicated to the corruption of the cryptographic computation and a second fault aims to avoid the erasure of the faulty result by skipping some operations. According to this model, we assume that an attacker can:

— inject a fault in Step 2 of Algorithm 28 producing a faulty result and a faulty pair of checking values $(\widetilde{c}, \widetilde{c^\star})$ such that $\widetilde{c} \neq \widetilde{c^\star}$ (this results from the soundness of the first-order countermeasure that is used),

— skip a set of contiguous operations of the overall computation.

We demonstrate hereafter that the skipping of any set of contiguous operations of Algorithm 28 cannot prevent the Lock procedure from erasing the faulty result $\widetilde{S}$ (or returning an unexploitable result) while the faulty checking values $\widetilde{c}$ and $\widetilde{c^\star}$ are different.

Let us first assume that the adversary skips the entire Step 2 of the Lock procedure. In this case, Algorithm 28 returns the initialization value of tmp which is unexploitable. On the other hand, if Step 2 of the Lock procedure is not entirely skipped, then either all the previous operations or all the following operations are properly executed since the set of skipped operations is contiguous. As a result, either Step 1 or Step 3 of the Lock procedure is executed which ensures the result erasure in case of fault detection (*i.e.* if $\widetilde{c}$ and $\widetilde{c^\star}$ are different).

To conclude, any corrupt-and-skip second-order fault attack implies the return of an unexploitable output and the attacker gains no sensitive information.

Chapter 19

Conclusions and perspectives

In this third part, we have investigated fault analysis. After presenting the principle of these attacks, we have reviewed several attacks and countermeasures for block ciphers and for RSA. We have seen that securing these algorithms against fault analysis requires the implementation of error detection schemes to prevent the exposure of faulty results.

The simplest way to detect if an error has corrupted a computation is to perform the computation twice and to check whether the two obtained results are equal. Such a solution implies a doubling of the execution time (or of the circuit size in hardware). Doubling the execution time is conceivable for block ciphers implementation as they are usually quite fast. Nevertheless, an improved solution consists in only doubling certain rounds of the cipher. In fact, fault attacks against block ciphers usually target the few last rounds or the few initial rounds of the cipher. Consequently, the overall computation does not need to be doubled. However, one must be careful while protecting a limited number of rounds to avoid the induction of any security flaw. We investigated this issue for the DES cipher for which we described an attack on the middle rounds. Our attack makes it possible to break DES by inducing some faults at the end of rounds 12, 11, 10 and 9, more or less efficiently depending on the round number and the fault model. These results allowed us to refine the lower bounds on the number of rounds of DES that should be protected against fault analysis.

We then addressed the RSA cryptosystem which is a privileged target of fault attacks. Contrary to block ciphers, a doubling of the computation time is often prohibitive in the case of RSA. As a consequence, some alternative countermeasures were developed during the past years. Two approaches have been mainly followed: the modulus extension based countermeasures that include redundancy in modular operations and the self-secure exponentiation algorithms that include redundancy at the exponentiation level. Both approaches have some assets and drawbacks depending on the implementation context. The two self-secure exponentiation algorithms that have been previously proposed have a significant timing overhead compared to classical exponentiation algorithms. Indeed, they require to perform 2 multiplications per bit of the exponent whereas a classical exponentiation requires 1.5 multiplications per bit of the exponent on average. Our researches yield a new self-secure exponentiation algorithm which requires an average of 1.65 multiplications per bit of the exponent. The core idea of our method is to use a double exponentiation that computes two different powers of a given element. Therefore, we introduced a double exponentiation algorithm which is based on the construction of a double addition chain for the underlying pair of exponents. We analyzed the security of our solution *vs.* fault analysis and we showed how it can be protected against side channel analysis. We also studied the time and memory complexities of our countermeasure and we showed that it offers an efficient alternative to the existing schemes. A direction for further research would be to investigate more efficient double exponentiation algorithms possibly with the use of pre-computed powers.

Finally, our investigations about fault analysis focused on a practical issue: the implementation of coherence checks. As we have seen, most countermeasures against fault analysis are based on the addition of redundancy in the computation in order to check its coherence afterwards. Such a check is therefore a key point of the security of fault analysis countermeasures in practice. Two approaches have been followed in the literature for the implementation of coherence checks: simple checking procedures involving a test

and conditional branch and infective procedures rendering faulty results useless. However both approaches have been shown to be vulnerable to second-order fault attacks corrupting the computation and skipping the checking/infective procedure through a second fault injection. We proposed a simple method to implement a coherence check which counteracts these attacks.

Bibliography

Publications

[1] J.-S. Coron, C. Giraud, E. Prouff, and M. Rivain. Attack and Improvement of a Secure S-Box Calculation Based on the Fourier Transform. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2008. (Best Paper Award).

[2] J.-S. Coron, E. Prouff, and M. Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007. (Best Paper Award).

[3] E. Dottax, C. Giraud, M. Rivain, and Y. Sierra. On Second-Order Fault Analysis Resistance for CRT-RSA Implementations. In O. Markowitch, A. Bilas, J.-H. Hoepman, C. J. Mitchell, and J.-J. Quisquater, editors, *Information Security Theory and Practices – WISTP 2009*, volume 5746 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2009. (Best Paper Award).

[4] E. Prouff and M. Rivain. A Generic Method for Secure SBox Implementation. In S. Kim, M. Yung, and H.-W. Lee, editors, *Information Security Applications – WISA 2007*, volume 4867 of *Lecture Notes in Computer Science*, pages 227–244. Springer, 2008.

[5] 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 – ANCS 2009*, volume 5536 of *Lecture Notes in Computer Science*, pages 499–518. Springer, 2009.

[6] E. Prouff, M. Rivain, and R. Bévan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Comput.*, 58(6):799–811, 2009.

[7] M. Rivain. On the Exact Success Rate of Side Channel Analysis in the Gaussian Model. In R. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography – SAC 2008*, Lecture Notes in Computer Science. Springer, 2008.

[8] M. Rivain. Differential Fault Analysis on DES Middle Rounds. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 457–469. Springer, 2009.

[9] M. Rivain. Securing RSA against Fault Analysis by Double Addition Chain Exponentiation. In M. Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 459–480. Springer, 2009.

[10] M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In T. Baignères and S. Vaudenay, editors, *Fast Software Encryption – FSE 2008*, Lecture Notes in Computer Science, pages 127–143. Springer, 2008.

[11] M. Rivain, E. Prouff, and J. Doget. Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

Patents

[12] E. Dottax, C. Giraud, M. Rivain, and Y. Sierra. Procédé de traitement conditionnel de données protégé contre les attaques par génération de fautes et dispositif associé. Under examination. Filed on July 24th 2008. Application number: FR0855042.

[13] M. Rivain. Procédé de traitement de données impliquant une exponentiation et un dispositif associé. Under examination. Filed on February 24th 2009. Application number: FR0951166.

[14] M. Rivain and E. Prouff. Method for accessing a sub-word in a binary word, and related device and software. Filed on December 13th 2007. Publication number: WO2009074727, 2009.

[15] M. Rivain and E. Prouff. Method for cryptographic data processing, particularly using an S-box, and related device and software. Filed on December 13th 2007. Publication number: WO2009074728, 2009.

[16] M. Rivain and E. Prouff. Method for cryptographic data processing, particularly using an S-box, and related device and software. Filed on December 13th 2007. Publication number: WO2009074726, 2009.

References

[17] O. Aciiçmez, Çetin Kaya Koç, and J.-P. Seifert. Predicting Secret Keys Via Branch Prediction. In M. Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2007.

[18] D. Agrawal, J. Rao, and P. Rohatgi. Multi-channel Attacks. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2003.

[19] M.-L. Akkar. *Attaques et méthodes de protections de systèmes cryptographiques embarqués*. PhD thesis, Université de Versailles Saint-Quentin, Oct. 1er, 2004.

[20] M.-L. Akkar, R. Bévan, P. Dischamp, and D. Moyart. Power Analysis, What is Now Possible. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 489–502. Springer, 2000.

[21] M.-L. Akkar, R. Bévan, and L. Goubin. Two Power Analysis Attacks against One-Mask Method. In B. Roy and W. Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2004.

[22] M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

[23] M.-L. Akkar and L. Goubin. A Generic Protection against High-order Differential Power Analysis. In T. Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 192–205. Springer, 2003.

[24] 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, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 223–236. Springer, 2006.

[25] 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 – SAC 2007*, Lecture Notes in Computer Science, pages 110–125. Springer, 2007.

[26] R. Anderson and M. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In B. Christianson, B. Crispo, T. Mark, A. Lomas, and M. Roe, editors, 5*th Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997.

[27] ANSI X9.62. *Public Key Cryptography for The Financial Service Industry : The Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standards Institute, 1998.

[28] C. Archambeau, E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Template Attacks in Principal Subspaces. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[29] S. Aumônier. Generalized Correlation Power Analysis. Published in the Proceedings of the Ecrypt Workshop Tools For Cryptanalysis 2007, 2007.

[30] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.

[31] F. Bao, R. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T.-H. Ngair. Breaking Public Key Cryptosystems and Tamper Resistance Devices in the Presence of Transient Fault. In *5th Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 115–124. Springer, 1997.

[32] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *IEEE*, 94(2):370–382, 2006.

[33] J. Beirlant, E. J. Dudewicz, L. Györfi, and E. C. Meulen. Nonparametric entropy estimation: An overview. *International Journal of the Mathematical Statistics Sciences*, 6:17–39, 1997.

[34] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.

[35] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Symposium on Foundations of Computer Science – FOCS'97*, pages 394–403, 1997.

[36] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, 1996.

[37] A. Berzati, C. Canovas, J.-G. Dumas, and L. Goubin. Fault Attacks on RSA Public Keys: Left-To-Right Implementations Are Also Vulnerable. In M. Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 414–6428. Springer, 2009.

[38] A. Berzati, C. Canovas, and L. Goubin. (In)security Against Fault Injection Attacks for CRT-RSA Implementations. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2008*, pages 101–107. IEEE Computer Society, 2008.

[39] A. Berzati, C. Canovas, and L. Goubin. Perturbating RSA Public Keys: An Improved Attack. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2008.

[40] R. Bévan and E. Knudsen. Ways to Enhance Power Analysis. In P. Lee and C. Lim, editors, *Information Security and Cryptology – ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2002.

[41] I. Biehl, B. Meyer, and V. Müller. Differential Fault Analysis on Elliptic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2000.

[42] E. Biham, L. Granboulan, and P. Nguyen. Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. In H. Handschuh and H. Gilbert, editors, *Fast Software Encryption – FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 359–367. Springer, 2005.

[43] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

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

[45] J. Black and P. Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2000.

[46] J. Blömer, J. G. Merchan, and V. Krummel. Provably Secure Masking of AES. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.

[47] J. Blömer, M. Otto, and J.-P. Seifert. A New RSA-CRT Algorithm Secure against Bellcore Attacks. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM Conference on Computer and Communications Security – CCS'03*, pages 311–320. ACM Press, 2003.

[48] J. Blömer and J.-P. Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard. In R. Wright, editor, *Financial Cryptography – FC 2003*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.

[49] A. Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 2008.

[50] D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[51] D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology*, 14(2):101–119, 2001.

[52] M. Boreale. Attacking Right-to-Left Modular Exponentiation with Timely Random Faults. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2006.

[53] J. Bos and M. Coster. Addition Chain Heuristics. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 400–407. Springer, 1989.

[54] A. Boscher and H. Handschuh. Masking Does Not Protect Against Differential Fault Attacks. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2008*, pages 35–40. IEEE Computer Society, 2008.

[55] A. Boscher, R. Naciri, and E. Prouff. CRT RSA Algorithm Protected against Fault Attacks. In D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *Information Security Theory and Practices – WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2007.

[56] E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier. Why One Should Also Secure RSA Public Key Elements. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2006.

[57] E. Brier, C. Clavier, and F. Olivier. Optimal Statistical Power Analysis. Cryptology ePrint Archive, Report 2003/152, 2003. `http://eprint.iacr.org/`.

[58] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[59] E. Brier, H. Handschuh, and C. Tymen. Fast Primitives for Internal Data Scrambling in Tamper Resistant Hardware. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 2001.

[60] C. Cachin. *Entropy Measures and Unconditional Security in Cryptography.* PhD thesis, Swiss Federal Institute of Technology, 1997.

[61] C. Canovas and J. Clediere. What do S-boxes Say in Differential Side Channel Attacks? Cryptology ePrint Archive, Report 2005/311, 2005. `http://eprint.iacr.org/`.

[62] D. Canright and L. Batina. A Very Compact "Perfectly Masked" S-Box for AES. In S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security – ACNS 2008*, volume 5037 of *LNCS*, pages 446–459, 2008.

[63] A. Canteaut and M. Trabbia. Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer, 2000.

[64] M. Carreira-Perpinan. Mode-finding for mixtures of Gaussian distributions. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 22(11):1318–1323, November 2000.

[65] S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[66] S. Chari, J. Rao, and P. Rohatgi. Template Attacks. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–29. Springer, 2002.

[67] C.-N. Chen and S.-M. Yen. Differential Fault Analysis on AES Key Schedule and Some Countermeasures. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy - 8th Australasian Conference – ACISP 2003*, volume 2727 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2003.

[68] H. Chen, W. Wu, and D. Feng. Differential Fault Analysis on CLEFIA. In S. Qing, H. Imai, and G. Wang, editors, *Information and Communications Security, 9th International Conference – ICICS 2007*, volume 4861 of *Lecture Notes in Computer Science*, pages 284–295. Springer, 2008.

[69] Z. Chen and Y. Zhou. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2006.

[70] B. Chevallier-Mames. Self-Randomized Exponentiation Algorithms. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2004.

[71] 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.

[72] M. Ciet and M. Joye. Practical Fault Countermeasures for Chinese Remaindering Based RSA. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'05*, pages 124–132, 2005.

[73] C. Clavier. Secret External Encodings Do Not Prevent Transient Fault Analysis. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[74] C. Clavier, J.-S. Coron, and N. Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000.

[75] C. Clavier, B. Gierlichs, and I. Verbauwhede. Fault Analysis Study of IDEA. In T. Malkin, editor, *Topics in Cryptology – CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 2008.

[76] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.

[77] J.-S. Coron. A New DPA Countermeasure Based on Permutation Tables. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *Security and Cryptography for Networks, 6th International Conference, SCN 2008*, volume 5229 of *Lecture Notes in Computer Science*, pages 278–292. Springer, 2008.

[78] J.-S. Coron and L. Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 231–237. Springer, 2000.

[79] J.-S. Coron and I. Kizhvatov. An Efficient Method for Random Delay Generation in Embedded Software. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2009.

[80] J.-S. Coron, P. Kocher, and D. Naccache. Statistics and Secret Leakage. In Y. Frankel, editor, *Financial Cryptography – FC 2000*, volume 1962 of *Lecture Notes in Computer Science*. Springer, 2000.

[81] J.-S. Coron and A. May. Deterministic Polynomial-Time Equivalence of Computing the RSA Secret Key and Factoring. *J. Cryptology*, 20(1):39–50, 2007.

[82] J.-S. Coron and A. Tchulkine. A New Algorithm for Switching from Arithmetic to Boolean Masking. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 89–97. Springer, 2003.

[83] N. Courtois and L. Goubin. An Algebraic Masking Method to Protect AES against Power Attacks. In D. Won and S. Kim, editors, *Information Security and Cryptology – ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 199–209. Springer, 2006.

[84] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, 2002.

[85] B. den Boer, K. Lemke, and G. Wicke. A DPA Attack against the Modular Reduction within a CRT Implementation of RSA. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2002.

[86] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the Timing Attack. In J.-J. Quisquater, editor, *CARDIS 1998*, volume 1820 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2000.

[87] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transaction on Information Theory*, 22(6):644–654, Nov. 1976.

[88] N. Draper and H. Smith. *Applied Regression Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, 1998.

[89] 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 – ANCS 2003*, volume 2846 of *Lecture Notes in Computer Science*, pages 293–306. Springer, 2003.

[90] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, C. Anguille, M. Bardouillet, C. Buatois, and J.-B. Rigaud. Hardware Engines for Bus Encryption: A Survey of Existing Techniques. *CoRR*, abs/0710.4803, 2007.

[91] FIPS PUB 186. *Digital Signature Standard*. National Institute of Standards and Technology, 1994.

[92] FIPS PUB 197. *Advanced Encryption Standard*. National Institute of Standards and Technology, Nov. 26, 2001.

[93] FIPS PUB 46. *The Data Encryption Standard*. National Bureau of Standards, Jan. 15, 1977.

[94] FIPS PUB 46-3. *Data Encryption Standard (DES)*. National Institute of Standards and Technology, Oct. 25, 1999.

[95] P.-A. Fouque and F. Valette. The Doubling Attack: Why Upwards is Better than Downwards. In C. Walter, Ç. 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.

[96] G. Fumaroli, E. Mayer, and R. Dubois. First-Order Differential Power Analysis on the Duplication Method. In K. Srinathan, C. P. Rangan, and M. Yung, editors, *Progress in Cryptology – INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 210–223. SV, 2007.

[97] T. E. Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In G. Blakley and D. Chaum, editors, *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[98] B. M. Gammel and S. Mangard. On the Duality of Probing and Fault Attacks. Cryptology ePrint Archive, Report 2009/352, 2009. `http://eprint.iacr.org/`.

[99] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[100] L. Genelle, C. Giraud, and E. Prouff. Securing AES Implementation Against Fault Attacks. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2009*. IEEE Computer Society, 2009.

[101] A. Genz. Numerical Computation of Multivariate Normal Probabilities. *Journal of Computational and Graphical Statistics*, 1:141–149, 1992.

[102] A. Genz. Comparison of Methods for the Computation of Multivariate Normal Probabilities. *Computing Science and Statistics*, 25:400–405, 1993.

[103] B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede. Revisiting Higher-Order DPA Attacks: Multivariate Mutual Information Analysis. Cryptology ePrint Archive, Report 2009/228, 2009. `http://eprint.iacr.org/`.

[104] B. Gierlichs, L. Batina, and P. Tuyls. Mutual Information Analysis – A Universal Differential Side-Channel Attack. Cryptology ePrint Archive, Report 2007/198, 2007. `http://eprint.iacr.org/`.

[105] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[106] B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. Stochastic Methods. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[107] C. Giraud. DFA on AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Fourth Conference on the Advanced Encryption Standard (AES) – AES 4*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2005.

[108] C. Giraud. Fault Resistant RSA Implementation. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'05*, pages 142–151, 2005.

[109] C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, Sept. 2006.

[110] C. Giraud. *Attaques de cryptosystèmes embarqués et contre-mesures associées.* Thèse de doctorat, Université de Versailles, Oberthur Card Systems, Oct. 2007.

[111] O. Goldreich. A Uniform-Complexity Treatment of Encryption and Zero-Knowledge. *J. Cryptology*, 6(1):21–53, 1993.

[112] S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[113] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[114] J. Golić and C. Tymen. Multiplicative Masking and Power Analysis of AES. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.

[115] J. D. Golic. DeKaRT: A New Paradigm for Key-Dependent Reversible Circuits. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2003.

[116] D. M. Gordon. A Survey of Fast Exponentiation Methods. *J. Algorithms*, 27(1):129–146, 1998.

[117] L. Goubin. A Sound Method for Switching between Boolean and Arithmetic Masking. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2001.

[118] L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[119] S. Govindavajhala and A. Appel. Using Memory Errors to Attack a Virtual Machine. In *IEEE Symposium on Security and Privacy*, pages 154–165. IEEE Computer Society, 2003.

[120] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *Proceedings of the Third SIAM International Conference on Data Mining*. SIAM, 2003.

[121] S. Guilley, P. Hoogvorst, Y. Mathieu, and R. Pacalet. The "Backend Duplication" Method. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 383–397. Springer, 2005.

[122] S. Guilley, L. Sauvage, P. Hoogvorst, R. Pacalet, G. M. Bertoni, and S. Chaudhuri. Security Evaluation of WDDL and SecLib Countermeasures against Power Attacks. *Computers, IEEE Transactions on*, 57(11):1482–1497, november 2008.

[123] 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.

[124] P. Herbst, E. Oswald, and S. Mangard. An AES Smart Card Implementation Resistant to Power Analysis Attacks. In J. Zhou, M. Yung, and F. Bao, editors, *Applied Cryptography and Network Security – ANCS 2006*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 2006.

[125] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 308–327. Springer, 2006.

[126] Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[127] K. Itoh, T. Izu, and M. Takenak. Address-bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2002.

[128] K. Itoh, T. Izu, and M. Takenaka. A Practical Countermeasure against Address-Bit Differential Power Analysis. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 382–396. Springer, 2003.

[129] I. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002.

[130] N. Joshi, K. Wu, and R. Karri. Concurrent Error Detection Schemes for Involution Ciphers. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 400–412. Springer, 2004.

[131] M. Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2007.

[132] M. Joye, A. Lenstra, and J.-J. Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *Journal of Cryptology*, 12(4):241–245, 1999.

[133] M. Joye, P. Paillier, and B. Schoenmakers. On Second-order Differential Power Analysis. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[134] M. Joye, J.-J. Quisquater, F. Bao, and R. Deng. RSA-type Signatures in the Presence of Transient Faults. In M. Darnell, editor, *Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 155–160. Springer, 1997.

[135] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.

[136] P. Junod and S. Vaudenay. FOX: a New Family of Block Ciphers. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2004.

[137] M. Karpovsky, K. Kulikowski, and A. Taubin. Differential Fault Analysis Resistant Architectures for the AES. In J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A. E. Kalam, editors, *Smart Card Research and Advanced Applications VI – CARDIS 2004*. Kluwer Academic Publishers, 2004.

[138] R. Karri, G. Kuznetsov, and M. Gössel. Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2003.

[139] M. M. Kermani and A. Reyhani-Masoleh. A Lightweight Concurrent Fault Detection Scheme for the AES S-Boxes Using Normal Basis. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2008.

[140] C. H. Kim and J.-J. Quisquater. Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *Information Security Theory and Practices – WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2007.

[141] C. H. Kim and J.-J. Quisquater. How Can We Overcome Both Side Channel Analysis and Fault Attack on RSA-CRT? In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 21–29. IEEE Computer Society, 2007.

[142] C. H. Kim and J.-J. Quisquater. New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough. In G. Grimaud and F.-X. Standaert, editors, *Smart Card Research and Advanced Applications, 8th International Conference – CARDIS 2008*, volume 5189 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2008.

[143] C. H. Kim, J. H. Shin, J.-J. Quisquater, and P. J. Lee. Safe-Error Attack on SPA-FA Resistant Exponentiations Using a HW Modular Multiplier. In K.-H. Nam and G. Rhee, editors, *Information Security and Cryptology – ICISC 2007*, volume 4817 of *Lecture Notes in Computer Science*. Springer, 2007.

[144] D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, third edition, 1988.

[145] Ç. Koç. Analysis of the Sliding Window Techniques for Exponentiation. *Computer & Mathematics with applications*, 30(10):17–24, 1995.

[146] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[147] P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc., 1998. `http://www.cryptography.com/resources/whitepapers/DPATechInfo.pdf`.

[148] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[149] O. Kommerling and M. Kuhn. Design Principles for Tamper Resistant Smartcard Processors. In *the USENIX Workshop on Smartcard Technology (Smartcard '99)*, pages 9–20, 1999.

[150] K. J. Kulikowski, M. G. Karpovsky, and A. Taubin. Power Attacks on Secure Hardware Based on Early Propagation of Data. In *International On-Line Testing Symposium – IOLTS 2006*, pages 131–138. IEEE Computer Society, 2006.

[151] K. Lemke, K. Schramm, and C. Paar. DPA on n-Bit Sized Boolean and Arithmetic Operations and Its Application to IDEA, RC6, and the HMAC-Construction. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 2004.

[152] K. Lemke-Rust and C. Paar. Gaussian Mixture Models for Higher-Order Side Channel Analysis. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2007.

[153] J. Lv and Y. Han. Enhanced DES Implementation Secure against High-order Differential Power Analysis in Smartcards. In C. Boyd and J. M. González Nieto, editors, *Information Security and Privacy - 10th Australasian Conference – ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 195–206. Springer, 2005.

[154] F. Macé, F.-X. Standaert, and J.-J. Quisquater. Information Theoretic Evaluation of Side-Channel Resistant Logic Styles. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2007.

[155] T. Malkin, F.-X. Standaert, and M. Yung. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 159–172. Springer, 2006.

[156] S. Mangard. Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[157] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks – Revealing the Secrets of Smartcards.* Springer, 2007.

[158] S. Mangard, T. Popp, and B. M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[159] S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[160] J. Massey. Guessing and Entropy. *IEEE International Symposium on Information Theory*, page 204, 1994.

[161] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997. `http://www.cacr.math.uwaterloo.ca/hac/`.

[162] T. Messerges. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms.* PhD thesis, University of Illinois, 2000.

[163] T. Messerges. Securing the AES Finalists against Power Analysis Attacks. In B. Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2000.

[164] T. Messerges. Using Second-order Power Analysis to Attack DPA Resistant Software. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

[165] T. Messerges, E. Dabbish, and R. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *the USENIX Workshop on Smartcard Technology (Smartcard '99)*, pages 151–161, 1999.

[166] T. Messerges, E. Dabbish, and R. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcard. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999.

[167] D. S. Mitrinovic, J. Sándor, and B. Crstici. *Handbook of Number Theory.* Springer, 1995.

[168] B. Möller. Algorithms for Multi-exponentiation. In S. Vaudenay and A. Youssef, editors, *Selected Areas in Cryptography – SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2001.

[169] P. Montgomery. Modular Multiplication Without Trial Division. *Math. Comp.*, 44(170):519–521, Apr. 1985.

[170] A. Moradi, N. Mousavi, C. Paar, and M. Salmasizadeh. A Comparative Study of Mutual Information Analysis under a Gaussian Assumption. To appear in WISA 2009, 2009.

[171] R. Moreno. Procédé et dispositif de commande électronique. Numéro de publication: FR2266222, Mar. 25, 1974.

[172] J. A. Muir. Seifert's RSA Fault Attack: Simplified Analysis and Generalizations. In P. Ning, S. Qing, and N. Li, editors, *International Conference on Information and Communications Security – ICICS'06*, volume 4307 of *Lecture Notes in Computer Science*, pages 420–434. Springer, 2006.

[173] O. Neiße and J. Pulkus. Switching Blindings with a View Towards IDEA. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 230–239. Springer, 2004.

[174] S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In P. J. Lee and J. H. Cheon, editors, *Information Security and Cryptology – ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.

[175] E. Oswald and S. Mangard. Template Attacks on Masking—Resistance is Futile. In M. Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 2007.

[176] E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical Second-order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2006.

[177] E. Oswald, S. Mangard, and N. Pramstaller. Secure and Efficient Masking of AES – A Mission Impossible ? Cryptology ePrint Archive, Report 2004/134, 2004. `http://eprint.iacr.org/`.

[178] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A Side-Channel Analysis Resistant Description of the AES S-box. In H. Handschuh and H. Gilbert, editors, *Fast Software Encryption – FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2005.

[179] E. Oswald and K. Schramm. An Efficient Masking Scheme for AES Software Implementations. In J. Song, T. Kwon, and M. Yung, editors, *Information Security Applications – WISA 2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 292–305. Springer, 2006.

[180] M. Otto. *Fault Attacks and Countermeasures.* PhD thesis, University of Paderborn, Dec. 2004.

[181] D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Cryptology ePrint Archive, Report 2002/169, 2002. `http://eprint.iacr.org/`.

[182] J. Patarin. How to Construct Pseudorandom and Super Pseudorandom Permutation from one Single Pseudorandom Function. In R. Rueppel, editor, *Advances in Cryptology – EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1992.

[183] E. Peeters, F.-X. Standaert, N. Donckers, and J.-J. Quisquater. Improved Higher-order Side-Channel Attacks with FPGA Experiments. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2005.

[184] C. Petit, F.-X. Standaert, O. Pereira, T. Malkin, and M. Yung. A Block Cipher Based Pseudo Random Number Generator Secure Against Side-Channel Key Recovery. In M. Abe and V. D. Gligor, editors, *Symposium on Information, Computer and Communications Security – ASIACCS 2008*, pages 56–65. ACM, 2008.

[185] R. Phan and S.-M. Yen. Amplifying Side-Channel Attacks with Techniques from Block Cipher Cryptanalysis. In J. Domingo-Ferrer, J. Posegga, and D. Schreckling, editors, *Smart Card Research and Advanced Applications, 7th International Conference – CARDIS 2006*, volume 3928 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2006.

[186] J. Pieprzyk. How to Construct Pseudorandom Permutations from Single Pseudorandom Functions Advances. In I. Damgård, editor, *Advances in Cryptology – EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 140–150. Springer, 1990.

[187] K. Pietrzak. A Leakage-Resilient Mode of Operation. In A. Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2009.

[188] G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In C. Walter, Ç. 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.

[189] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2007.

[190] T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 2005.

[191] E. Prouff. DPA attacks and S-Boxes. In H. Handschuh and H. Gilbert, editors, *Fast Software Encryption – FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 424–442. Springer, 2005.

[192] E. Prouff, C. Giraud, and S. Aumônier. Provably Secure S-Box Implementation Based on Fourier Transform. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2006.

[193] E. Prouff and R. P. McEvoy. First-Order Side-Channel Attacks on the Permutation Tables Countermeasure. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2009.

[194] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards. In I. Attali and T. Jensen, editors, *Smart Card Programming and Security – E-smart 2001*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[195] J.-J. Quisquater and D. Samyde. Eddy Current for Magnetic Analysis with Active Sensor. In *e-Smart 2002*, 2002.

[196] M. Renauld and F.-X. Standaert. Algebraic Side-Channel Attacks. Cryptology ePrint Archive, Report 2009/279, 2009. `http://eprint.iacr.org/`.

[197] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[198] B. Robisson and P. Manet. Differential Behavioral Analysis. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 413–426. Springer, 2007.

[199] A. Satoh, T. Sugawara, N. Homma, and T. Aoki. High-Performance Concurrent Error Detection Scheme for AES Hardware. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 100–112. Springer, 2008.

[200] P. Schaumont and K. Tiri. Masking and Dual-Rail Logic Don't Add Up. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2007.

[201] W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.

[202] J. Schmidt and C. Herbst. A Practical Fault Attack on Square and Multiply. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2008*, pages 53–58. IEEE Computer Society, 2008.

[203] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc, 2nd edition, 1996.

[204] K. Schramm and C. Paar. Higher Order Masking of the AES. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.

[205] J.-P. Seifert. On Authenticated Computing and RSA-Based Authentication. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security – CCS'05*, pages 122–127. ACM Press, 2005.

[206] N. Selmane, S. Bhasin, S. Guilley, T. Graba, and J. Danger. WDDL is protected against fault attacks. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2009*. IEEE Computer Society, 2009.

[207] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[208] A. Shamir. Improved Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attacks. Publication Number : WO9852319, Nov. 1998. Also presented at the Rump Session of Eurocrypt'97.

[209] C. E. Shannon. Communication theory of secrecy systems. *Bell System Tech. J.*, 28:656–715, 1949.

[210] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[211] S. Skorobogatov and R. Anderson. Optical Fault Induction Attack. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[212] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev. Improving the Security of Dual-Rail Circuits. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 282–297. Springer, 2004.

[213] F.-X. Standaert and C. Archambeau. Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.

[214] 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 P. J. Lee and J. H. Cheon, editors, *Information Security and Cryptology – ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2008.

[215] F.-X. Standaert, F. Koeune, and W. Schindler. How to Compare Profiled Side-Channel Attacks? In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *Applied Cryptography and Network Security – ANCS 2009*, volume 5536 of *Lecture Notes in Computer Science*, pages 485–498. Springer, 2009.

[216] F.-X. Standaert, T. Malkin, and M. 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.

[217] F.-X. Standaert, T. G. Malkin, and M. Yung. A Formal Practice-Oriented Model For The Analysis of Side-Channel Attacks. Cryptology ePrint Archive, Report 2006/139, 2006. `http://eprint.iacr.org/`.

[218] F.-X. Standaert, S. B. Örs, and B. Preneel. Power Analysis of an FPGA: Implementation of Rijndael: Is Pipelining a DPA Countermeasure? In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 2004.

[219] F.-X. Standaert, E. Peeters, C. Archambeau, and J.-J. Quisquater. Towards Security Limits of Side-Channel Attacks. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2006.

[220] F.-X. Standaert, E. Peeters, G. Rouvroy, and J.-J. Quisquater. An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays. *IEEE*, 94(2):383–394, 2006.

[221] D. Stinson. *Cryptography – Theory and Practice*. CRC Press, 1995.

[222] Sun Microsystems. Application Programming Interface – Java Card^{TM} Plateform, Version 2.2.2, Mar. 2006. `http://java.sun.com/products/javacard/specs.html`.

[223] D. Suzuki and M. Saeki. Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 255–269. Springer, 2006.

[224] D. Suzuki, M. Saeki, and T. Ichikawa. Random Switching Logic: A Countermeasure against DPA based on Transition Probability. Cryptology ePrint Archive, Report 2004/346, 2004. `http://eprint.iacr.org/`.

[225] Svetla Nikova and Christian Rechberger and Vincent Rijmen. Threshold Implementations against Side-Channel Attacks and Glitches. In P. Ning, S. Qing, and N. Li, editors, *International Conference on Information and Communications Security – ICICS'06*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[226] J. Takahashi and T. Fukunaga. Improved Differential Fault Analysis on CLEFIA. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2008*, pages 25–34. IEEE Computer Society, 2008.

[227] J. Takahashi, T. Fukunaga, and K. Yamakoshi. DFA Mechanism on the AES Key Schedule. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 62–74. IEEE Computer Society, 2007.

[228] S. Tillich and C. Herbst. Attacking State-of-the-Art Software Countermeasures-A Case Study for AES. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2008.

[229] S. Tillich, C. Herbst, and S. Mangard. Protecting AES Software Implementations on 32-Bit Processors Against Power Analysis. In J. Katz and M. Yung, editors, *ACNS*, volume 4521 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2007.

[230] K. Tiri and P. Schaumont. Changing the Odds Against Masked Logic. In E. Biham and A. Youssef, editors, *Selected Areas in Cryptography – SAC 2006*, Lecture Notes in Computer Science, pages 134–146. Springer, 2006.

[231] K. Tiri and I. Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2003.

[232] K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *Design, Automation and Test in Europe Conference and Exposition – DATE 2004*, pages 246–251. IEEE Computer Society, 2004.

[233] E. Trichina. Combinatorial Logic Design for AES SubByte Transformation on Masked Data. Cryptology ePrint Archive, Report 2003/236, 2003. `http://eprint.iacr.org/`.

[234] E. Trichina, D. DeSeta, and L. Germani. Simplified Adaptive Multiplicative Masking for AES. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2002.

[235] E. Trichina and L. Korkishko. Secure and Efficient AES Software Implementation for Smart Cards. In C. H. Lim and M. Yung, editors, *Information Security Applications – WISA 2004*, volume 3325 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2004.

[236] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 62–77. Springer, 2003.

[237] M. Tunstall and O. Benoît. Efficient Use of Random Delays in Embedded Software. In D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *Information Security Theory and Practices – WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2007.

[238] B. A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, pages 23–493, 1993.

[239] M. Ugon. Support d'information portatif muni d'un microprocesseur et d'une mémoire morte programmabale. Numéro de publication: FR2401459, Aug. 26, 1977.

[240] W. van Eck. Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? *Computer & Security*, 4:269–286, 1985.

[241] N. Veyrat-Charvillon and F.-X. Standaert. Mutual Information Analysis: How, When and Why? In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2009.

[242] D. Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.

[243] D. Wagner. Cryptanalysis of a Provable Secure CRT-RSA Algorithm. In B. Pfitzmann and P. Liu, editors, *ACM Conference on Computer and Communications Security – CCS'04*, pages 82–91. ACM Press, 2004.

[244] M. P. Wand. Data-based choice of histogram bin width. *The American Statistician*, 51:59–64, 1997.

[245] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference.* Springer Texts in Statistics, 2005.

[246] S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

[247] S.-M. Yen, D. Kim, and S. Moon. Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 53–61. Springer, 2006.

[248] S.-M. Yen, S.-J. Kim, S.-G. Lim, and S.-J. Moon. A Countermeasure against one Physical Cryptanalysis May Benefit Another Attack. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 414–427. Springer, 2001.

[249] S.-M. Yen, S.-J. Kim, S.-G. Lim, and S.-J. Moon. RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. *IEEE Transactions on Computers*, 52(4):461–472, 2003.

[250] J. F. Ziegler and W. A. Lanford. Effect of Cosmic Rays on Computer Memories. *Science*, 206(4420):776–788, 1979.

Index