On the security of embedded systems against physical attacks

Alberto Battistello

NNT: 2016SACLV047

Thèse de Doctorat
de
L'Université Paris-Saclay
préparée à
l'UVSQ - Université de Versailles-Saint-Quentin-en -Yvelines

ÉCOLE DOCTORALE nº 9 : STIC - Sciences et Technologies de l'Information et de la communication

Spécialité de doctorat "Informatique"

par

Alberto Battistello

On the security of embedded systems against physical attacks

Thèse présenté et soutenue à Paris, le June 29, 2016

Jury

M. Goubin Louis,	Directeur	Université de Versailles
M. Giraud Christophe,	Co-Encadrant	Oberthur Technologies
M. Jean-Sébastien Coron,	Rapporteur	Université du Luxembourg
M. Emmanuel Prouff,	Rapporteur	Safran Identity and Security
M. Benedikt Gierlichs,	Examinateur	Katholieke Universiteit Leuven
M. Sylvain Guilley,	Examinateur	TELECOM-ParisTech
M. David Pointcheval,	Président	École Normale Supérieure
M. Gilles Zémor,	Examinateur	Université de Bordeaux
M. Christophe Clavier,	Invité	Université de Limoges

UVSQ - Laboratoire LMV,
45 avenue des Etats-Unis
78035 Versailles, France

THESE

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Titre: Sécurité des systèmes cryptographiques embarqués vis à vis des attaques physiques

Mots clés: Embarqué, cryptographie, canaux auxiliaires, fautes

Résumé: Le sujet de cette thèse est l'analyse de sécurité des implantation cryptographiques embarquées. La sécurité a toujours été un besoin primaire pour les communications stratégiques et diplomatiques dans l'histoire. Le rôle de la cryptologie a donc été de fournir les réponses aux problèmes de sécurité, et le recours a la cryptanalyse a souvent permis de récupérer le contenu des communications des adversaires. L'arrivée des ordinateurs a causé un profond changement des paradigmes de communication et aujourd'hui le besoin de sécuriser les communication ne s'étend aux échanges commerciaux et économiques. La cryptologie moderne offre donc les solutions pour atteindre ces nouveaux objectifs de sécurité, mais ouvre la voie à des nouvelles attaques: c'est par exemple le cas des attaques par fautes et par canaux auxiliaires, qui représentent aujourd'hui les dangers plus importants pour les implantations embarquées. Cette thèse résume le travail de recherche réalisé ces trois derniers années dans le rôle d'ingénieur en sécurité au sein d'Oberthur Technologies. La plupart des résultats a été publiée sous forme d'articles de recherche [9,13–17] ou de brevets [1–6]. Les objectifs de recherche en sécurité pour les entreprises du milieu de la sécurité embarqué sont doubles. L'ingénieur en sécurité doit montrer la capacité d'évaluer correctement la sécurité des algorithmes et de mettre en avant les possibles danger futurs. Par ailleurs il est désirable de découvrir des nouvelles techniques de défense qui permettent d'obtenir un avantage sur les concurrents. C'est dans ce contexte que ce travail est présenté.

Ce manuscrit est divisé en quatre chapitre. Le premier chapitre présente une introduction aux outils mathématiques nécessaires pour comprendre la suite. Des résultats et notions fondamentaux de la théorie de l'information, de la complexité, et des probabilités sont présentés, ainsi qu'une introduction à l'architecture des micro-ordinateurs.

Le chapitre suivant présente la notion d'attaque par faute et des stratégies connues pour contrecarrer ce type d'attaques. Le corps du deuxième chapitre est ensuite dédié à notre travail sur le code infectif pour les algorithmes symétriques et asymétriques [15–17] ainsi que à notre travail sur les attaques par faute sur courbes elliptiques [13].

Le troisième chapitre est dédié aux attaques par canaux auxiliaires, et présente une introduction à certains attaques et contremesures classiques du domaine. Ensuite nos deux nouveaux attaques ciblant des contremesures considérées sécurisées sont présentés [9,14]. Dans ce troisième chapitre est enfin présenté notre nouvelle attaque combinée qui permet de casser des implémentations sécurisées à l'état de l'art.

A la fin de ce manuscrit, le quatrième chapitre présente les conclusions de notre travail, ainsi que des perspectives pour des nouveaux sujets de recherche.

Pendant nos investigations nous avons trouvé différentes contremesures qui permettent de contrecarrer certains attaques. Ces contremesures ont été publiées sous la forme de brevets [1–6]. Dans certains cas les contremesures sont présentées avec l'attaque qu'elles contrecarrent.

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Title: On the security of embedded systems against physical attacks

Keywords: Embedded, cryptography, side-channels, faults

Abstract: The subject of this thesis is the security analysis of cryptographic implementations. The need for secure communications has always been a primary need for diplomatic and strategic communications. Cryptography has always been used to answer this need and cryptanalysis have often been solicited to reveal the content of adversaries secret communications. The advent of the computer era caused a shift in the communication paradigms and nowadays the need for secure communications extends to most of commercial and economical exchanges. Modern cryptography provides solutions to achieve such new security goals but also open the way to a number of new threats. It is the case of fault and side-channel-attacks, which today represents the most dangerous threats for embedded cryptographic implementations. This thesis resumes the work of research done during the last years as a security engineer at Oberthur Technologies. Most of the results obtained have been published as research papers [9,13–17] or patents [1–6]. The security research goals of companies around the world working in the embedded domain are twofold. The security engineer has to demonstrate the ability to correctly evaluate the security of algorithms and to highlight possible threats that the product may incur during its lifetime. Furthermore it is desirable to discover new techniques that may provide advantages against competitors. It is in this context that we present our work.

This manuscript is divided into four main chapters.

The first chapter presents an introduction to various mathematical and computational aspects of cryptography and information theory. We also provide an introduction to the main aspects of the architecture of secure microcontrollers.

Afterwards the second chapter introduces the notion of fault attacks and presents some known attack and countermeasure. We then detail our work on asymmetric and symmetric infective fault countermeasures [15–17] as long as on elliptic curves fault attacks [13].

The third chapter discusses about side-channels, providing a brief introduction to the subject and to well known side-channel attacks and countermeasures. We then present two new attacks on implementations that has been considered secure against side channels [9,14]. Afterwards we discuss our combined attack which breaks a state-of-the-art secure implementation [10].

Finally, Chap. 4 concludes this works and presents some perspectives for further research.

During our investigations we have also found many countermeasures that can be used to thwart attacks. These countermeasures have been mainly published in the form of patents [1–6]. Where possible some of them are presented along with the attack they are conceived to thwart.

Acknowledgments Acknowledgments represent one of the most funny parts of writing a thesis. The main reason is that the humorous style makes the acknowledgments particularly light to read, while offering at the same time a brief aperçu of the author's character. I will thus do my best to respect this tradition.

It is incredible the number of people I have met during these years and while the margin of this page is too narrow to cite all of them, I will do my best to acknowledge those who most participated in this work. So I start with a great thank to all people which I have encountered and which in some way participated to this manuscript.

My journey into cryptography started with Prof. Ezio Stagnaro of the University of Padova, which introduced me to the pleasure of maths in its course of algebraic geometry. His courses were a pleasure as much as his digressions on fishing, prostitutes and Ramanujan. My first thoughts go to him, which instilled the passion which accompanies me everyday.

I am very grateful to Christophe Giraud, which mentored me during these last years. His experience, rigor and advice allowed me to grow both as a scientist and as an engineer. It is amazing how he helped me to develop my knowledge, and how he still have so many lessons to teach.

A special thank also goes to Prof. Louis Goubin, for accepting me as a Ph.D student and guided me during these three years. I would also like to thank all the members of the PRISM group of UVSQ for their warm welcome each time I was there.

I also thank Christophe Clavier, Jean-Sébastien Coron, Benedikt Gierlichs, Sylvain Guilley, David Pointcheval, Emmanuel Prouff and Gilles Zémor, for honoring me by taking part in the jury committee of my thesis.

My adventure in France would not have been the same without Gilles Zémor, Emmanuel Fleury and Guilhem Castagnos for their courses, for their support and for the many frank advice they provided whenever I was in need.

I would also like to mention my colleagues at Oberthur Technologies Guillaume Dabosville, Rina Zeitoun, Soline Renner, Laurie Genelle, Olivier Chamley, Thomas Chabrier, Yannick Bequer, Sarah Lopez, Nicolas Debande and in particular Philippe Andouard, Guillaume Barbu and Nicolas Morin, which helped me in so many occasions, while teaching me the secrets of physical security and sharing many funny moments and beers. I likely want to thank my other co-authors: Guénaël Renault, Jean-Sébastien Coron and Emmanuel Prouff for the invaluable discussions and lessons I learned from them.

I would like to thanks Prof. Alberto Tonolo and Prof. Alessandro Languasco for introducing me to cryptography. At the same time I'd like to thank all the people at Universitas Patavina, in particular: Luca Vidale, Marco Beggio, Enrico Carlesso, Andrea Greggio, Andrea Donazzan, Mario Collavo, Davide Soldan, and Daniela Adore for the uncountable moments of fun we had together while becoming the engineers that we are. Kudos to you!

I also send a great thank to the guys at Apple. It has been a wonderful and extremely exciting experience to work with you. Thank you very much for transmitting me the love for research which has been the starting point of the present work.

A particular thought goes to my family: Giuseppe, Romana and Francesco. You are present in all my work and your support, love and examples will always guide my life. Finally I would like to thanks Elisa for accepting to share the wonderful adventure of life with me. There is no obstacle in life that we can't overcome together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"*It is curiosity that makes me wake up in the morning.*"

Federico Fellini

Contents

1.1 Mathematical Tools

This section summarizes notions of number theory, probabilities, complexity theory, algebra and coding theory. While we do not claim to be exhaustive on these subjects, we try to provide all the necessary notions required to understand the results presented in this manuscript.

1.1.1 Algebra and Number Theory

The purpose of number theory is to study the behavior of integers. Further investigations include more general inquiries which provide further insight into the general structures of mathematics. In particular we investigate the concepts of group, ring and field, which are the basic tools required in modern cryptography.

Groups

The first and most important structure identified on the integers is that of a group. We provide in this section a glimpse of the theory of groups and on the use of group structures to investigate some problems related to cryptography.

Definition 1 (Natural numbers)**.** *The set* $0, 1, 2, 3, \ldots$ *is called the set of* natural numbers *and is denoted by the symbol* $\mathbb{N}$*.*

Definition 2 (Integers)**.** *The set* $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$ *is called the set of* integers *and is denoted by the symbol* $\mathbb{Z}$*.*

Definition 3 (Least Residue System)**.** *The subset of integers* $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ *is called the* least residue system modulo n*. It is used to identify the common choice for representing the classes of equivalences modulo the integer* n*, that is each element* $\overline{a} \in \mathbb{Z}_n$ *represents the values in the set* $\{\ldots, a-2n, a-n, a, a+n, a+2n, \ldots\}$*.*

In the following we drop the notation $\overline{a}$ in favor of the more practical a when no confusion may arise.

We furthermore use the notation $\mathbb{Z}_n^*$ which stands for $\mathbb{Z}_n \setminus \{0\}$.

Definition 4 (Group)**.** *A set* G *with the composition operation* $\circ$ *is a* group *if:*

- $\forall\ \mathbf{g}, \mathbf{h} \in G, \mathbf{g} \circ \mathbf{h} \in G$*;*
- $\exists\ \mathbf{e} \in G$ *such that* $\mathbf{e} \circ \mathbf{g} = \mathbf{g} \circ \mathbf{e} = \mathbf{g}\ \forall \mathbf{g} \in G; \mathbf{e}$ *is the* neutral element *or* identity *of* G*;*
- $\forall\ \mathbf{g} \in G\ \exists\ \mathbf{h} \in G$ *such that* $\mathbf{h} \circ \mathbf{g} = \mathbf{g} \circ \mathbf{h} = \mathbf{e}; \mathbf{h}$ *is the inverse of* $\mathbf{g} \in G$ *and is denoted by* $\mathbf{g}^{-1}$*;*
- $\forall\ \mathbf{g}, \mathbf{h}, \mathbf{j} \in G, \mathbf{g} \circ (\mathbf{h} \circ \mathbf{j}) = (\mathbf{g} \circ \mathbf{h}) \circ \mathbf{j}$*; this property is called* associativity*;*

If furthermore:

- $\mathbf{g} \circ \mathbf{h} = \mathbf{h} \circ \mathbf{g}\ \forall\ \mathbf{g}, \mathbf{h} \in G$*; this property is called* commutativity

then $(G, \circ)$ *is a* commutative *or* abelian *group.*

Typically, the group $(\mathbb{Z}_n, +)$ is an abelian group, with neutral element 0. Furthermore, $(\mathbb{Z}_n^*, \times)$ is also a group, with neutral element 1. The former is conventionally called the additive group, the latter the multiplicative group of integers modulo n. In multiplicative groups the expression $\mathbf{g}^n$ stands for $\underbrace{\mathbf{g} \circ \mathbf{g} \circ \ldots \circ \mathbf{g}}_{n \text{ times}}$, while the notation $n\mathbf{g}$ is often preferred in additive groups.

Definition 5 (Subgroup)**.** *Let $(G, \circ)$ be a group, if H is a subset of G and H is a group with respect to $\circ$, then H is a* subgroup *of G.*

Definition 6 (Cyclic Group)**.** *A group $(G, \circ)$ is* cyclic *if there exists $\mathbf{g} \in G$ such that $\forall\ \mathbf{h} \in G,\ \exists n \in \mathbb{N}$ such that $\mathbf{h} = \mathbf{g}^n$. Such an element $\mathbf{g}$ is called* generator *of the cyclic group.*

The notation $\langle \mathbf{g} \rangle$ is commonly used to denote the set of elements generated by $\mathbf{g}$, i.e.: $\langle \mathbf{g} \rangle = \{\mathbf{g}^0, \mathbf{g}, \mathbf{g}^2, \ldots, \mathbf{g}^{n-1}\}$.

Definition 7 (Order of an element)**.** *The* order *of an element $\mathbf{g} \in G$ is the smallest positive integer n (if it exists) such that $\mathbf{g}^n = \mathbf{e}$, and it is denoted $o(\mathbf{g})$.*

Definition 8 (Finite Group)**.** *A group is said to be* finite *if it has a finite number of elements.*

Theorem 1 (Gauss)**.** *It can be proved that for $n = 1, 2, 4, p^\alpha, 2p^\alpha$, where p is a prime number and $\alpha > 1$, the multiplicative group $\mathbb{Z}_n^*$ is cyclic. In particular it can be proved that if p is prime, then $\mathbb{Z}_p^*$ is a finite multiplicative cyclic group.*

Proof. This is a famous result of Gauss. Due to the length of the proof we refer the reader to [84]. □

Theorem 2 (Lagrange)**.** *In a finite group $(G, \circ)$ each element $\mathbf{g} \in G$ satisfies $o(\mathbf{g})$ divises $|G|$.*

Proof. Let $\mathbf{g} \in G$ and $o(g) = d$. If $d = |G|$ then we have nothing to demonstrate. Otherwise, if $d < |G|$ then there exists $\mathbf{h_1} \in G \setminus \langle \mathbf{g} \rangle$. Let us consider the set $H_1 = \{\mathbf{h}_1 \cdot \mathbf{g}^j : j = 0, \ldots, n-1\}$, it can be observed that $|H_1| = d$ and that $H_1 \bigcap \langle \mathbf{g} \rangle = \varnothing$. If $G = H_1 \cup \langle \mathbf{g} \rangle$ then $|G| = 2d$. Otherwise consider the subset H_2 generated by $\mathbf{h_2} \in G \setminus (\langle \mathbf{g} \rangle \cap H_1)$. H_2 has d distinct elements and is disjoint from $\langle \mathbf{g} \rangle \cup H_1$. Repeat the process until all elements of G have been consumed. The process terminates because G is finite. □

Corollary 1. *Let $(G, \circ)$ be a finite group, then for each element $\mathbf{g} \in G$, $\mathbf{g}^{|G|} = e$.*

Proof. Let $d = o(\mathbf{g})$ then $|G| \mid d \in \mathbb{N}$, so we can write $\mathbf{g}^{|G|} = (\mathbf{g}^d)^{|G||d} = e$. □

A particular case of the above result is the following:

Theorem 3 (Fermat)**.** *If p is prime and $p \nmid a$ then $a^{p-1} \equiv 1 \bmod p$.*

Proof. The theorem is a consequence of 1. □

Definition 9 (Euler totient function)**.** *The set of elements of $\mathbb{Z}_n$ having a multiplicative inverse is denoted $\mathbb{Z}_n^*$ and its cardinality is denoted $\varphi(n)$, called the* Euler totient function.

We recall here some interesting result on groups and in particular on integers that are of fundamental importance for modern cryptography.

Definition 10 (multiples)**.** *The set of* multiples *of an integer $d \in \mathbb{Z}$ is denoted $d\mathbb{Z} = \{ad : a \in \mathbb{Z}\}$.*

Definition 11 (greatest common divisor)**.** *Given two or more integers $n_1, \ldots, n_r \in \mathbb{Z}$ not all zero, the* greatest common divisor *of $n_1, \ldots, n_r$ ($gcd(n_1, \ldots, n_r)$), is the largest positive integer that divides the numbers without a remainder.*

Theorem 4 (Euclid)**.** *Given two integers $n, m \in \mathbb{Z}$, let $\mathcal{A}(n, m) = \{an + bm : a, b \in \mathbb{Z}\}$ and $d = gcd(n, m)$. Then $\mathcal{A}(n, m) = d\mathbb{Z}$ so that there exists $\lambda, \mu \in \mathbb{Z}$ such that $\lambda n + \mu m = d$.*

Proof. We want to prove that $\mathcal{A}(n, m) = d\mathbb{Z}$. It is easy to notice that $d \mid \lambda n + \mu m$ for all λ, μ. Then assume $\delta = \lambda n + \mu m$ is the smallest positive element in $\mathcal{A}(n, m)$, it exists as long as m or n is not null. We want to show that $\delta \mid d$. Let us take the remainder r of the division of n by δ. Let $n = \delta q + r$ then $r \in \mathcal{A}(n, m)$ because $r = (1 - \lambda q)n - \mu q m$ by substituting the expression for δ. Moreover r is smaller than δ by division, thus it is 0 because δ is by definition the smallest element of $\mathcal{A}(n, m)$. Equivalently $\delta \mid m$. Thus $\delta \mid d$ so $\delta = d$. □

Euclid proved that it is possible to write the gcd between two integers as a linear combination of the two, and provided an algorithm to compute the integer coefficients λ, μ of the linear combination. The interest of this algorithm is that it allows to compute the multiplicative inverse of an element of the field $\mathbb{Z}_n^*$.

A great theorem of unknown author is the famous Chinese Remainder Theorem, dating back at least to the 3rd century. In order to understand it we need the following results.

Definition 12 (group homomorphism)**.** *Let $(G, +), (R, \circ)$ be two groups. A map $f : G \to R$ is called a* group homomorphism *if $f(x + y) = f(x) \circ f(y)$, for every $x, y \in G$.*

Definition 13. *The* kernel *of a group homomorphism $f : G \to R$ is*

$$Kerf = \{g \in G : f(g) = e\}$$

where e is the neutral element of R. A bijective group homomorphism is called a group isomorphism*.*

In the following we use the notation e_R to explicitly denote that $e \in R$ when necessary.

Proposition 1. *Given a group homomorphism $f : G \to R$, if $Kerf = \{e_R\}$ then f is injective ($\forall\, x, y\, x \neq y \Rightarrow f(x) \neq f(y)$).*

Proof. Since $f(e_G) = e_R$ it follows that $Kerf = \{e_R\}$ if f is injective. Conversely assume $Kerf = \{e_R\}$. Assume by contradiction that there exists $x, y \in G$ such that $x \neq y$ but $f(x) = f(y)$. Let $x = y + a$ for some $a \in G \setminus \{e_R\}$. Then by homomorphism we have $f(y) = f(x) \circ f(a)$ which implies $a \in Kerf$, a contradiction. □

Theorem 5 (Chinese Remainder Theorem (CRT)). *Let $n_1, n_2, \ldots, n_r \in \mathbb{Z} \setminus \{0\}$, and $gcd(n_i, n_j) = 1 \ \forall \ i \neq j$, then the solution to the system of modular equations*

$$\begin{cases} x \equiv a_1 \bmod n_1 \\ x \equiv a_2 \bmod n_2 \\ \ldots \\ x \equiv a_r \bmod n_r \end{cases}$$

is unique modulo $n_1 \cdot n_2 \cdot \ldots \cdot n_r$.

Proof. In fact we can prove, more generally, that

$$f : \mathbb{Z}_N \to \mathbb{Z}_{n_1} \times, \ldots, \times \mathbb{Z}_{n_r}$$

given by $f(x) = (x \bmod n_1, \ldots, x \bmod n_r)$ is an isomorphism. It is easy to see that f is a homomorphism of groups. Now let us prove that $Ker f = \{e\}$. If $n \in Ker f$ then $n = 0 \bmod n_1, \ldots, n = 0 \bmod n_r$. This means that $n_1 \mid n, \ldots, n_r \mid n$, so $n_1 \cdots n_r \mid n$, so $n \in N\mathbb{Z}$. This proves that $Ker f \subseteq N\mathbb{Z}$. Now let $n \in N\mathbb{Z}$. Then we can write $n = k \cdot n_1 \cdots n_r$ for some $k \in \mathbb{Z}$. So we obtain $n_1 \mid n, \ldots, n_r \mid n$. This proves that $N\mathbb{Z} \subseteq Ker f$. So $Ker f = \{e\}$, thus f is injective. Now we can observe that the cardinality of $\mathbb{Z}_N$ and $\mathbb{Z}_{n_1} \times, \ldots, \times \mathbb{Z}_{n_r}$ are the same, thus from the pigeonhole principle we can conclude that f is also surjective, thus an isomorphism. □

Despite the author of the CRT being still unknown a good deal of its history can be found in [102].

Finite Fields

Further investigation on numbers allow definitions that are more and more precise, and which collect objects sharing even more interesting properties. This is the case of rings and fields, which are the subject of this section.

Definition 14 (Commutative Ring). *A set R together with the operations $+$ and $\times$ is a commutative* ring *with identity if $(R, +)$ is an abelian group with neutral element 0, and the neutral element of $\times$ is $1 \neq 0$, it is associative and commutative, and the* distributive *property is satisfied:*

- $\forall \ \mathbf{x}, \mathbf{y}, \mathbf{z} \in R, (x + y) \times z = (x \times z) + (y \times z)$.

In this work we will omit the symbol $\times$ to indicate $a \times b$ when no confusion arise.

Definition 15 (unit). *An element x of a ring R is called a* unit *if there exists $y \in R$ such that $xy = yx = 1$. In this case y is denoted x^{-1} and called the inverse of x. The set of units in R is denoted R^*.*

Definition 16 (Irreducible elements). *An element r of a ring $R \backslash R^*$ is called* irreducible *if $r = ab$ for $a, b \in R$ implies that either a or b is a unit.*

Thus if r is irreducible and u is a unit, then ur is also an irreducible element.

Definition 17 (factorization into irreducible elements). *A non-zero element r of a ring $R \setminus R^*$ is said to have a* factorization into irreducible elements *if there exists irreducible elements $p_1, p_2, \ldots, p_r \in R$ such that*

$$r = p_1 \cdot p_2 \cdots p_r \ .$$

Definition 18 (Prime element)**.** *A non-zero element p of a ring $R \setminus R^*$ is called a* prime element *if $p \mid xy$ for $x, y \in R$ implies that $p \mid x$ or $p \mid y$.*

Definition 19. *A prime element is irreducible.*

Proof. Let p be a prime element. Suppose that $p = ab$. By definition of a prime element we can conclude that $p \mid a$ or $p \mid b$. Suppose that $p \mid a$. Then we can write $a = rp$ for some $r \in R$. This implies that $p = rpb$. Now $rb = 1$ implies that b is a unit. Thus p is irreducible. □

Definition 20 (Characteristic)**.** *Let R be a ring. The* characteristic *of a ring is the smallest positive integer n (if it exists), such that $\underbrace{1 + 1 + \ldots + 1}_{n \text{ times}} = 0 \in R$. If n does not exists, then R is said to have characteristic 0.*

Definition 21 (Field)**.** *A commutative ring R with identity is a* field *if $R \setminus \{0\}$ is a group with respect to the $\times$ operation. In this case we denote the field $\mathbb{R}$ in order to emphasize the field notation.*

In other words a field $\mathbb{K}$ is a group with respect to both "addition" and "multiplication". Furthermore each element different from 0 must have a multiplicative inverse. A field is said to be *finite* if it has a finite number of elements. We denote by $\mathbb{F}_q$ a finite field with q elements.

Definition 22 (Polynomial ring)**.** *The* polynomial ring, *$\mathbb{K}[X]$, in X over the ring K is defined as the set of expressions, called polynomials, in X, of the form:*

$$P(X) = a_0X + a_1X^2 + \ldots + a_{n-1}X^{n-1}$$

where $a_0, a_1, \ldots, a_{n-1}$, the coefficients of $P(X)$, are elements of K, and $X, X^2, \ldots$, are formal symbols ("the powers of X"). By convention $X^0 = 1$ and $X^1 = X$, and the product of powers of X is defined by the formula $X^lX^k = X^{l+k}$. The symbol X is called a variable or indeterminate.

Definition 23 (Irreducible polynomials)**.** *A polynomial $a_0X + a_1X^2 + \ldots + a_{n-1}X^{n-1}$ with coefficients $a_0, a_1, \ldots, a_{n-1}$ (over $\mathbb{K}$) is* irreducible, *over $\mathbb{K}$, if it cannot be factored into the product of two non-constant polynomials with coefficients in $\mathbb{K}$.*

Theorem 1 shows that it is easy to build fields with p elements just by taking $\mathbb{Z}_p$, where p is a prime number. However, it is much more difficult to build finite fields of non prime orders.

The canonical way to obtain a field $\mathbb{F}_q$ of cardinality $q = p^\alpha$ is called *algebraic closure*. It consists in generating the smallest field $\mathbb{K}$ that contains $\mathbb{F}_p$ and the roots of an irreducible polynomial of degree α with coefficients in $\mathbb{F}_p$. A finite field of $q = p^n$ elements is often denoted $GF(p^n)$, where GF stands for "Galois Field", in honor of the famous French mathematician.

Example 1. *A classical example is the construction of the complex field $\mathbb{C}$ with the polynomials over the real numbers $\mathbb{R}[X]$ and the polynomial $x^2 + 1$, irreducible over $\mathbb{R}$. By reducing each polynomial of $\mathbb{R}[X]$ modulo $x^2 + 1$, one obtain elements of the form $a + \overline{X}b$, where a, b are real coefficients and $\overline{X}^2 = -1$, due to the modular reduction. The set obtained is a field which is isomorphic, by changing the variable $\overline{X}$ with i, to the complex field $\mathbb{C}$, that is $\{a + ib : a, b \in \mathbb{R}, i^2 = -1 \in \mathbb{R}\}$.*

Elliptic Curves

Among the most interesting objects discovered in number theory are *Elliptic Curves*. Their study allowed to prove one of the most interesting theorem of Fermat [174], and they are at the basis of many modern cryptosystems. We will however restrain our investigation to some basic definition and properties that are sufficient to understand the remainder of this work. However, for further information we suggest [122] and [99].

Let $\mathbb{K}$ be a field with $\text{Char}(\mathbb{K}) \neq 2, 3$, and $a, b \in \mathbb{K}$. The short Weierstraß form of an elliptic curve is defined by the following equation:

$$\mathcal{E} : y^2 = x^3 + ax + b \tag{1.1}$$

The set of points $(x, y) \in \mathbb{K} \times \mathbb{K}$ satisfying (1.1), together with the point at infinity $\mathcal{O}$ is denoted $\mathcal{E}(\mathbb{K})$. For any point $P = (x_P, y_P) \in \mathcal{E}(\mathbb{K})$, the opposite is defined as $-P = (x_P, -y_P)$, and $P + \mathcal{O} = \mathcal{O} + P = P$. For any two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q) \in \mathcal{E}(\mathbb{K})$, the sum $R = (x_R, y_R)$ of $P + Q$ is defined as:

- If $P \neq -Q$ then:

$$\begin{cases} x_R = s^2 - x_P - x_Q \\ y_R = s(x_P - x_R) - y_P \end{cases}$$

  with

$$s = \begin{cases} \frac{(y_Q - y_P)}{(x_Q - x_P)} & \text{if } P \neq Q \\ \frac{(3x_P^2 + a)}{(2y_P)} & \text{otherwise} \end{cases}$$

- If $P = -Q$ then $P + Q = P - P = \mathcal{O}$

The set $\mathcal{E}(\mathbb{K})$ with the "+" operation defined above forms an abelian group with neutral element $\mathcal{O}$. For the point operations to be well defined it is important to avoid *non singular* curves. This condition translates into verifying that both partial derivatives does not annihilates. By applying this condition to the curve equation one obtains:

$$\begin{cases} \frac{\delta \mathcal{E}}{\delta x} = x^2 + a = 0 \\ \frac{\delta \mathcal{E}}{\delta y} = 2y = 0 \end{cases}$$

Solving the system provides the condition for *non singular* curves:

$$27b^2 + 4a^3 = 0 \tag{1.2}$$

We denote by q the cardinality (or order) of $\mathcal{E}(\mathbb{K})$. For common curves it is often the product of a big prime times a small cofactor. In the following we will drop the explicit field notation when it is implicit or when it is not necessary for the given statement.

Discrete Logarithm

The discrete logarithm problem in a finite group $(G, \cdot)$ can be stated as follows: compute x from $\mathbf{g}$ and $\mathbf{y} = x \cdot \mathbf{g}$. The integer x is called the discrete logarithm of $\mathbf{y}$ in base $\mathbf{g}$, $x = \log_{\mathbf{g}}(\mathbf{y})$. We will restrict our attention to the case where G is a finite cyclic group of prime order q and $\mathbf{g}$ is a generator of G. In the above, we use bold letters to denote elements of G, so that no confusion arises with scalars, such as x. We write the group operation in additive notation: $x \cdot \mathbf{g}$ is simply $x \cdot \mathbf{g} = \mathbf{g} + \cdots + \mathbf{g}$, where $\mathbf{g}$ is repeated x times. Examples of cryptographic interest are the subgroup of order q of $(\mathbb{Z}_p, \times)$ where q is a large prime factor of $p - 1$, and subgroups of prime order of an elliptic curve.

1.1.2 Probabilities

We recall in this section some notations and results of probability theory on discrete random variables that are used in the remainder of this work. For an introduction to probability theory the user can refer to [150].

Definition 24. *An* experiment *is a procedure that yields one of a given set of outcomes. The individual possible outcomes are called* simple events. *The set of all possible outcomes is called the* sample space.

We will only considers discrete sample spaces; that is, sample spaces with only finitely many possible outcomes. Let the simple events of a sample space S be labeled $s_1, s_2, \ldots, s_n$. Because the value of a random variable is determined by the outcome of the experiment, we may assign probabilities to the possible values of the random variable.

Definition 25. *A probability distribution P on S is a sequence of numbers $p_1, p_2, \ldots, p_n$ that are all non-negative and sum to 1. The number p_i is interpreted as the probability of s_i being the outcome of the experiment.*

Definition 26. *An event E is a subset of the sample space S. The probability that event E occurs, denoted $P(E)$, is the sum of the probabilities p_i of all simple events s_i which belong to E. If $s_i \in S, P(\{s_i\})$ is simply denoted by $P(s_i)$.*

Definition 27 (random variable)**.** *A* random variable *is a measurable function from the set of possible outcomes to some event E.*

A sequence of random variables is said to be independent and identically distributed (i.i.d), if each random variable has the same probability distribution as the others and all are mutually independent.

Definition 28 (probability mass function)**.** *For a discrete random variable X, we define the probability mass function $p(\cdot)$ of X by*

$$p(a) = P(\{X = a\}) .$$

Definition 29 (Expectation)**.** *If X is a discrete random variable having a probability mass function $p(\cdot)$, then the expectation, or the expected value, of X, denoted by $E[X]$, is defined by*

$$E[X] = \sum xp(x) .$$

Definition 30 (variance)**.** *Given a random variable X with expectation μ, then the variance of X, denoted by $Var(X)$, is defined by*

$$Var(X) = E[(X - \mu)^2 .]$$

It is often useful to be able to get some prediction on the results of experiments involving two random variables. This is the subject of conditional probability.

Conditional Probabilities

Definition 31. *Let E_1 and E_2 be two events with $P(E_2) > 0$. The* conditional probability *of E_1 given E_2, denoted $P(E_1 \mid E_2)$, is*

$$P(E_1 \mid E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)} \, .$$

$P(E_1 \mid E_2)$ measures the probability of event E_1 occurring, given that E_2 has occurred.

Definition 32. *Events E_1 and E_2 are said to be* independent *or* mutually independent *if $P(E_1 \mid E_2) = P(E_1)$.*

Observe that if E_1 and E_2 are independent, then $P(E_1 \mid E_2) = P(E_1)$ and $P(E_2 \mid E_1) = P(E_2)$. That is, the occurrence of one event does not influence the likelihood of occurrence of the other one.

Theorem 6 (Bayes)**.** *If E_1 and E_2 are events with $P(E_2) > 0$, then*

$$P(E_1 \mid E_2) = P(E_1)\frac{P(E_2 \mid E_1)}{P(E_2)} \, .$$

Theorem 7 (Law of total probability)**.** *Let $\{B_n : n = 1, 2, 3, \ldots\}$ is a finite or countably infinite partition of the sample space and each event B_n is measurable, then for any event $\boldsymbol{A}$ of the same probability space:*

$$P(A) = \sum_n P(A \cap B_n \, .$$

Birthday paradox

Definition 33. *The* birthday paradox *analyzes the probabilities that in a set of randomly chosen people, some pair of them will have the same birthday.*

The birthday paradox can be generalized to the following problem.

Proposition 2. *Let E be a finite set. The probability $P(n)$ of extracting at least twice the same element in n uniform extractions is*

$$P(n) = 1 - \frac{|E|!}{(|E| - n)!} \cdot \frac{1}{|E|^n} \, .$$

The probability $P(n)$ can be approximated by using Taylor series to

$$P(n) = 1 - e^{-\frac{n \cdot (n-1)}{2 \cdot |E|}} \, .$$

The name *birthday paradox* comes from the fact that for birthdays (assuming non leap years) $P(n)$ is greater than 0.5 as soon as $n > 23$. Thus as soon as 23 people are interviewed, chances are that two of them share the same birthday, which seems counterintuitive at first glance.

Birthday Paradox in Cryptography

While the birthday paradox seems a mathematical curiosity, it is of fundamental importance in cryptography. Its generalization can be (very) roughly approximated to the fact that as soon as $n = \sqrt{|E|}$ elements are uniformly extracted from a set of size $|E|$, then it is probable that a collision has been found. This provides a practical lower bound for the number of queries required to find a collision on the outputs of a given function, for example, on hash functions.

1.1.3 Complexity Theory

The theory of complexity dates back to the work of Alan Turing [166, 167] at the end of 30's. Its main objective is to provide mathematical models to classify computational problems according to the resources needed to solve them. The classification should capture the intrinsic difficulty of the problem, measuring resources such as time and storage space.

Definition 34 (Algorithm)**.** *An algorithm is a well-defined ordered sequence of elementary steps that takes a variable input and halts with an output.*

This definition is rather informal, as the concept of elementary steps is not well defined. While a more formal definition can be achieved by introducing objects like *Turing machines*, we do not need such deep formalism, and it is easier to think of algorithms as a computer program written in some language which takes some variable input and halts with an output after some time.

Definition 35 (input size)**.** *The* size *of the input of an algorithm is the minimum number of bits required to represent an input of the algorithm.*

Definition 36 (running time)**.** *The* running time *of an algorithm is the number of elementary steps to be executed on a particular input.*

The lack of a specific definition of "elementary step" comes in hand on particular algorithms, where it may be substituted with "binary operation" or "machine instruction" or "modular multiplication", etc., depending on what is more appropriate for the context.

Definition 37 (worst-case)**.** *The* worst-case *running time of an algorithm is an upper bound on the running time of the algorithm for any input, expressed as a function of the input size.*

Definition 38 (average-case)**.** *The* average-case *running time of an algorithm is the average running time of the algorithm on uniformly distributed inputs.*

Asymptotic Notation

The *big O* notation describes the asymptotic behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions.

Definition 39 (Big O)**.** *Let f and g be two functions defined on some subset of the real numbers. One writes $f(x) = O(g(x))$ if and only if there exists a positive real number c and a real number x_0 such that for any $x > x_0$ we have:*

$$|f(x)| \leq c|g(x)|\,.$$

Example 2. *Simple examples of the Big O notation allows to define the complexity of some naive algorithm with respect to the number of binary operations involved. In the following assume $n, m \in \mathbb{Z}$ and $n > m$ such that $log(n)$ approximates the bits required to write n or m. Then:*

- *the complexity of $m + n$ is $O(log(m) + log(n))$, which corresponds to $O(log(n))$*
- *the complexity of $m \cdot n$ is $O(log(m) \cdot log(n))$, equivalently $O(log^2(n))$,*
- *the complexity of the extended Euclid's algorithm(Theorem 4) on input n, m is $O(log^2(n))$ (see [108]),*

Definition 40 (polynomial complexity)**.** *An algorithm is said to have* polynomial complexity *if its worst-case running time on input of size* $\log(n)$ *bit is a function of the form $O(log^k(n))$, for a constant k.*

Definition 41 (exponential complexity)**.** *An algorithm is said to have* exponential complexity *if its worst-case running time on input of size* $\log(n)$ *bit is a function of the form $O(n^k) = O(exp(klog(n))$, for a constant k.*

Definition 42 (sub-exponential complexity)**.** *An algorithm is said to have* sub-exponential complexity *if its worst-case running time on input of size* $\log(n)$ *bit is a function of the form $O(exp(\epsilon log(n))$, for any $\epsilon > 0$.*

1.1.4 Lattices

We review in this section some results on lattices. These results are mostly taken from [125, 126, 134].

Definition 43 (vector space)**.** *A* vector space $\mathbb{V}$ *over a field $\mathbb{K}$ is a set with two operations:*

- *an internal composition law denoted "+" called* addition*,*
- *an external left composition law denoted "$\cdot$", called* scalar multiplication*.*

The elements of $\mathbb{K}$ are called scalars*, those of $\mathbb{V}$ are called* vectors *and are denoted with bold letters. It is required that $(V, +)$ is an abelian group. This means that there exists a neutral element denoted $\mathbf{0}$, called* null vector*, and that each vector $\mathbf{v}$ has an opposite denoted $-\mathbf{v}$. Furthermore the following properties must be satisfied for the '$\cdot$' law:*

- *distributivity on the left with respect to the $+$ and on the right with respect to the field addition:*

$$\begin{aligned} a \cdot (\mathbf{v} + \mathbf{u}) &= a \cdot \mathbf{v} + a \cdot \mathbf{u} \ , \quad \forall\ a \in \mathbb{K},\ \forall\ \mathbf{v}, \mathbf{u} \in \mathbb{V} \\ (a + b) \cdot \mathbf{v} &= a \cdot \mathbf{v} + b \cdot \mathbf{v} \ , \quad \forall\ a, b \in \mathbb{K},\ \forall\ \mathbf{v} \in \mathbb{V} \end{aligned}$$

- *pseudo-associativity with respect to the multiplication in $\mathbb{K}$,*

$$(ab) \cdot \mathbf{v} = a \cdot (b \cdot \mathbf{v}) \ , \quad \forall\ a, b \in \mathbb{K},\ \forall\ \mathbf{v} \in \mathbb{V}$$

- *the multiplicative neutral element of $\mathbb{K}$, denoted 1 is neutral on the left for $\cdot$.*

$$1 \cdot \mathbf{v} = \mathbf{v} \ , \quad \forall\ \mathbf{v} \in \mathbb{V}$$

Definition 44. *Let $S = \{v_1, v_2, \ldots, v_n\}$ be a finite subset of a vector space $\mathbb{V}$ over a field $\mathbb{K}$.*

1. *A linear combination of S is an expression of the form $a_1v_1 + a_2v_2 + \cdots + a_nv_n$, where each $a_i \in \mathbb{K}$.*

2. *The set S is linearly dependent over $\mathbb{K}$ if there exist scalars $a_1, a_2, \ldots, a_n$, not all zero, such that $a_1v_1 + a_2v_2 + \cdots + a_nv_n = 0$. If no such scalars exist, then S is linearly independent over $\mathbb{K}$.*

3. *A linearly independent set of vectors that spans $\mathbb{V}$ is called a basis for $\mathbb{V}$.*

Definition 45 (euclidean norm)**.** *On a n-dimensional space $\mathbb{R}^n$, the intuitive notion of length of the vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ is captured by the formula:*

$$\|v\| := \sqrt{v_0^2 + v_1^2 + \ldots + v_{n-1}^2}$$

and $\|v\|$ is called the euclidean norm *of the vector $\mathbf{v}$.*

Definition 46 (lattice)**.** *A* lattice *is a discrete (additive) subgroup of $\mathbb{R}^n$. In particular an* integer lattice *is a subgroup of $\mathbb{Z}^n$.*

Alternatively, a lattice $\mathbf{L}$ may also be defined as:

$$\mathbf{L} = \left\{ \sum_{i=1}^{d} a_i \mathbf{v}_i : a_i \in \mathbb{Z} \right\} .$$

where d is the *lattice dimension* and the set of $\mathbf{v}_i$ is a basis of $\mathbf{L}$. All bases of a lattice have the same dimension denoted $dim(L)$. Since a lattice is discrete by definition, it has a shortest non-zero vector: the euclidean norm of this vector is called the *first minimum*, and is denoted by $\lambda(\mathbf{L})$.

Definition 47 (volume)**.** *Let $\mathbf{L}$ be a lattice, the* volume *$vol(\mathbf{L})$ of the lattice corresponds to the d-dimensional volume of the parallelepiped spanned by the $\mathbf{b}_i$'s.*

In the important case of full-dimensional lattices where $dim(\mathbf{L}) = n$, the volume is equal to the absolute value of the determinant of any lattice basis.

Definition 48 (Minkowski's i-th minimum)**.** *Given a lattice $\mathbf{L}$, the* Minkowski's i-th minimum *denoted $\lambda_i(\mathbf{L})$ is defined as the minimum of $max_{1 \leq j \leq i} \|\mathbf{v}_j\|$ over all linearly independent lattice vectors $\mathbf{v}_0, \ldots, \mathbf{v}_{d-1}$.*

Algorithmic Problems

Definition 49 (Shortest Vector Problem (SVP))**.** *Given a basis of a lattice $\mathbf{L}$, find $\mathbf{u}$ such that $\|\mathbf{u}\| = \lambda(\mathbf{L})$.*

A relaxed SVP is the approximated SVP, where it is asked a non-zero vector $\mathbf{v} \in \mathbf{L}$ with norm bounded by some approximation factor: $\|\mathbf{u}\| = f(d)\lambda(\mathbf{L})$.

Definition 50 (Closest Vector Problem (CVP))**.** *Given a basis of a lattice $\mathbf{L} \subseteq \mathbb{R}^n$ and a vector $\mathbf{v} \in \mathbb{R}^n$ find $\mathbf{u}$ such that the distance between $\mathbf{v}$ and $\mathbf{u}$ is minimal, i.e. $\|\mathbf{v} - \mathbf{u}\| \leq \|\mathbf{v} - \mathbf{w}\|$,$\forall\ \mathbf{w} \in \mathbf{L}$.*

Again there exists an approximated problem where it is only required to find a vector whose distance to $\mathbf{v}$ is bounded by some approximation factor.

No polynomial time algorithm is known for approximating either SVP or CVP to within a polynomial factor in the dimension d. However, there exists polynomial time algorithms that achieve sub exponential factors solutions, all of these algorithms are based on the *LLL* algorithm [115]. It has been remarked that such algorithms typically perform much better than it is theoretically guaranteed.

Theorem 8 (Lenstra Lenstra Lovaz (LLL)). *Given as input a basis of a lattice $\mathcal{L}$, LL provably outputs in polynomial time a basis $(b_0, b_1, \ldots, b_{d-1})$ satisfying:*

$$\|b_1\| \leq 2^{(d-1)/4} vol(\mathbf{L})^{1/d} \ , \|b_i\| \leq 2^{(d-1)/2} \lambda_i(\mathbf{L}) \quad and \quad \prod_{i=1}^{d} \|\mathbf{b}_i\| \leq 2^{\binom{d}{2}/2} vol(\mathbf{L}).$$

Proof. For the proof of this result we refer the reader to [115]. □

The above theorem thus shows that it is possible to approximate the SVP within a factor $2^{d-1}/2$. Furthermore, Shnorr [154] improved this result by showing that it is possible to achieve a bound of $2^{O(d(loglogd)^2/logd)}$ and Ajtai *et al.* [1] improved it to $2^{O((dloglogd)/logd)}$. The LLL algorithm finds many useful applications, in this work we focus on one of them, which allows to find small roots of univariate and multivariate modular equations. The main result on the subject is due to Coppersmith [54, 55] who showed how to find roots of modular polynomials in polynomial time.

Theorem 9 (Coppersmith). *Let $f(x)$ be a monic polynomial of degree δ in one variable modulo an integer n of unknown factorization. One can find all integers x_0 with $f(x_0) \equiv 0 \bmod n$ and $\|x_0\| < n^{\frac{1}{\delta}}$ in time polynomial in $\log n$ and δ.*

The core idea consists in reducing the problem to solving univariate polynomial equations over the integers, by transforming modular roots into integral roots. More precisely, it constructs a polynomial $P(X) \in \mathbb{Z}[X]$ whose roots are all the $x_0 \in \mathbb{Z}$ that verifies $f(x_0) = 0 \bmod N$ and $\|x_0\| < N^{\frac{1}{\delta}}$. $P(X)$ is constructed such that it can easily be solved over $\mathbb{Z}$. To do so, several multiples of f are considered, all admitting x_0 as a root modulo N^m for a given parameter m. The lattice spanned by these polynomials thus contains a polynomial V which admits x_0 as root modulo N^m and which has small coefficients. If x_0 is small enough then its solution will hold over the integers and thus can easily be solved. The LLL algorithm thus allows to "extract" such polynomial from the lattice as the smallest (up to a polynomial factor) vector. Despite lattices being an exciting and developing field, they only play a small role in this work. We thus refer the interested reader to the work of Nguyen *et al.* [133, 134] for further investigations.

1.1.5 Coding Theory

The following section provides basic definitions and notations sufficient to understand the masking scheme discussed in Sec. 3.3.

Definition 51 (Linear Code). *A linear code $\mathcal{C}$ of length n and dimension k is a subspace of dimension k of the vector space $\mathbb{F}_2^n$.*

A word of the code $\mathcal{C}$ is then a vector w such that $w \in \mathcal{C}$.

Definition 52 (Supplement of vector space)**.** *The supplement of the vector space $\mathcal{C}$ in $\mathbb{F}_2^n$ is the set of vectors $\mathcal{D}$ such that $\mathcal{C} \oplus \mathcal{D} = \mathbb{F}_2^n$, where $\oplus$ denotes the direct sum of two vector spaces.*

Then an element z in $\mathbb{F}_2^n$ can be decomposed uniquely as the sum of two elements c and d, respectively in $\mathcal{C}$ and $\mathcal{D}$:

$$z = c \oplus d \tag{1.3}$$

Definition 53 (Generating matrix)**.** *The vectors of the basis of a linear code $\mathcal{C}$ forms the generating matrix of $\mathcal{C}$.*

In the following we will denote respectively by G and H the generating matrices of $\mathcal{C}$ and $\mathcal{D}$. Then every element of $\mathcal{C}$ (resp. $\mathcal{D}$) can be written uniquely as xG (resp. yH) for some x (resp. y) in $\mathbb{F}_2^k$ (resp. $\mathbb{F}_2^{k'}$):

$$z = xG \oplus yH \tag{1.4}$$

Definition 54 (Dual code)**.** *The dual code of $\mathcal{C}$ is the linear code $\mathcal{C}^\perp = \{w \in \mathbb{F}_2^n | \forall c \in \mathcal{C},\ c \cdot w = 0\}$. This notion is similar to that of the orthogonal of a vector space.*

In the case where $\mathcal{C}^\perp = \mathcal{D}$, it comes straightforwardly that the dimension of $\mathcal{D}$ is $n - k$ if the dimension of $\mathcal{C}$ is k.

Finally, we recall a necessary and sufficient condition to have $\mathcal{C}$ and $\mathcal{C}^\perp$ supplementary in $\mathbb{F}_2^n$:

Proposition 3. *Without loss of generality (a permutation of coordinates might be necessary), we can assume that the generating matrix of $\mathcal{C}$ is systematic, and thus takes the form $[I_k \| M]$, where I_k is the $k \times k$ identity matrix. The supplementary $\mathcal{D}$ of $\mathcal{C}$ is equal to $\mathcal{C}^\perp$ iff the matrix $I_k \oplus MM^T$ is invertible.*

1.2 Cryptosystems

In this section we recall basic notions on information theory, introduced by Shannon at the end of the 40's [159, 160]. In particular we focus on information security, in order to provide the concepts of attacker and basic concepts of cryptography. For further references we suggest [122, 164]. In this section we also describe the cryptosystems which are of interest for the understanding of this work. While many more interesting algorithms exist, space constraints do not allow us to include them all.

1.2.1 Cryptographic Schemes

The theory of information provides models of the reality in order to capture the relevant aspects of information exchanges. We thus start with some notation which describe our model.

Definition 55. *A finite set $\mathcal{A}$ is called the* alphabet *of definition. For example, $\mathcal{A} = \{0, 1\}$, the binary alphabet, is a frequently used alphabet of definition. Note that any alphabet can be encoded in terms of the binary alphabet. For example, since there are 32 binary strings of length five, each letter of the English alphabet can be assigned a unique binary string of length five.*

Notation. *A set $\mathcal{M}$ is called the* message space*. $\mathcal{M}$ consists of strings of symbols from an alphabet of definition. An element of $\mathcal{M}$ is called a* plaintext*. For example, $\mathcal{M}$ may consist of binary strings, English texts, computer codes, etc.*

Notation. *A set $\mathcal{C}$ is called the* ciphertext space*. $\mathcal{C}$ consists of strings of symbols from an alphabet of definition, which may differ from the alphabet of definition for $\mathcal{M}$. An element of $\mathcal{C}$ is called a* ciphertext*.*

Notation. *$\mathcal{K}$ denotes a set called the* key space*. An element of $\mathcal{K}$ is called a* key*.*

Definition 56 (Encryption function)**.** *Each element $e \in \mathcal{K}$ uniquely determines a bijection from $\mathcal{M}$ to $\mathcal{C}$, denoted by $\mathbf{E}_e$. The bijection $\mathbf{E}_e$ is called an* encryption function *or an* encryption transformation*. Note that $\mathbf{E}_e$ must be a bijection if the process is to be reversed and a unique plaintext recovered for each distinct ciphertext. Similarly, for each $d \in \mathcal{K}$, $\mathbf{D}_d$ denotes a bijection from $\mathcal{C}$ to $\mathcal{M}$ (i.e., $\mathbf{D} : \mathcal{C} \to \mathcal{M}$). The bijection $\mathbf{D}_d$ is called a* decryption function *or* decryption transformation*.*

The process of applying the transformation $\mathbf{E}_e$ to a message $m \in \mathcal{M}$ (resp. $\mathbf{D}_d$ to a message $c \in \mathcal{C}$) is usually referred to as *encrypting* m (resp. *decrypting* c) or the *encryption* of m (resp. *decryption* of c).

Definition 57 (Cryptosystem)**.** *A* cryptosystem *consists of a tuple $\{\mathcal{M}, \mathcal{C}, \mathbf{E}, \mathbf{D}, \mathcal{K}\}$ where $\mathbf{E}_e : e \in \mathcal{K}$ is the encryption transformation and $\mathbf{D}_d : d \in \mathcal{K}$ the decryption transformation with the property that for each e there is a key $d \in \mathcal{K}$ such that $\mathbf{D}_d = \mathbf{E}_e^{-1}$; that is, $\mathbf{D}_d(\mathbf{E}_e(m)) = m$ for all $m \in \mathcal{M}$. A cryptosystem is sometimes referred to as a cipher. The keys e and d in the preceding definition are referred to as a key pair and sometimes denoted by (e, d). Note that e and d could be the same.*

Thus the construction of a cryptosystem requires to select a message space $\mathcal{M}$, a ciphertext space $\mathcal{C}$, a key space $\mathcal{K}$, a set of encryption transformations $\mathbf{E}_e : e \in \mathcal{K}$, and a corresponding set of decryption transformations $\mathbf{D}_d : d \in \mathcal{K}$.

Definition 58 (block cipher)**.** *A block cipher is an encryption scheme which breaks up the plaintext messages to be transmitted into strings (called blocks) of a fixed length t over an alphabet $\mathcal{A}$, and encrypts one block at a time.*

Definition 59 (Symmetric cryptosystem)**.** *An encryption scheme is said to be* symmetric *if for each associated encryption/decryption key pair (e, d), it is computationally "easy" to determine d knowing only e, and to determine e from d.*

The term *symmetric* comes from the fact that for most practical encryption schemes, $e = d$.

Definition 60 (Asymmetric cryptosystem)**.** *An encryption scheme is said to be* asymmetric *if it is not symmetric.*

Despite such simple definitions and differences, symmetric and asymmetric cryptography play very different roles in a secure communication. For example, a user may publish her asymmetric encryption key e, such that everybody can encrypt messages for her. The difficulty of retrieving d from e in an asymmetric scheme assures that only the user can decipher the messages. Due to this fact we will often refer to the encryption key of an asymmetric cryptosystem as the *public key*, and to the decryption key as the *private key*. It is necessary, however, to remark that in all practical

cases asymmetric cryptography algorithms has greater computational complexity than symmetric ones. For this reason their use is often limited to the secure exchange of a symmetric key between two parties, that can afterwards use symmetric cryptography to secure their communications.

Asymmetric cryptography does provide many handy tools to overcome particular tricky problems. For example the Diffie-Hellman key exchange protocol [66] based on the discrete log allows the secure exchange of keys between two parties over an insecure channel. Another example of the use of asymmetric cryptography are modern digital signatures, which are described by the following three algorithms.

Definition 61 (Key generation algorithm)**.** *A* key generation algorithm $\mathcal{K}$*, which, on input* 1^k*, where* k *is the security parameter, outputs a pair* (p_k, s_k) *of matching public and private keys. Algorithm* $\mathcal{K}$ *is probabilistic.*

Definition 62 (Signing algorithm)**.** *A* signing algorithm Σ*, which receives a message* m *and the private key* s_k*, and outputs a signature* $\sigma = \sum_{s_k}(m)$*. The signing algorithm might be probabilistic.*

Definition 63 (Verification algorithm)**.** *A* verification algorithm $\mathcal{V}$ *, which receives a candidate signature* σ*, a message* m *and a public key* p_k*, and returns an answer* $\mathcal{V}_{p_k}(m, \sigma)$ *testing whether* σ *is a valid signature of* m *with respect to* p_k*. In general, the verification algorithm need not be probabilistic.*

In order to achieve such functionalities various algorithms have been proposed. We will describe in the remainder of this section two of the basic constructions that provide the above functionalities.

1.2.2 Security Models

A fundamental premise in cryptography is that the sets $\mathcal{M}, \mathcal{C}, \mathcal{K}, \{\mathbf{E}_e : e \in \mathcal{K}\}, \{\mathbf{D}_d : d \in \mathcal{K}\}$ are public knowledge. When two parties wish to communicate securely using an encryption scheme, the only thing that they keep secret is the particular key pair (e, d) which they are using, and which they must select.

Informally a cryptosystem is said to be *breakable* if a third party, without prior knowledge of the key pair (e, d), can systematically recover plaintext from corresponding ciphertext within some appropriate time frame. We provide more precise definitions below.

Cryptographic Goals

The objective of information security is to provide a set of tools and techniques in order to assure a given set of goals on the digital information exchanged between two or more parties. We list hereafter the goals of information security.

Definition 64 (Confidentiality)**.** Confidentiality *is achieved when the exchanged messages can be read only by authorized users.*

Definition 65 (Data Integrity)**.** Data Integrity *is achieved when it is possible to detect the unauthorized alteration of transmitted data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.*

Definition 66 (Authentication)**.** Authentication *goal is achieved when it is possible to assure that a message was sent by the purported author.*

Definition 67 (Non repudiation)**.** Non repudiation *goal is achieved when it is not possible for one user to deny some previous commitment or action.*

Attacker Models

The security of a cryptosystem can be defined only with respect to the resources of the attacker and her objectives or goals. For example, an attacker which knows the symmetric key of a cryptosystem can systematically break it. It is thus more interesting to evaluate the difficulty for the attacker to retrieve the key when she does not know it. We thus recall the classical framework of resources and goals which is considered for an attacker against a cryptosystem.

All attackers are assumed to employ *Brute-force* attacks. In this scenario the attacker tries all possible keys until the correct one is found. While almost all ciphers are vulnerable to such an attack (for a counterexample refer to the *one time pad* citeMil1882), the difficulty relies not on the cryptosystem properties but only on the key length. If the key is sufficiently long, then the time required to retrieve the key makes the attack useless with respect to the cryptographic goal. For example, in some exchanges the confidentiality of data may be required only for a few minutes, if an attacker needs several days to brute force the key, then the cryptosystem is considered secure against brute force with respect to key recovery. The resources which the attacker may use are the following:

Definition 68 (Ciphertext-only attack (COA))**.** *In this scenario the attacker has access only to the ciphertexts which are exchanged between parties, and has no access to the plaintext*

Definition 69 (Known-plaintext attack (KPA))**.** *In this scenario the attacker has access to a limited number of pairs of plaintext and the corresponding enciphered texts.*

Definition 70 (Chosen-plaintext attack (CPA))**.** *In this scenario the attacker can choose a number of plaintexts to be enciphered and have access to the resulting ciphertexts. If furthermore at each step the attacker can analyze the previous result and choose the next plaintext, it is called* Adaptive Chosen-Plaintext Attack (CPA2)*.*

Definition 71 (Chosen-ciphertext attack (CCA))**.** *In this scenario the attacker can choose arbitrary ciphertext and have access to plaintext decrypted from it. If furthermore at each step the attacker can analyze the previous ciphertext-plaintext pairs before choosing the next ciphertext, it is called* Adaptive chosen-ciphertext attack (CCA2)*.*

Definition 72 (Related-key attack)**.** *In this scenario the attacker has access to ciphertext encrypted from the same plaintext using other (unknown) keys which are related to the target key in some mathematically defined way.*

The goals of the attacker are:

Definition 73 (Total break)**.** *Disclosing the private key of the cryptosystem. It is the most drastic attack. It is termed* total break*.*

Definition 74 (Semantic Security). *In cryptography, a cryptosystem is* semantically secure *if any adversary that is given the ciphertext of a certain message m (taken from any distribution of messages), and the message's length, cannot determine any partial information on the message with probability non-negligibly higher than all attackers that only have access to the message length (and not to the ciphertext). A cryptosystem is said to be* semantically broken *if it is not semantically secure.*

Definition 75 (Indistinguishability). *The* indistinguishability *property of a cryptosystem is characterized by the following game. The adversary chooses two sequences of message pairs,* $(M_{0,1}, M_{1,1}), \ldots, (M_{1,q}, M_{1,q})$*, where, in each pair, the two messages have the same length. The encryption scheme is used to produce a sequence of ciphertexts* $C_1, \ldots, C_q$ *that is returned to the adversary, where either (1)* C_i *is an encryption of* $M_{0,i}$ *for all* $1 \leq i \leq q$ *or, (2)* C_i *is an encryption of* $M_{1,i}$ *for all* $1 \leq i \leq q$*. During encryptions, the encryption algorithm uses the same key but fresh randoms, or an updated state, each time. The encryption scheme is said to be* indistinguishable *if the adversary cannot distinguish with probability better than random guess whether* $M_{0,1}, \ldots, M_{0,q}$ *or* $M_{1,1}, \ldots, M_{1,q}$ *were encrypted.*

Definition 76 (Malleability). *An encryption algorithm is* malleable *if it is possible for an adversary to transform a ciphertext into another ciphertext which decrypts to a related plaintext. That is, given an encryption of a plaintext* m*, it is possible to generate another ciphertext which decrypts to* $f(m)$*, for a known function* f*, without necessarily knowing or learning* m*.*

It is remarkable that in some particular scenarios, like homomorphic encryption, malleability is considered as a positive property rather than a weakness.

The goals of an attacker against signature schemes slightly differs:

Definition 77 (Universal forgery). *Constructing an efficient algorithm which is able to sign any message with significant probability of success. This is called* universal forgery*.*

Definition 78 (Existential forgery). *Providing a single message/signature pair. This is called* existential forgery*.*

In the following sections we provide some example of cryptosystems which are of particular interest for this work.

1.2.3 Advanced Encryption Standard

We recall that a cryptosystem is said to be symmetric if it is computationally easy to retrieve the decryption key from the encryption key and vice-versa (59).

The Advanced Encryption Standard (AES) has been published in 2001 after a process started in 1997 by the National Institute for Standards and Technology (NIST) [81]. Its development has been motivated by some serious weaknesses found in its predecessor, the Data Encryption Standard (DES), which was standardized in 1977. The AES is an SPN-based symmetric key cryptosystem originally known as Rijndael. An SPN cryptosystem, from the acronym of Substitution-Permutation-Network, denotes a general cryptosytems construction, which tries to realize the suggestions of Shannon [159,160]. Shannon suggested that a cryptosystem should perform two main operations on input values, namely substitution and permutation, where substitution is in charge of breaking linearity between input and output, and permutation should spread input differences as much as possible into the output. The SPN construction suggests to

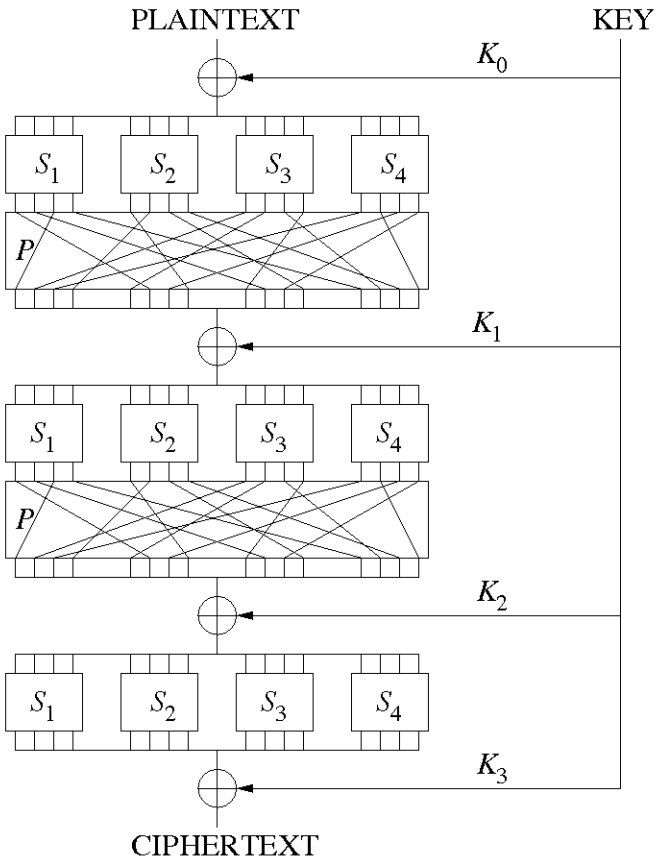


Figure 1.1 – An example of Substitution-Permutation Network with 3 rounds.

build a set of rounds which applies substitution and permutation blocks to the round input. An example of an SPN network is depicted in Figure 1.1, where the S_i blocks are the substitution boxes and P represents the permutation block.

A block-size of 128 bit can be used with three different key-lengths, namely: 128, 192, or 256 bits. The number of rounds **Nr** associated with each key is 10, 12, and 14, respectively.

The AES internal state is usually represented as a matrix of 16 bytes, where the 128-bit of the state are arranged row-wise. AES transformations operate bit-wisely, where each byte is interpreted as an element in the field $GF(2^8)$. Therefore each byte is interpreted as a finite field element using the polynomial representation $\sum_{i=0}^{7} b_i X^i$. The AES is composed of four functions, denoted *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*.

SubBytes The *SubBytes* transformation is a non-linear byte invertible substitution that operates independently on each byte of the State using a substitution table (S-Box). The SubBytes is composed of the following two functions:

- Inversion in the finite field $GF(2^8)$ where the element 0x00 is mapped to itself.

- The affine transformation which maps each byte b_i by using the transformation: $b_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$, where $\oplus$ is the addition modulo 2 and c_i is the i-th bit of 0x63.

ShiftRows In the ShiftRows transformation, the bytes in the last second, third and fourth rows of the State are cyclically shifted over 1, 2, and 3 bytes respectively.

MixColumns The MixColumns transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

AddRoundKey In the AddRoundKey transformation, a Round Key of the same size of the State is combined with the latter by an addition modulo 2.

Alg. 1 describes the encryption of one block of plaintext. We do not explicitly state the decryption algorithm, as it only requires to perform each operation backward and use the Round Keys from the last to the first.

Algorithm 1: AES Encryption Algorithm

Inputs : byte in[16], word w[4 * **Nr** + 1]
Output: byte out[16]

```
byte state[16]
state = in
AddRoundKey(state, w[0, Nb-1])
for round = 1 step 1 to Nr-1 do
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
out = state
return (out)
```

Known attacks

When AES has been conceived particular choices has been guided by attacks discovered in its predecessor, the DES. As an example the structure of the S-Boxes has been conceived in order to thwart differential and linear cryptanalysis [24, 120].

Since its publication the AES security has been challenged by a very broad set of attacks. Some attack have been found on reduced versions of the cipher, for example Biryukov *et al.* [26] show how to break AES variants with up to 10 rounds. The best attacks on full AES has been obtained by Bogdanov *et al.* [33] being able to retrieve the AES key with $2^{126.0}$ operations for AES-128, $2^{189.9}$ for AES-192 and $2^{254.3}$ for AES-256. A number of related-key attacks have also been demonstrated in several publications [26–28], attaining a complexity of 2^{96} for one out of every 2^{35} keys. However, properly designed protocols should take care not to allow related-keys, thus the practicality of these attacks has been criticized. It is worth noticing, however, that such attacks still require billions of years on modern computers, thus they do not represent

a real threat for the security of involved communications. For further details on the key schedule, on implementations tricks, and an overview of the choices that led to such algorithm, we refer the reader to [64].

1.2.4 RSA

Diffie, Hellman and Merkle, in their exceptional work [66], suggested the use of trapdoor functions to provide asymmetric cryptography. However, practical realizations of such cryptosystems were only discovered about a year later by Rivest, Shamir and Adleman [147]. The cryptosystem, known as RSA by the names of its inventors is one of the most used, studied, challenged, and discussed of all times.

Definition 79 (one-way function)**.** *A function f from a set X to a set Y is called a* one-way *function if $f(x)$ is "easy" to compute for all $x \in X$ but for "essentially all" elements $y \in Im(f)$ it is "computationally infeasible" to find any $x \in X$ such that $f(x) = y$.*

Conceptually, a *one-way* function is a function that is difficult to invert once computed. Typical examples of one-way functions are Hash functions.

Definition 80 (Hash function)**.** *A* hash function *is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called* hash-values*.*

Definition 81 (Trapdoor one-way function)**.** *A* trapdoor one-way *function is a one-way function $f : X \to Y$ with the additional property that given some extra information (called the* trapdoor information*) it becomes feasible to find for any given $y \in Im(f)$, an $x \in X$ such that $f(x) = y$.*

Algorithm

The RSA algorithm is commonly used to ensure privacy and authenticity of digital data. We provide hereafter a simplified version of the RSA algorithm. Let $N = p \cdot q$ be the product of two large primes of about the same size. Then we can compute two values e, d such that $d = e^{-1} \bmod \varphi(N)$, where $\varphi(N) = (p-1) \cdot (q-1)$ is the size of the multiplicative field $\mathbb{Z}_N^*$. The pair (N, e) is the public key, and (N, d) is the private one. Let $M \in \mathbb{Z}_N^*$ be a message. M can be encrypted by computing the value $C = M^e \bmod N$. The ciphertext C can then be decrypted by computing $C^d \bmod N$. It is relatively easy to prove that the two transformations are mutually inverse. It simply comes from the fact that $ed \equiv 1 \bmod \varphi(N) \Rightarrow ed = 1 + k\varphi(N)$. As $\varphi(N)$ is the order of $\mathbb{Z}_N$, by Theorem 2 $M^{\varphi(N)} \equiv 1 \bmod N$ for any M in $\mathbb{Z}_N^*$. Now $C^d = (M^e)^d = M^{ed}$, by substituting ed with $1 + k\varphi(N)$ we obtain $C^d = M^{1+k\varphi(N)} \equiv M^1 \bmod N$. The same happens if we compute $S = M^d \bmod N$ and then $S^e \bmod N$.

As e usually denotes the public exponent, and d the private one, the transformation $C = M^e \bmod N$ is called encryption as only the owner of the private key can retrieve the initial message. On the other hand $S = M^d \bmod N$ is called signature as it only needs public parameters to retrieve the initial message and thus anyone can verify that the owner of the private key has "signed" M. A compound of many attacks on the RSA and a good resource for an initial study on the subject is the work of Boneh [35].

Complexity The complexity of one RSA exponentiation can be roughly approximated by assuming that a multiplication is performed for each bit of the exponent. In this case we can bound the operations performed during an exponentiation by the bit-size of the exponent, times the complexity of one multiplication. As modular multiplications are performed, one can expect the size of operands to be approximately the same size of the modulus, thus approximately $O(log^2(N))$ elementary operations, see Sec. 1.1.3. We thus obtain that an RSA decryption (equivalently a signature) takes $O(log(d)log^2(N))$ operations, or $O(log^3(N))$ if d and N have the same size.

CRT-RSA

The CRT-RSA algorithm allows a speedup of about 4 times over a classical RSA implementation.

The CRT-RSA speedup exploits the properties of the Chinese Remainder Theorem(Theorem 5) in order to independently compute the exponentiations modulo p and modulo q. Two subfield exponents are computed as $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ and stored together with the private key. The message M is also split in two : $M_p = M \bmod p$ and $M_q = M \bmod q$. Afterwards two exponentiations are performed:

$$\begin{cases} S_p = M_p^{d_p} \bmod p \\ S_q = M_q^{d_q} \bmod q \end{cases} \tag{1.5}$$

Finally S_p and S_q are combined to obtain the signature S by using e.g., Garner's formula: $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p)$ where $i_q = q^{-1} \bmod p$. The complexity of the CRT-RSA can be evaluated with respect to the complexity of the standard RSA. Assume that p and q have the same size, assume also that the final recombination takes negligible time with respect to the exponentiations.

The ratio between the execution time of the standard RSA and that of a single CRT-RSA exponentiation is $\frac{O(log^3(N))}{O(log^3(N^{1/2}))}$. Assuming the constant $c > 0$ embedded in the Big O notation Sec. 1.1.3 to be independent on the size of the input, for N greater than a given threshold the ratio becomes $\frac{c \cdot log^3(N)}{c \cdot log^3(N^{1/2})}$. By taking into account both sub-field exponentiations and simplifying the square root with the log we obtain: $\frac{2^3 \cdot c \cdot log^3(N)}{2 \cdot c \cdot log^3(N)} = 4$, thus standard RSA is about four times slower than the CRT-RSA with the assumptions made above, and by considering the recombination step negligible with respect to the exponentiations.

While the asymptotic complexities remain the same, the practical gain makes the RSA-CRT the favored version for implementation in most platforms.

Parameter choices The security of RSA is strictly connected with the difficulty of factorization. It is known that being able to factor the public modulus implies a complete break of the RSA cryptosystem. Indeed once p, q are known the adversary can compute $\varphi(n)$ and thus retrieve the private key as the modular inverse of e mod $(\varphi(n))$. Inversely it can be shown that knowledge of the public and private keys allows factoring the public modulus N. A probabilistic algorithm is as follows. Recall that knowledge of d and e allows to compute $de - 1 = \ell\varphi(N)$ for some $\ell \in \mathbb{Z}$. Now, $\varphi(N)$ is even so let $k = \ell\varphi(N) = 2^t r$ for r odd. As $a^{2^t r} \equiv 1 \bmod N \, \forall \, a \in \mathbb{Z}_n^*$, $a^{k/2}$ is a square root of 1 modulo N that have four roots. Nontrivial roots of 1 modulo N are of the form $x = 1 \bmod p$ and $x = -1 \bmod q$. So $gcd(x-1, N)$ reveals a nontrivial factor of N. Randomly choosing $a \in \mathbb{Z}_N^*$ the probability to obtain a nontrivial root of 1

is at least 0.5, so after u tries the attacker has probability greater than $1 - 2^{-u}$ of successful factorization. Furthermore May [121] showed that there exist a deterministic polynomial time algorithm to retrieve the factors p, q form the knowledge of e, d, N. In view of these attacks, the choice of parameters (i.e the size of p, q and their structure) must be done in order to avoid the possibility for the attacker to factor the public modulus N. There exists factorization algorithms such as the *Pollard rho* that have running time bounded using the birthday paradox by $O(\sqrt{n})$ steps. In particular if $p \lll q$ then the factorization has running time bounded by the square root of the smaller prime p. It is thus wise to choose p, q of the same size $n^{1/2}$ in order to avoid such attacks. The best known general factorization algorithm is the general number field sieve which have complexity $O\left(\exp\left(\left(\frac{64}{9}b\right)^{\frac{1}{3}}(\log b)^{\frac{2}{3}}\right)\right)$. Most recent efforts in factorization techniques demonstrated the possibility to factor a 768-bit RSA modulus by using high sophisticated equipment [107]. In view of these results NSA recommends to use modulus of at least 2048 bits as of 2014. Concerning the public and private keys it has been shown by Wiener [173] that whenever $d < n^{1/4}/3$ then it is easy to recover d from N, e. We recommend the reader to refer to the work of Boneh [35] which collects all of the attacks on RSA appeared up to 2010.

1.2.5 ECDSA

In 1994 the US Digital Signature Standard DSA was proposed [79], essentially as an adaptation of the ElGamal signature scheme [73]. With the advent of ECCs, it has been remarked that it was possible to adapt the DSA to elliptic curves, providing faster algorithms for both signature and verification.

Scalar Multiplication and ECDLP. The *scalar multiplication* of the curve point P by the scalar $k \in \mathbb{N}$ is denoted $[k]P$. This is defined as the addition of the point P to himself k times. The interest of scalar multiplication in cryptography is due to its computational *one-wayness*. In fact it is easy to compute the result point $Q = [k]P$ but it is computationally difficult to compute the integer k knowing Q and P. This problem is known as the ECDLP and it is one of the fundamental building blocks used to construct cryptographic protocols on elliptic curves [109, 127]. The ECDLP is an interesting problem that challenges mathematicians from decades. For example, by using the Pollard's ρ [138] or Shanks' "baby-step/giant-step" algorithm [158], the problem can be solved in time polynomial on the square root of the biggest factor of the order of the base. For example, for a curve of order q prime, the complexity of the aforementioned attacks is $O(q^{0.5})$ and no significantly better algorithms are known.

Algorithm

Signer and verifier must beforehand agree on a common curve $\mathcal{E}(\mathbb{F}_p)$, and a point $P \in \mathcal{E}(\mathbb{F}_p)$. Both the curve $\mathcal{E}(\mathbb{F}_p)$ and the point on it are public parameters. We will denote with $f_x(P)$ the function that returns the x-coordinate of a point P on the elliptic curve. The signer then produces a private key denoted $d \in \mathbb{Z}_{q-1}$ and a public key $Q = d[P]$, shared with the verifier. In order to sign a message m, the user produces an ephemeral key k and computes the point $R = [k]P$. She then computes the value r by extracting the x-coordinate of the point R. She afterwards computes

$$s = \frac{SHA(m) + rd}{k} \bmod q\,, \tag{1.6}$$

where $SHA(m)$ denotes the algorithm described in the US Secure Hash Standard [162]. Once obtained the signature (r, s), the verifier computes $h_m = SHA(m)$, and accepts the signature if

$$r = f_x([s^{-1}([h]P + [r]Q)]) \tag{1.7}$$

holds.

Known Attacks It has been showed that as soon as two signatures shares the same ephemeral key k, then the secret key can easily be retrieved by an attacker. By using Eq. 1.6 we can write the two signatures as

$$\begin{cases} s_1 = \frac{h_{m_1}+rd}{k} \bmod q \\ s_2 = \frac{h_{m_2}+rd}{k} \bmod q \end{cases}$$

Then the attacker can retrieve the secret d by computing $s_1 - s_2 = k(h_{m_1} + rd) - k(h_{m_2} + rd)$, which gives $k(h_{m_1} - h_{m_2})$. As the value $h_{m_1} - h_{m_2}$ can be computed by the attacker which knows m_1 and m_2, she can inverse the value and retrieve k.

If it was not sufficient to show that a fresh k must be regenerated at each execution, Howgrave-Graham *et al.* [101] and Nguyen *et al.* [132] showed that the knowledge of a few bits of k on different signatures are sufficient to retrieve the secret key by using LLL techniques.

Pollard-ρ The best known algorithm for solving the DLP independently of the specific group G is Pollard's ρ algorithm. In order to compute the discrete log of $\mathbf{y}$ in base $\mathbf{g}$ a random walk is computed on the elements of G starting from $\mathbf{g}$. The elements produced are of the form $a\mathbf{g} + b\mathbf{y}$ where the next element $\mathbf{x}_{i+1}$ is chosen between

- $\mathbf{x}_{i+1} = \mathbf{g} + \mathbf{x}_i$
- $\mathbf{x}_{i+1} = 2\mathbf{x}_i$
- $\mathbf{x}_{i+1} = \mathbf{y} + \mathbf{x}_i$

depending on a random looking but deterministic computation, using e.g. hash values. The values $\mathbf{x}_i$ together with their representation $a_i\mathbf{g} + b_i\mathbf{y}$. When a collision is found in the list the requested discrete logarithm becomes known. We also recall that the expected running time of Pollard's ρ algorithm is $\sqrt{\pi n/2}$.

1.3 Secure Microcontrollers

It was back in 1968 and 1969 that the first patents for automated chip card were submitted. Later on Roland Moreno patented the memory card concept (1974). The original idea was to have a miniaturized device capable of storing information such that it was impossible for an attacker to tamper with the embedded data. Such devices were originally deployed in France to store the credit of users for public phones, such that at each call the user credit was debited by the phone machine. Later on, similar ideas were applied to banking accounts, allowing transactions without the physical exchange of money. Such microcontrollers were originally in the form of *smart cards*,

but have since been widely developed, evolving their initial form, shape and destination of use. They are nowadays real miniaturized computers, which are intended to provide secure communication between several different devices. We can today find the usual smart card format in credit cards, passports, sim cards, etc., Further miniaturized processors can be found in smartphones in the form of a *secure element*, allowing the access to secured services. Also, Machine-to-machine (M2M) applications allow secure communications between personal vehicles, or the access to on-board controls like the driving system through wireless channels. Finally, *IoT* devices start to embed some miniaturized device similar to a secure element in order to provide cryptographic services with very high security.

Concerning smart cards, their physical support is a plastic rectangle on which can be printed user readable information. The support usually also carry a magnetic stripe or a bar code label. The micro-processor presents eight contacts in accordance with the ISO 7816 standard but only six are actually useful contacts. The contacts relies the power supplies (Vcc, Vpp), ground, clock, reset and an I/O link to the micro-processor pins.

A micro-processor is build of several modules, each solving a particular function. In the remainder of this section we present the principal building blocks which can be found on a modern micro-processor architecture, from processing units to memories. We provide a brief detail of its characteristics and some figures of state-of-the-art examples. We afterwards briefly introduce the concept of physical attacks that will be the object of this manuscript.

1.3.1 Processor

A *Processor* or central processing unit (CPU) is the electronic circuitry that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions. A processor contains several building blocks, mainly operation units like an ALU, and memories like registers. We now detail each of them.

Definition 82 (ALU)**.** *An arithmetic and logic unit (ALU) is the digital electronic circuit within a CPU that performs arithmetic and bitwise logical operations on binary numbers. The inputs of the ALU are called operands, and the operation-code indicating the operation to perform on the operands. Additionally, an ALU often exchanges information with a status register which carries information on the result of the current or previous operations.*

Definition 83 (Register)**.** *A processor register is a small amount of storage available as part of a digital processor, such as a central processing unit (CPU). Such registers are typically addressed by mechanisms other than main memory and can be accessed faster. Almost all computers, load data from a larger memory into registers where it is used to feed operands and code to an ALU.*

The size of the CPU registers actually defines the size of data on which CPU can work on each instruction, thus for example we can find smart cards with 8-bit up to 32-bit CPU registers. In general a micro-processor may contain a number of different ALUs, a general purpose one for coding communication routines, high level protocols, etc., and possibly further computation units specific to some kind of algorithm. For example it is possible to find an integrated AES co-processor to provide high throughput AES routines. In order to keep the assembler language the more portable, the instructions to

the co-processors are usually provided by writing into dedicated registers. Such registers contain for example the particular cipher settings key size, encryption or decryption, key value, etc., and many other depending on the co-processor functional specification and objective.

Definition 84 (Memory)**.** *Data storage on a micro-processor is referred as memory.*

Different types of memories can be present at the same time on a micro-processor, for example, ROM, EEPROM, FLASH, etc. Each kind presents different scopes, for example ROM memories are writable only once, thus the program executed from ROM cannot be modified once written during the mask-producing stage. Instead, RAM memories can be written and read at will and store information as long as they are powered.

In order to provide an example of the current microcontrollers technologies, we list in Figure 1.3.1 some of the characteristics of the top products targeting secure banking applications. The manufacturers have been chosen as the three most important manufacturers according to their stock price.

Manufacturer	Family	Memory	CPU	Sym	Asym
NXP Semiconductors	SmartMX2	• 384KB ROM • 144KB EEPROM • 8KB RAM	8/32-bit	DES, AES	RSA, ECC
Infineon Technologies	SLE	• 348KB Flash • 10KB RAM	16-bit	DES, 3DES, AES	RSA, ECC
STMicroelectronics	ST33	• 512-1280KB Flash • 30kB RAM	ARM SC300	3DES, AES	RSA, ECC

1.3.2 Random Sources

A smart card is often responsible for producing its own keys and secret parameters, thus it is a fundamental security requirement that it is capable of producing reliable random bits. During for example an ECDSA signature, the card is required to provide the ephemeral key k. It has been shown that the difficulty for an attacker to guess even a few bits of such a value between different executions is fundamental in order to assure the security of the overall algorithm [101, 132]. That is for such reasons that every smart card provides a unit that is responsible for generating random bits.

Definition 85 (TRNG)**.** *A* True Random Number Generator *(TRNG) is a hardware device that generates random numbers from a physical process.*

Typical examples of sources of such physical processes are thermal noise or the photoelectric effect.

Definition 86 (PRNG)**.** *A* Pseudo Random Number Generator *(PRNG) also called* deterministic random bit generator *is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers.*

The sequence generated by the PRNG is not truly random, as it can be completely determined by a small set of initial values called *seed*. Despite being deterministic, they can be very useful in particular settings, where for example the output bits are not directly used as keys.

1.3.3 Physical Attacks

In order to introduce the concept of physical attacks we like to cite an interesting work of Zhou *et al.* [182]:

> [...] This kind of "separation of concerns" between security mechanisms and their implementation has enabled (and is, arguably, necessary for) rigorous theoretical analysis and design of cryptosystems and security protocols. However, in the process, various assumptions are made about the implementation of security mechanisms. For example, it is typically assumed that the implementations of cryptographic computations are ideal "black-boxes" whose internals can neither be observed nor interfered with by any malicious entity. [...] In practice, however, these security mechanisms alone are far from being complete security solutions. It is unrealistic to assume that attackers will attempt to directly take on the computational complexity of breaking the cryptographic primitives employed in security mechanisms. An interesting analogy can be drawn in this regard between strong cryptographic algorithms and a highly secure lock on the front door of a house. Burglars attempting to break into a house will rarely try all combinations necessary to pick such a lock; they may break in through windows, break a door at its hinges, or rob owners of a key as they are trying to enter the house. Similarly, almost all known security attacks on cryptographic systems target weaknesses in the implementation and deployment of mechanisms and their cryptographic algorithms. These weaknesses can allow attackers to completely bypass, or significantly weaken, the theoretical strength of security solutions.

The first works on the subject are those of Kocher *et al.* from 1996 to 1999 [110–112]. However, some hints on the risk posed by side-channels had already been suggested by Van Eck [168] concerning phreaking on CRT displays in 1985. While these were the first academic results, various secret services around the world already exploited such weaknesses as of WWII. For example the NSA reports the deciphering of encrypted teletyped messages using a far away freestanding oscilloscope [83] as of 1943. The MI5 exploited EM and acoustic emanations to spy on the French and Egyptian embassies in London during the *STOCKADE* and *ENGULF* operations [176] of the cold war.

A *physical attack* thus stands for an attack on an embedded cryptosystem that exploits the physical behavior (*i.e.* interactions) of the micro-processor with the environment in order to obtain sensitive information susceptible of allowing the attacker to break the cryptosystem. It is thus mandatory to update the attacker model in order to take into account such developments. We can distinguish two different behaviors for the attacker, which lead to two almost orthogonal sets of attacks types. A *Fault Attack* (FA) is an attack on a cryptosystem where the attacker tries to tamper with the encryption device in order to produce an erroneous computation. These are called *active attacks* due to the active role of the attacker during the tampering, trying to modify the normal execution flow, memory values, etc., of the micro-processor, in order

to exploit the faulty result to retrieve sensitive information. We analyze in detail this subject in Chap. 2.

On the other hand a *side-channel* is any information that can be retrieved from the encryption device that is neither plaintext nor ciphertext and that does not involve tampering with the encryption device. Consequently, a *Side-Channel Attack* (SCA) exploits side-channels to retrieve sensitive information on the cryptosystem. There exist several such physical interactions, for example time attacks, acoustic or power attacks. We provide a more precise treatment of the subject in Chap. 3. The terminology in the literature is not quite set, thus some work may refer to side-channel attacks as *passive attacks* in contrast to *active attacks*. Also, some authors denotes side-channel attacks with the acronym PA (for Power Analysis) for historical reasons.

It is the subject of this work to extend and improve current knowledge on side-channel and fault attacks and countermeasures on microprocessors.

Chapter 2

Fault Attacks

" If it is provably secure, it is probably not"

Lars Knudsen

Contents

In this chapter we start by presenting the classical model of fault attacks in view of the resources of the attacker described in Chap. 1. We then present the two main different families of fault attacks: simple and differential. In particular we provide some classical examples for both and introduce the classical countermeasures used to thwart them. Afterwards we present our contributions to the area of so-called infective computation, which seems a promising yet subtle subject for the research community. In particular in Sec. 2.2 we present our publication at the FDTC 2015 workshop which analyzes the security of some RSA infective countermeasures [16]. Afterwards we present our contributions to infective symmetric cryptanalysis. In particular in Sec. 2.3 and Sec. 2.4 we present the results of our analyses published at FDTC 2014 on two symmetric infective countermeasures [15]. Afterwards we analyze the security of a recent publication in Sec. 2.5 [17]. The results of this work have been published at COSADE 2016. All the previous results are joint works with Christophe Giraud. We conclude this chapter with our contribution to elliptic curve security against fault attacks, where we show that state-of-the-art countermeasures may not be secure as one may consider in Sec. 2.6. We have published this result at COSADE 2014 [13].

2.1 Fault Attacks

Cryptosystems are nowadays implemented as software or hardware circuits in portable devices, like smartcards, smartphones, and many other devices of the "smart" fauna. It thus becomes practical for an attacker to steal a smart device to attack it. It is also common to provide users with a device embedding a third party secret key. It seems counter intuitive, but it is at the base of very useful applications like music distribution and digital right management control. However, the presence of secret keys on the users' device represents a great risk, and the implementation thus needs to be protected from attacks. It is thus evident that simple black box attacks are no more sufficient to model modern attackers, which may be impersonated by the very user of the device, which can tamper with the cryptosystem internals in order to modify its execution. It is thus the objective of this session to present the threats caused by attackers tampering with the running cryptosystem, and to highlight some of the new attack paths and countermeasures that we suggested in this field.

As we have discussed in Chap. 1, a fault attack is an active attack where the attacker tries to disturb the execution flow or the data manipulated by the microprocessor in order to obtain a wrong result. We recall here the definition of fault attacks:

Definition 87 (Fault Attack)**.** *A* Fault Attack *(FA) is an attack on a cryptosystem where the attacker tries to tamper with the encryption device in order to produce an erroneous computation.*

2.1.1 Fault Vectors

Several ways exists to tamper with the code execution and memories. The first example of disturbances where the accidental effect of cosmic rays at high altitudes and outer atmosphere on semiconductors [183] encountered for example by the NASA or Boeing during their missions. Further researches allowed the community to discover many other fault vectors, which can be better controlled (at least up to some degree) by the attackers. We list hereafter the most common fault vectors found in literature [8]:

- *Power Glitches*: Inject a glitch (i.e an instantaneous peak), into the VCC/Gnd microprocessor pins.
- *Clock Glitches*: Inject an asynchronous instantaneous change in the clock source,
- *Light*: Inject a powerful light beam into the silicon's microprocessor. In particular laser sources are light sources with high instantaneous power and tight frequency band often centered around the infrared.
- *Electro-Magnetic* (EM): Inject a particular electro-magnetic radiation into the microprocessor' metal layers.

The exact error induced in the computation by a fault depends on many factors, ranging from the fault vector, the fault locality, the technology of the microprocessor, and many more. As a result it is difficult to know in advance the kind of fault which an algorithm has to withstand. Instead, it is common to produce a list of probable faults and study the resistance of the algorithm against them. Such a list is often denoted as the fault model, and may be proper to the fault vector and the particular microprocessor. We introduce in the next section the most common fault models used in the literature.

2.1.2 Fault Models

Fault models represent the possible kinds of error that may be introduced by the attacker into the code and the probability to obtain each of them. For example it is common to assume that an attacker can turn the value of a register into 0x00. Such a statement translates to the *stuck-at-zero* fault model, with associated probability of obtaining such particular fault. Other fault models, that are indeed less common, assume the effect of the fault to be a *XOR* of the register content with an error value, which may be controlled or not by the attacker. Furthermore, when the probability is not specified we assume that the fault occurs with 100% probability. In this section we recall some of the most popular fault models used in the literature [7, 105].

Random Error In the random error fault model scenario, the attacker can change the value of a chosen variable into a random value unknown to the attacker. We remark that it is often also useful to consider the so called *unknown constant error* model, which is similar to the random error one but assumes that the random value injected into the value remains constant among different fault campaigns.

Stuck-at With a stuck-at fault, we model an attacker that can set the value of a variable to a given value during a particular step of the algorithm. In the literature two main kinds of stuck-at faults seems to be relevant, namely stuck-at-0 and stuck-at-1. Where the former models an attacker that can set a variable to zero, the latter accounts for attackers that can set every bit of a variable to one. Even if it may seem quite unrealistic to set a whole variable to zero for example, especially if it is composed of several bytes, this fault model finds real application when variables are passed by reference instead of by value. This means that the value is accessed through a pointer and in this case an attacker can fault the pointer to address a zone of the memory that contains only zeros.

Instruction Skip The instruction skip fault model formalizes an attacker that can skip the execution of (one or more) consecutive instructions of her choice. Depending on the granularity of the analysis the model may account for faults that skip pseudo-code down to assembly code level instructions. As a general rule the attacker can skip instructions at the same abstraction level as the definition of the algorithm to be attacked.

Bit flip This model assumes that the attacker is able to flip each bit of a chosen variable.

We will see in the following how the different fault models can be used to break particular cryptosystems. However it is a matter of fact that some simple observations allow the reader to understand why fault attacks represent such a threat for security. We have seen in Sec. 1.2.2 that one of the goals of digital signatures is to certify that a message was sent by the purported author. In such protocols, the output of the verification algorithm is a simple "Accept" or "Refuse", 0 or 1 in binary. It thus becomes feasible for an attacker to disturb the result of the verification algorithm, turning a 1 into a 0. The receiver may thus accept a counterfeited message thus breaking the authenticity goal of the cryptosystem. Even though such attack may seem really simple and obvious countermeasures can easily thwart it, there exists a whole bunch of other attacks that prove stronger, whilst keeping the attack complexity very low. It is thus

obvious that many countermeasures have been conceived to thwart fault injections. In practice there exist general countermeasures that can detect and protect from a very wide range of faults and ad-hoc countermeasures adapted to particular algorithms. In the following we provide examples for some of them.

2.1.3 General Countermeasures

Two main kinds of general countermeasures exists against fault attacks, namely hardware and software countermeasures. While the former are very effective, they require space on chip, which in turn increases the size of the silicon surface and thus the cost of the chip. On the other hand, software countermeasures do not require physical space but they require memory/speed trade-offs which are often difficult to achieve. Furthermore, algorithmic countermeasures allow to exploit mathematical properties of the algorithm in order to reduce the countermeasure overhead. For example several suggestions for asymmetric countermeasures, mainly for CRT-RSA, have been suggested, and we investigate some of them in the following.

Hardware

The most common hardware countermeasure are current and voltage regulators. Such hardware modules allow to limit the effectiveness of glitches in the microprocessor inputs. In order to thwart laser faults, instead, light sensors are often employed. Such physical modules can detect peaks of electromagnetic radiations in some specific frequency range (namely those of lasers). Light sensors are very effective versus specific fault vectors in thwarting fault attacks, however they occupy physical space, thus they suffer from two main drawbacks. First of all there can not be too much of them because of the cost of the space on silicon. Furthermore, depending on each light sensor detection area, sensitive modules far enough may remain unprotected. For these reasons light sensors are usually considered as deterrents rather than effective countermeasures. Further example of hardware countermeasures are glitch detectors, which can thwart current/voltage peaks injected in the input pins. Despite their cost in silicon space, they are often used due to their high efficiency. More advanced techniques rely on the use of redundant circuits. Duplicating the hardware grants the detection of incoherent states between the multiple elements executing the same operations or storing the same data.

Software

Among software countermeasures, the one which emerged as one of the most general and powerful is redundancy. Similarly to the use of redundant circuits in hardware, the most general version of such countermeasure consists in doubling the algorithm execution and then comparing the results before output. If the results of the two executions are different then a security action may be taken, otherwise one of the two results can be output. The basic assumption is that the attacker is not capable of disturbing the algorithm twice with the same fault. Evolutions of such countermeasures can protect from various kinds of fault models. The countermeasure can obviously be adapted to protect from several faults, however the cost in term of execution is clearly linear in the order of faults that one wants to thwart. The redundancy countermeasure is particularly interesting in asymmetric cryptography where particular algebraic structures of the

cryptosystem can be exploited to achieve redundancy without actually doubling the algorithm. We discuss some of them in the following.

These two general countermeasures can be adapted to exploit some of the properties of the particular algorithm to be protected. In the next sections we discuss the fault effects on algorithms like the CRT-RSA, ECC and AES, and some of the countermeasures commonly adopted.

2.1.4 Fault Attack on CRT-RSA

In this section we recall the classic fault attack on CRT-RSA cryptosystems of Boneh *et al.* [21], we then provide an overview of some properties of the CRT-RSA cryptosystem that may be used to thwart such attack.

While it may seem impossible to gain information from an erroneous result, this kind of attack showed to be extremely powerful against modern cryptographic algorithms. The first ones who noticed the risk posed by faults were researchers at the Bellcore Labs [34]. As described in Sec. 1.2.4, in order to break the security of RSA it is sufficient to retrieve one of the secret factors p or q of the public modulus N. While factoring N is a well known hard problem, the so called "Bellcore" attack [34] demonstrated that if an error occurred during one of the subfield computations of the CRT-RSA, then the corresponding faulty signature can be used to factor the modulus. Indeed, if one of the two subfield computations of Eq. 1.5 is wrong, let us say S_p (the case where S_q is wrong is similar), then the recombination step produces a signature $\widetilde{S}$ wrong modulo p and correct modulo q, i.e.:

$$\begin{cases} S - \widetilde{S} \not\equiv 0 \bmod p \\ S - \widetilde{S} \equiv 0 \bmod q \end{cases} . \tag{2.1}$$

They noticed that the secret factor q of N can thus be retrieved by computing $\gcd(S - \widetilde{S}, N)$. The attack was further improved by Lenstra [114] who showed that by computing $\gcd(\widetilde{S}^e - M, N)$, the attacker does not even need the knowledge of a correct signature to recover the private key. The power of such attack relies on the facts that it only needs one faulty computation and that it is not straightforward to protect algorithms from faults. Indeed many authors provided ideas to thwart such attacks, in the following we present some of their results.

CRT-RSA Countermeasures

The CRT-RSA presents particular algebraic properties that can be used to build ad hoc countermeasures to thwart fault attacks. These properties have been exploited by various authors (*e.g.* [4, 60, 157, 171]) to suggest cheap and efficient countermeasures. Two main approaches have been suggested in the literature to thwart faults in software, namely detective and infective countermeasures. In the following we present both of them with some examples.

Detective countermeasures There exist three main families of detective countermeasures suggested in theory and still used in practice. While many other countermeasures exists [91, 145], we have chosen to limit our presentation to those necessary to understand this work and that are mostly adopted. We are going to give some details of each of them in the following. The Shamir countermeasure against fault attacks, is due to Adi Shamir and firstly appeared in [157]. The idea is to perform the two

computations of Eq. 1.5 modulo $p \cdot r$ and $q \cdot r$ respectively where r is a small random variable:

$$\begin{cases} S'_p = M^{d_p} \bmod p \cdot r \\ S'_q = M^{d_q} \bmod q \cdot r \end{cases} . \tag{2.2}$$

In this way, the integrity of each sub-field signature S'_p and S'_q can be verified by comparing the values modulo r.

Aumüller *et al.* in [4] suggested some improvement of Shamir's countermeasure. They suggest to verify that $p \cdot r$ and $q \cdot r$ have been correctly computed modulo p and q respectively. They also add other tests to verify the signature S by comparing it versus S'_p modulo p and S'_q modulo q. If an inconsistency is found, an error is returned and a security action may be taken. Once again, Yen *et al.* [181] published at ICICS 2002 an improvement of the countermeasure to thwart some new attack they found.

The countermeasure of Vigilant exploits the fields properties by using a set of four values denoted A_p, B_p, A_q, B_q. The four values have the following properties. Let the field $\mathbb{F}_p$, where all numbers are modulo p, be extended by using the modulo $p \cdot r^2$, where r, p are co-prime numbers. Now A_p, B_p have been chosen such that:

$$\begin{cases} B_p = 1 \bmod r^2 \\ B_p = 0 \bmod p \cdot r^2 \end{cases} \qquad \begin{cases} A_p = 0 \bmod r^2 \\ A_p = 1 \bmod p \cdot r^2 \end{cases} .$$

The following properties are verified:

- $A_p^2 = A_p \bmod p \cdot r^2$
- $B_p^2 = B_p \bmod p \cdot r^2$
- $A_p B_p = 0 \bmod p \cdot r^2$

Taking $m'_p = A_p m_p + B_p(1 + r) \bmod p \cdot r^2$, by using the properties of A_p, B_p one can deduce:

$$m_p'^{\,x} = A_p m_p{}^x + B_p(1 + x \cdot r) \bmod p \cdot r^2 \tag{2.3}$$

for any x. Now from the above properties it follows that:

$$\begin{cases} m_p'^{\,x} = A_p m_p{}^x \bmod p \\ B_p \cdot m_p'^{\,x} = B_p(1 + x \cdot r) \bmod r^2 \end{cases}$$

Where the property $B_p(1 + r)^x = B_p(1 + x \cdot r) \bmod r^2$ is easily proven by the fact that working modulo r^2 boils down to reduce modulo the ideal generated by r^2, which in turn means that all polynomials of degree greater than 1, (i.e: $a + rb$) are reduced to 0. Now the countermeasure consists in exploiting these properties to perform computations modulo $p \cdot r^2$, (resp. $q \cdot r^2$) and verifying the computation modulo r^2 by using $B_p \cdot m_p'^{\,x} = B_p(1 + x \cdot r) \bmod r^2$ with a simple multiplication instead of an additional exponentiation.

Finally, in order to generate A_p, B_p, it has been remarked that one can simply take:

$$\begin{cases} B_p = p \cdot (p^{-1} \bmod r^2) \\ A_p = 1 - B_p \bmod p \cdot r^2 \end{cases} .$$

Computations modulo q are symmetric.

Infective Countermeasures In 2001, Yen *et al.* [180] suggested another approach to thwart the Bellcore attack. To secure a CRT-RSA they proposed to spread a possible fault into both subfield computations results. Two protocols were proposed in [180]. Unfortunately both were broken by Yen *et al.* in [178, 179] by adapting the Bellcore attack [21].

In 2003, Blömer *et al.* suggested in [31] another protocol to realize infective computation on CRT-RSA algorithm. They suggested to return a random power of the signature if a fault is injected in one of the two subfield exponentiations. The authors' security proof was based on the resistance to all known attacks. This method called *BOS* was later broken by Wagner in [172] thus proving that such security proofs are not exhaustive.

Trying to resurrect infective computation, Ciet *et al.* [50] proposed at FDTC 2005 an adaptation of the BOS method which was resistant to the attacks found by Wagner. However, this method was broken by Berzati *et al.* in [22], in which was suggested a patch to circumvent the flaw but no security analysis was given.

In 2010, Schmidt *et al.* [153] proposed an exponentiation resistant to both side-channel and fault attacks by using an infective countermeasure. Once again this method was broken. Feix *et al.* found a flaw in the exponent blinding procedure that allows attackers to retrieve the private key [77]. They proposed a patch to thwart their attack but offered no guarantee against other possible attack paths.

In Sec. 2.2 we present our results on the security analysis of some recent propositions on asymmetric infective countermeasures.

In the following section we present the fundamental results of fault attacks on elliptic curves cryptosystems and their countermeasures.

2.1.5 Fault Attacks on ECC

Before presenting the main fault attacks published so far, we recall that the ECDLP can be efficiently solved if the order of the logarithm base is smooth [116]. This can happen for example if the input point does not lie on the curve, as observed in [3].

The first DFA on elliptic curves was published in [23]. Similarly to [3], the authors of [23] suggest to disturb the input point P after the parameters checking such that the faulted point P' does not lie on the original curve. The attacker may be able to retrieve information on the secret scalar using the corresponding faulty output since the curve on which the computation has been done probably has a smooth order. A second attack presented in [23] suggests that by disturbing intermediate values during the scalar multiplication, one may be able to retrieve information on the secret scalar. The idea is to guess the produced error value and a few bits of the secret. The attacker then computes the scalar multiplication backward up to the fault, corrects the error, and recomputes onward to obtain the result. If it is the correct result, then it is likely that the guesses were correct. It is clear that the entropy of the guesses must not be too high in order to exhaustively test them, thus the attack starts with faults near the end of the computation, then steps backward as more bits are known.

In [32], the authors propose another DFA that targets non adjacent form ECC implementations. By disturbing the sign of a point during the scalar multiplication they can retrieve the signed bits of the secret scalar k. As they need several faulty outputs to mount their attack, they focus on implementations that use the same secret scalar for all executions.

Further analysis of faults on ECC was carried out in [49]. The authors improved

the results of [23] by observing that an unknown fault on one of the coordinates of the input point produces a wrong curve parameter b which can be retrieved as a solution of the curve equation with the output point coordinates as known values. The authors also observed that a fault in the prime p defining the field $\mathbb{K}$ produces a curve where the input point will likely have smooth order, thus breaking the security of the cryptosystem. Finally [49] also shows that a fault in one of the public parameters (typically a in Eq. 1.1) is likely to transfer the elliptic curve on a new one where the ECDLP is easier to solve. They observe that by substituting the input and output point coordinates in Eq. 1.1 the solution of the obtained system allows the attacker to recover the faulted curve parameters. As it was the case for CRT-RSA, ad-hoc countermeasures have been suggested to thwart fault attacks on ECCs. We present them in the following.

ECC Countermeasures

In order to thwart fault attacks on elliptic curves several countermeasures have been conceived. First of all, the authors of [3] suggest that all implementations should verify that the point lies on the curve before performing the scalar multiplication. Namely, in order to thwart faults on the input values, Biehl *et al.* suggested in [23] to test if the input point lies on the curve before and after the scalar multiplication. Furthermore, the attacks of Biehl *et al.* [23] should also be thwarted if point on curve security countermeasure is adopted.

The authors of the BOS attack [32] suggest a more sophisticated countermeasure to thwart the attack presented in their work. The countermeasure they suggest is called combined curve check. They build a new curve containing both the given curve and a smaller one, such that scalar multiplication on the small curve is fast. Given the result of the scalar multiplication on the combined curve, with a simple modular reduction they can retrieve the result on both the given curve and the small one. Thus by executing a second scalar multiplication on the small curve they can compare the result with the one obtained from the combined curve and detect faults.

Finally, Ciet *et al.* remark in [49] that public parameters must be checked for faults prior and after the scalar multiplication. They also claim that performing these checks by using integrity checks (*i.e.:* cyclic redundancy check) or by point on curve test offer the same security level, while the former should perform faster.

We show in Sec. 2.6 that this claim is false by exhibiting an attack that can break the cryptosystem if the tests are performed by point on curve test.

The next section introduces fault attacks on AES, Due to the intrinsic difference between symmetric and asymmetric cryptography, the philosophy of the attack is quite different.

2.1.6 Fault attacks on AES

Since the appearance of fault attacks on asymmetric cryptosystems by Boneh *et al.* [34], the cryptographic community started exploring the effectiveness of faults on any sort of cryptosystem. Differential fault attacks (DFA) have been found on the Data Encryption Standard [25] and soon afterwards on AES [90]. We explain now how the DFA on AES works in its general principles. The basic principle of the DFA on AES is to inject a random fault into one byte of the MixColumns input before the last round execution. The faulty ciphertext thus obtained is then compared to a correct ciphertext derived from the same message under the same key. Under such fault assumptions, the correct and faulty ciphertext differ in four bytes due to the MixColumns execution during the

9-th round. By guessing the 32-bit value of the key on the differing bytes, one can rewind the computation of the correct and faulty ciphertext independently up to the input of the Sbox of the 9-th round. By XOR-ing the two internal states before the Sbox Input, the effect of the 9-th round AddRoundKey is suppressed due to the XOR linearity. The attacker can thus further rewind the computation up to the value of the differential between the two internal states before the execution of the MixColumns of the 9-th round. This information allows the attacker to decide if the key guess was correct. Indeed, a correct key guess leads to a differential internal state with a single byte different from 0, the one impacted by the fault. A wrong guess leads instead to a differential with more than one single byte different from 0. This attack can only retrieve 32 bits of key at a time, and some wrong guesses may, rarely, lead to false positives. To overcome these drawbacks one needs several faulty ciphertexts, usually between three and four to correctly retrieve the key. Several authors worked to improve the effectiveness and performance of this attack. In particular by adapting the fault model, the attack round, and exploiting particular AES properties, the works of Piret and Quisquater [137] and of Mukapadhyay [130] showed that one or two single faults are indeed sufficient to completely retrieve an AES-128 secret key.

In view of such attacks it is thus mandatory to provide efficient and effective countermeasures. We present in the following the most used of them.

AES Countermeasures

There exists several countermeasures to thwart fault attacks on symmetric ciphers. However most of them are based on generic countermeasures that fit several symmetric schemes rather than ad-hoc countermeasures. This is a direct consequence of the effort put into removing any property linking plaintext and ciphertext during cryptosystem development. This effort is required to remove mutual information between ciphertext and plaintext which may reduce the security of the cryptosystem from a general information theoretic point of view. Furthermore attacks like the differential and linear cryptanalysis [24, 120] have shown that relationships between intermediate rounds can be exploited in order to break the security of the cryptosystem.

We thus find among the most common countermeasures for AES the redundancy technique that aims at detecting inconsistencies between different executions on the same inputs. These techniques have been adapted in order to check only once the correct execution, for example by performing encryption-then-decryption and comparing the input of encryption with the output of decryption. Several other adaptation of the redundancy countermeasure exist, based for example on error detection and correcting codes.

It is therefore interesting to note that several authors suggested infective countermeasures for symmetric cryptosystems, mainly AES and DES, which are based on redundancy techniques. For example Lomné *et al.* [117] presented at FDTC 2012 an infective countermeasure for the AES based on the use of multiplicative random masking. Another infective countermeasure have been presented by Gierlichs *et al.* [88] at LatinCrypt 2012 and is built upon the idea of random dummy rounds. These works generated interest on infective symmetric countermeasures, we thus present our security analysis of these countermeasures, as long as some development by Tsupamudre *et al.* [165] and our subsequent analysis in Sec. 2.3, 2.4 and 2.5.

After this brief introduction to the various fault attacks and countermeasure on different cryptosystems, we are going to analyze in the remainder of this chapter several

different countermeasures that have been suggested in literature. For each of them we explain the countermeasure in detail and suggest fault attacks that can be used to break it.

We start in the next section with the analysis of two infective countermeasures suggested for the CRT-RSA. The results of our joint work with Christophe Giraud shows that the translation method to transform detective to infective countermeasures is not secure.

2.2 RSA Detective to Infective Countermeasure Translation Analysis

We present in this section our security analysis of the CRT-RSA infective countermeasures published at FDTC 2014. We show that the suggested infective versions of the classical Aumüller *et al.* and of Vigilant *et al.* countermeasures cannot withstand fault attacks. We use simple fault models like the stuck-at-zero and random-error to break the countermeasure. Furthermore we prove that the method suggested by the authors of [143] to translate detective countermeasures to infective ones is not yet mature. We finally provide evidence of the threat posed by messages with particular properties like $M = \pm 1$ and $M = 0$ with respect to the Bellcore attack.

2.2.1 Countermeasure Presentation and General Remarks

The authors of [143] developed an interesting theory to translate a detective countermeasure into an infective one. In [143] the authors provide a formal analysis of two well known detective fault countermeasures, namely those of Aumüller *et al.* [4] and of Vigilant *et al.* [60]. The use of the formal tool *finja* allowed them to simplify the two aforementioned countermeasures whilst preserving the same level of security. Afterwards they proved the security of their method in order to show that it is possible to convert a detective countermeasure into an infective one while keeping the same level of security.

We recall here Proposition 4 of [143] which provides the proof of security of the translation.

Proposition 4. *Each test-based (resp. infective) countermeasure has a direct equivalent infective (resp. test-based) countermeasure.*

Proof. (from [143]) The invariants that must be verified by countermeasures are modular equality, so they are of the form $a \stackrel{?}{\equiv} b \bmod m$, where a, b and m are arithmetic expressions. It is straightforward to transform this invariant into a Boolean expression usable in test-based countermeasures: `if a != b [mod m] then return error;`.

To use it in infective countermeasures, it is as easy to verify the same invariant by computing a value which should be 1 if the invariant holds: `c := a - b + 1 mod m;`.

The numbers obtained this way for each invariant can then be multiplied and their product c^*, which is 1 only if all invariants are respected, can be used as an exponent on the algorithm's result to infect it if one or more of the tested invariants are violated. Indeed, when the attacker perform the Bellcore attack by computing $\gcd(S - \widetilde{S^{c^*}}, N)$ as defined in Sec. 2.1.4 then if c^* is not 1 the attack would not work. □

The above result is constructive in the sense that it explicitly states how to translate from one type of countermeasure to the other.

However, we noticed that the security of the translation is only guaranteed with respect to the classical Bellcore attack that uses $\gcd(S - \widetilde{S^{c^*}}, N)$. Other variants of the Bellcore attack, like the one in [22] that uses $\gcd(N, (\widetilde{S}^e - m^\gamma))$ for example, are not taken into account. Therefore, it seems that to preserve the validity of the certificate after translation one would need to prove it for each and every variant of the Bellcore attack, which is computationally infeasible.

In the following we thus recall the two infective algorithms suggested in [143] and use the above observations to challenge the security of their results.

2.2.2 Infective Aumüller et al. Countermeasure

As described in Sec. 2.1.4, the countermeasure of Aumüller *et al.* [4] is based on an earlier proposal of Shamir [157]. Aumüller *et al.* suggested particular additional verifications in order to provide further protection versus fault attacks.

The basic idea to transform the detective countermeasure of Aumüller *et al.* into an infective one is to produce an infective exponent whenever the detective countermeasure should have returned an error. This exponent is afterwards used to raise the faulty signature to a random power. By applying the strategy described in Sec. 2.2, the authors of [143] thus suggest Alg. 2 and states that their algorithm is secure versus random and stuck-at-0 fault models in particular.

Algorithm 2: CRT-RSA with Infective Aumüller's Countermeasure [143, Alg. 13]

Inputs : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M_d \bmod N$, or a random value in $\mathbb{Z}_N$

1 Choose a small random integer r
2 $p' = p \cdot r$
3 $c_1 = p' + 1 \bmod p$
4 $q' = q \cdot r$
5 $c_2 = q' + 1 \bmod q$
6 $S'_p = M^{d_p \bmod \varphi(p')} \bmod p'$
7 $S'_q = M^{d_q \bmod \varphi(q')} \bmod q'$
8 $S_p = S'_p \bmod p$
9 $S_q = S'_q \bmod q$
10 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p)$
11 $c_3 = S - S'_p + 1 \bmod p$
12 $c_4 = S - S'_q + 1 \bmod q$
13 $S_{pr} = S'_p \bmod r$
14 $S_{qr} = S_q \bmod r$
15 $c_5 = S_{pr}^{d_q \bmod \varphi(r)} - S_{qr}^{d_p \bmod \varphi(r)} + 1 \bmod r$
16 **return** $S^{c_1 c_2 c_3 c_4 c_5} \bmod N$

At a first look it seems that Alg. 2 has a security comparable to [4]. While the random exponent used for the infection seems effective to mask a potential faulty signature, we observe that Ciet and Joye already suggested an infective countermeasure based on a random exponentiation [50] but their countermeasure was broken by Berzati *et al.* [22].

In Sec. 2.2.3 we exhibit two attacks that can retrieve the secret key from the output of Alg. 2 by using the same fault model as in [143].

2.2.3 Analysis of the Infective Aumüller Countermeasure

We present in this section two fault strategies that allow an attacker to retrieve the private key of the CRT-RSA cryptosystem when protected with the countermeasure described in Sec. 2.2.2.

Random Fault Attack on Alg. 2

Our first attack starts by querying the implementation for the signature of the message $M = 1$ (the value $M = -1$ can be used similarly) which should produce a correct signature $S = 1$. However, during the execution of Alg. 2 the attacker disturbs Step 6 with a random fault and thus transforms the variable S'_p into the randomized faulty variable $\widetilde{S'_p}$. The effect of such a fault is to produce a wrong value $\widetilde{S_p}$ at Step 8 and a wrong signature $\widetilde{S} = S_q + q \cdot (i_q \cdot (\widetilde{S_p} - S_q) \bmod p)$ at Step 10, which is not further modified until the last step.

We have noticed that such a faulty signature is not efficiently infected by the exponentiation in Step 16 of Alg. 2. Indeed we noticed that before the infection of Step 16 we have:

$$\widetilde{S} \neq 1 \bmod p \quad \text{and} \quad \widetilde{S} = 1 \bmod q \ .$$

Such a faulty signature is invariant by exponentiation modulo q but not modulo p. The infective exponentiation thus produces a random value modulo p and 1 modulo q. The attacker can thus use the faulty signature $\widetilde{S}$ to compute $\gcd(\widetilde{S} - 1, N)$ which exposes a nontrivial factor of N whatever the value of the infective exponents $c_1, \ldots, c_5$.

We also noticed that the attack presented in this section works similarly if the fault is injected at Steps 6, 7, 8 or 9 of Alg. 2, thus exposing four different fault target variables to the attacker.

Stuck-at-0 Fault Attack on Alg. 2

Our second attack on Alg. 2 exploits a fault injected by using a stuck-at-0 fault model. This attack works even if the input message of the algorithm is not chosen or unknown to the attacker. We thus assume in the following that the attacker queries Alg. 2 for the signature of a random message M.

By disturbing Step 6 with a stuck-at-0 fault during the algorithm execution, the value of S'_p is set to 0, while S'_q is not modified. These two values are then reduced modulo p and q during Steps 8 and 9 respectively, and recombined into a faulty signature $\widetilde{S}$ at Step 10. All these modifications preserve the property $\widetilde{S} = 0 \bmod p$. The faulty signature is not modified until the last step of the algorithm when the infective exponents are applied. However, as the value 0 is invariant by exponentiation, after the infection of Step 16 the faulty signature is:

$$\widetilde{S} = 0 \bmod p \quad \text{and} \quad \widetilde{S} \neq 0 \bmod q \ .$$

The attacker can thus compute $\gcd(\widetilde{S}, N)$ to retrieve the nontrivial factor p of N, whatever the value of the infective exponents.

While we assume that the fault is injected at Step 6, we remark that our attack works similarly if the fault is injected at any step from 7 through 9 of Alg. 2.

We present and analyze now the second infective countermeasure suggested in [143].

2.2.4 Infective Vigilant et al. Countermeasure

The second infective countermeasure suggested in [143] and depicted in Alg. 3, has been inspired by the original detective countermeasure of [60]. While the latter is still considered secure, we show in this work that the same does not hold true for its infective counterpart, at least as suggested in [143]. As we have discussed in Sec. 2.1.1, the Vigilant countermeasure exploits four coefficients, A_p and B_p modulo p and A_q and B_q modulo q, to perform a cheap comparison.

The authors of [143] based their countermeasure on the original idea of the algorithm but removed some error detection step and changed the remaining ones into infective exponent updates. The infective exponents are then applied to the signature before output. The rationale is that if no fault is injected then the infective exponent equals 1 and the correct signature is output, otherwise the faulty signature is raised to a random exponent such that it is useless for the attacker.

By applying the strategy described in Sec. 2.2, the authors of [143] thus suggest Alg. 3 and state that their algorithm is secure versus random and stuck-at-0 fault models in particular.

Algorithm 3: CRT-RSA with Infective Vigilant's Countermeasure [143, Alg. 11]

Inputs : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M_d \bmod N$, or a random value in $\mathbb{Z}_N$

Choose a small random integer r.
$N = p \cdot q$
$p' = p \cdot r^2$
$i_{pr} = p^{-1} \bmod r^2$
$M_p = M \bmod p'$
$B_p = p \cdot i_{pr}$
$A_p = 1 - B_p \bmod p'$
$M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \bmod p'$
$q' = q \cdot r_2$
$i_{qr} = q^{-1} \bmod r^2$
$M_q = M \bmod q'$
$B_q = q \cdot i_{qr}$
$A_q = 1 - B_q \bmod q'$
$M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \bmod q'$
$S'_p = M'^{d_p \bmod \varphi(p')}_p \bmod p'$
$S_{pr} = 1 + d_p \cdot r$
$c_p = M'_p + N - M + 1 \bmod p$
$S'_q = M'^{d_q \bmod \varphi(q')}_q \bmod q'$
$S_{qr} = 1 + d_q \cdot r$
$c_q = M'_q + N - M + 1 \bmod q$
$S' = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \bmod p')$
$S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \bmod p')$
$c_s = S' - S_r + 1 \bmod r^2$
return $S = S'^{c_p c_q c_s} \bmod N$

Once again, Alg. 3 uses a final exponentiation as infective vector, we thus raise the

same objections expressed in Sec. 2.2.2 about this choice.

2.2.5 Analysis of the Infective Vigilant Countermeasure

We present in this section two fault strategies that allow an attacker to retrieve the private key of the CRT-RSA cryptosystem when protected with the countermeasure described in Sec. 2.2.4.

Random Fault Attack on Alg. 3

The first attack that we present on the Vigilant-based infective countermeasure of [143] exploits a random fault to retrieve the value of one of the secret factors of the public modulus N. Similarly to Sec. 2.2.3, our attack starts by querying the implementation for the signature of the message $M = 1$. For the sake of simplicity we use $M = 1$ but the case $M = -1$ works similarly. During Step 15 the attacker injects a random fault such that variable S'_p is faulted into a random value $\widetilde{S'_p}$. As the fault does not impact the value S'_q, the recombination during Step 21 computes a faulty signature $\widetilde{S'}$ which is randomized modulo p but is still congruent to 1 modulo q. This value is not modified until the last step, where the infective exponents are applied. However, the faulty signature is invariant by exponentiation modulo q since $S_q = 1$. The same is not true modulo p, as $\widetilde{S'_p}$ have been randomized by the fault. The gcd between $\widetilde{S} - 1$ and N thus reveals the nontrivial factor q of the public modulus N thus breaking the security of the algorithm.

We remark that the attack presented in this section can also be applied if the fault disturbs Step 5, 8, 11, 14 or 18, thus exposing 6 different fault targets to the attacker.

Stuck-at-0 Fault Attack on Alg. 3

The second attack that we present against Alg. 3 exploits a stuck-at-0 fault to introduce an exponentiation invariant into one of the faulty signature sub-fields.

In order to mount the attack we assume that a random message M is used as input, furthermore the attacker does not need to know its value. The attacker starts by injecting a stuck-at-0 fault at Step 15 such that the variable S'_p is changed into 0. Afterwards at Step 21 the faulty variable produces a wrong signature $\widetilde{S'}$ which is equal to 0 modulo p and an unknown value, different from 0 modulo q. Afterwards the infective exponents are applied, but similarly to what we have noticed above, they do not have effect on 0, i.e. after the application of the infective exponents, the faulty signature $\widetilde{S'}$ is such that:

$$\begin{cases} \widetilde{S'} \equiv 0 \bmod p \\ \widetilde{S'} \not\equiv 0 \bmod q \end{cases} . \tag{2.4}$$

Once the attacker is in possession of such a faulty signature, she can retrieve the value of the secret prime p by computing the gcd between $\widetilde{S}$ and N.

For the sake of simplicity we only present the effect of a fault on Step 15, however the attack presented in this section works similarly for a fault injected at Steps 5, 8, 11, 14 and 18, so that the attacker can freely choose among 6 different steps of the algorithm as a target for its fault.

2.2.6 Conclusion

Our work exhibits a critical flaw in the main proof of security of the two infective countermeasures for CRT-RSA suggested at FDTC 2014. Furthermore two attacks are presented on each of the two infective countermeasures derived by using the translation method of [143]. Although the corresponding detective countermeasures have been certified secure versus randomizing, stuck-at-0 and instruction-skip attacks by the *finja* formal verification tool, the security of the infective countermeasures has been certified by using an original translation method. We show that the proof of the translation security provided in [143] is flawed. We then use such a remark to develop two attacks on the infective algorithms suggested in [143]. For both countermeasures, the first attack uses a chosen message input and retrieves the secret key by using one single randomizing fault. Our second attack does not require knowledge of the input message and uses a single stuck-at-0 fault to retrieve the private key. The basic idea of our attacks is to use exponentiation invariants to bypass the infective countermeasures. We remark that by accepting input messages equals to $\pm 1 \bmod N$, Alg. 2 and Alg. 3 inherently present some security problems. Our work, however, shows that the algorithms are not secure even when these particular values are not accepted on input. The aim of this work is to point out that the approach taken in [143] needs further investigation and improvements to be sound.

The next section presents another attack on asymmetric cryptosystems. We introduce the concept of *common points*, and show how to use them to break some countermeasures for elliptic curve cryptosystems.

Sec. 2.2 focused on fault attacks on the RSA cryptosystems. The following three sections instead concerns symmetric infective countermeasures.

We present in the next section the results of our joint analysis with Christophe Giraud on a symmetric infective fault countermeasure based on multiplications.

2.3 Analysis of the "Multiplicative" AES Infective Countermeasure

Infective symmetric countermeasure appear to be an attractive alternative to redundant countermeasures because in principle they do not need a double computation. We will see, however that every infective symmetric countermeasure makes use of a double computation to perform. This is mainly due to the intrinsic lack of exploitable properties between input and output of the cryptosystem. In this section we describe and analyze the first AES infective countermeasure that appeared in the literature.

2.3.1 Countermeasure presentation

The infective countermeasure for the AES algorithm by Lomné *et al.* [117] presented at FDTC 2012 is based on the use of multiplicative random masking. The authors state that infection countermeasures applied before the final round of the AES algorithm are useless, mainly due to the various attacks that have been published on the last round. Let us thus apply their countermeasure on an AES ciphertext C_0 and a redundant computation ciphertext C_1, masked with Boolean masks M_0 and M_1 respectively, cf. Alg. 4.

Algorithm 4: FDTC 2012 Countermeasure

Inputs : $C_0 \oplus M_0$, $C_1 \oplus M_1$, M_0, M_1 and a fresh mask $M_2 \neq 0$ and $\neq 1$
Output: The correct ciphertext C_0 or the infected ciphertext $\widetilde{C}_0 \oplus M_2 \cdot (\widetilde{C}_0 \oplus C_1)$

1 $a \leftarrow M_2 \cdot (C_0 \oplus M_0)$
2 $b \leftarrow M_2 \cdot (C_1 \oplus M_1)$
3 $c \leftarrow a \oplus b$
4 $d \leftarrow M_0 \oplus M_1$
5 $e \leftarrow M_2 \cdot d$
6 $f \leftarrow (C_0 \oplus M_0) \oplus c$
7 $g \leftarrow f \oplus e$ $\quad [= (C_0 \oplus M_0) \oplus M_2 \cdot (C_0 \oplus C_1)]$
8 **return** $(g \oplus M_0)$

The authors remark that upon a correct execution $C_0 = C_1$. Once the algorithm is executed, the user obtains C_0 if no fault was injected during the AES execution or the faulty ciphertext $\widetilde{C}_0$ masked by $M_2 \cdot (\widetilde{C}_0 \oplus C_1)$ otherwise. For optimization purposes the authors state that without loss of security the product in GF(2^{128}) may be replaced by 16 products in GF(2^8) as long as the bytes of M_2 remain independent. In the following we will thus take into account this improvement. The random distribution property of M_2 should provide enough masking on the faulty output, but the restriction to be different from 0 and 1 seems to introduce a bias on the output. We will show in the following how an attacker can exploit this bias.

2.3.2 Attack Description

Let us assume that the attacker is able to use the fault model suggested in the original paper [117], that is she can inject the same unknown byte error among different executions. Moreover we assume that the attacker disturbs the first AES execution such that the faulty input $\widetilde{C}_0$ of Alg. 4 corresponds to $C_{0_i} \oplus \varepsilon$ on byte i. In such a case the attacker obtains $g_i \oplus M_{0_i} = \widetilde{C}_{0_i} \oplus M_{2_i} \cdot \varepsilon$ as value for the i^{th} byte of the output.

By observing that the byte M_{2_i} is always different from 0 and 1 and that the fault is constant, one deduces that the two values $\widetilde{C}_{0_i}$ and $\widetilde{C}_{0_i} \oplus \varepsilon = C_{0_i}$ are never output at byte i when reiterating this attack on another execution of the AES-128 using the same input message. The attacker will thus disturb the AES execution until she obtains 254 different values for the i^{th} byte of the output of Algorithm 1. The two missing values being C_{0_i} and $\widetilde{C}_{0_i}$, the attacker can thus infer the value of $\widetilde{C}_{0_i}$ since the correct ciphertext value C_0 is known.

By injecting faults on the 8^{th} round, the diffusion property of the cipher will spread the constant error among the whole output. She can thus retrieve all the 16 bytes of the faulty ciphertext with the same faults used to retrieve the first one. Once the faulty ciphertext value is recovered, the attacker is thus able to retrieve the full AES-128 key by using the attack described in [130].

2.3.3 Simulations

Simulations of the attack were done on AES-128 by disturbing one byte of the state of the 8^{th} round before the MixColumns transformation. Different scenarios were simulated depending on the constant fault probability (*i.e* the percentage that the same fault is induced). For a 100% constant fault injection success rate, the attacker reit-

erates the fault until all but two values are returned for each byte. For fault injection success rates below 100% the attacker retains the two values that appear the less, and as said before she eliminates the one which is equal to the correct ciphertext.

Figure 2.1 shows the success rate over 100 000 simulations of the different attacks (for probabilities to inject the same error equals to 100%, 90% and 75%) to recover the value of the faulty ciphertext. Once the faulty ciphertext value being recovered, one applies the attack of [130] to deduce the corresponding AES-128 key.

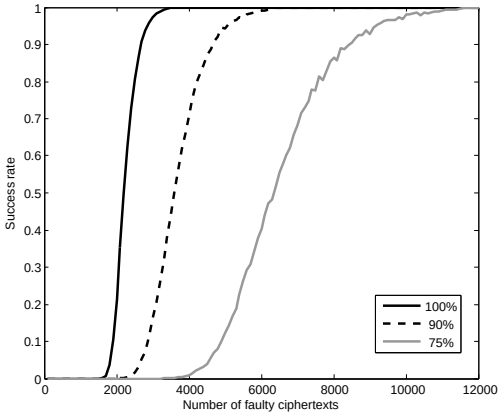


Figure 2.1 – Success rate vs number of faulted ciphertexts depending on the attacker's capability to induce a constant error.

Further results on simulations are given in Table 2.1 where one can observe that for the success rate to reach 99%, on average 3 300, 6 000 and 11 000 AES executions are required if the attacker's ability to inject the same constant error is 100%, 90% and 75% respectively. Table 2.1 also shows that even for an awkward attacker that is able to inject the same constant fault with only 25% probability, the attack is still possible and with 145 000 AES executions the success rate reaches 99%.

Percentage of constant error \ Success rate	90%	99%
100%	2 700	3 300
90%	4 700	6 000
75%	8 500	11 000
50%	24 000	32 000
25%	115 000	145 000

Table 2.1 – Number of disturbed AES executions required depending on constant error and success rates.

2.3.4 Conclusion

This section described a fault attack on the first symmetric infective countermeasure known in the literature. The infective countermeasure of Lomné *et al.* [117] is based on multiplicative random masks. Our attack exploits a bias on the multiplicative mask and retrieves the full AES-128 key with only 2 200 faults on average. It seems possible

to patch the countermeasure by generating a single $128-bit$ mask value, however, the overhead generated by a $128-bit$ by $128-bit$ multiplication seems prohibitive.

In the following section we present the results of our joint work with Christophe Giraud on the analysis of the security of another symmetric infective countermeasure conceived for DES and AES against fault attacks.

2.4 Analysis of the "Dummy-Rounds" AES Infective Countermeasure

In this section we describe the symmetric infective countermeasure suggested in [88]. We discuss its security and show a fault attack that allows to break the countermeasure security with a limited number of faults.

2.4.1 Countermeasure Presentation

At LatinCrypt 2012, Gierlichs *et al.* [88][1] proposed an infection countermeasure based on the use of dummy rounds and declined it for the AES and DES algorithms. For the sake of simplicity, we present in Alg. 5 their countermeasure applied to the particular case of AES-128, where `RandomBit`() denotes a function returning a random bit and `SNLF` represents a nonlinear function where $\texttt{SNLF}(0) = 0$. The $\texttt{RoundFunction}(S, k)$ is defined as the composition of the four AES transformations SubBytes, ShiftRows, MixColumns and AddRoundKey applied to state S and round key k. Let us also denote β and k_0 such that $\texttt{RoundFunction}(\beta, k_0) = \beta$. In order to keep the SPA resistance of the scheme, the round-key addition at the beginning of the algorithm is defined as a specific `RoundFunction` which is composed of dummy SubBytes, ShiftRows and MixColumns transformations. The same method applies for the last round which is defined as a specific `RoundFunction` with a dummy MixColumns transformation.

The principle of Alg. 5 is to perform each round twice. First a redundant round computation is done then the effective round is performed. These two rounds results are stored in intermediate states S_1 and S_0 respectively. Any difference between effective and redundant rounds infect effective state S_0 through Steps 9 and 11. Furthermore each time `RandomBit`() outputs 0, a dummy round is executed and its result is stored in S_2. Whenever dummy round result is not equal to β then Steps 8 and 10 infect subsequent computations. One may note that one last dummy round is always executed in the last step 14 of the algorithm to spread possible infections.

In [88], the authors provide a security analysis that takes into account only classical literature attacks. We show that this analysis is not sufficient and that their countermeasure is flawed when applied to the AES algorithm.

2.4.2 Attack Description

For the sake of simplicity, we applied the countermeasure to AES-128, however the same attack applies on other key sizes. As a general observation about the proposed countermeasure, we remark that the last while loop (i.e. $i = 22$) always performs the

[1]The ePrint version of the paper [89] corrects some typographic error of the original version. However, we notice that this updated version is not correct in the case of AES. We present in Alg. 5 the updated version of their method.

Algorithm 5: LatinCrypt 2012 Countermeasure applied on AES-128

Inputs : P, roundkeys $k_j, j \in \{1, \ldots, 11\}$, derived from AES key K, β, k_0
Output: $C = \text{AES-128}(P, K)$

1 State $S_0 \leftarrow P$; Redundant state $S_1 \leftarrow P$; Dummy state $S_2 \leftarrow \beta$
2 $R_0 \leftarrow 0$; $R_1 \leftarrow 0$; $R_2 \leftarrow \beta$; $i \leftarrow 1$
3 **while** $i \leq 22$ **do**
4 $\quad r \leftarrow \texttt{RandomBit}$
5 $\quad a \leftarrow (i \wedge r) \oplus 2(\neg r)$
6 $\quad b \leftarrow \lceil i/2 \rceil r$
7 $\quad S_a \leftarrow \texttt{RoundFunction}(S_a, k_b)$
8 $\quad R_a \leftarrow S_a \oplus R_2 \oplus \beta$
9 $\quad e \leftarrow r(\neg(i \wedge 1)) \cdot \texttt{SNLF}(R_0 \oplus R_1)$
10 $\quad S_2 \leftarrow S_2 \oplus e$
11 $\quad S_0 \leftarrow S_0 \oplus e$
12 $\quad i \leftarrow i + r$
13 **end**
14 $S_0 \leftarrow S_0 \oplus \texttt{RoundFunction}(S_2, k_0) \oplus \beta$
15 **return** (S_0)

following steps:

7 $S_0 \leftarrow \texttt{RoundFunction}(S_0, k_{11})$
8 $R_0 \leftarrow S_0 \oplus R_2 \oplus \beta$
9 $\ldots$
10 $S_2 \leftarrow S_2 \oplus \texttt{SNLF}(R_0 \oplus R_1)$
11 $S_0 \leftarrow S_0 \oplus \texttt{SNLF}(R_0 \oplus R_1)$

This last loop thus represents the execution of the last effective round of the block cipher. Therefore the only round that is executed after the last effective one is the dummy round of Step 14. We thus noticed that if the last effective round is disturbed then the infection is limited to the efficiency of 1 dummy round.

Let us assume that the attacker is able to inject an error ε on the input of the last effective round (i.e. on S_0 at the beginning of Step 7 of Alg. 5 when $i = 22$). For the sake of simplicity, we assume that the fault disturbs the first byte s of the second row of S_0 (i.e. $s = S_{0_1}$), but the attack can be easily extended to any byte of the 3 last rows of the state. The differential between the faulty and the redundant intermediate states after Step 7 at loop $i = 22$ is thus:

$$\widetilde{S}_0 \oplus S_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

where $\alpha = SubByte(s \oplus \varepsilon) \oplus SubByte(s)$. The same differential is reflected in temporary states differential $\widetilde{R}_0 \oplus R_1$ due to Step 8. Once applied, the nonlinear transformation of Step 9 does not change the shape of the differential, apart for transforming the faulted value α into $\texttt{SNLF}(\alpha)$. Steps 10 and 11 then spread this infection to the dummy and

effective states, obtaining in particular after Step 11:

$$\widetilde{S}_0 \oplus S_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha \oplus \texttt{SNLF}(\alpha) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

As stated above, only the dummy round of Step 14 is executed afterwards in which the faulty dummy state $\widetilde{S}_2$ re-infects the output. However, one can observe that the ShiftRows transformation of the dummy round moves the infection to column 3 thus not masking column 4. By denoting $\widetilde{\beta} = \texttt{RoundFunction}(\widetilde{S}_2, k_0)$ at Step 14, the resulting differential between a correct and a faulty output is thus:

$$C \oplus \widetilde{C} = \begin{pmatrix} 0 & 0 & (\beta \oplus \widetilde{\beta})_8 & 0 \\ 0 & 0 & (\beta \oplus \widetilde{\beta})_9 & \alpha \oplus \texttt{SNLF}(\alpha) \\ 0 & 0 & (\beta \oplus \widetilde{\beta})_{10} & 0 \\ 0 & 0 & (\beta \oplus \widetilde{\beta})_{11} & 0 \end{pmatrix}.$$

From this relation, we explain hereafter how the byte 13 of the last round key can be retrieved.

From the correct ciphertext C, for each key byte hypothesis $k_h \in \{0, ..., 255\}$ she computes the corresponding last round input byte:

$$s_h = SubByte^{-1}(C_{13} \oplus k_h),$$

and for each error hypothesis $\varepsilon_h \in \{1, ..., 255\}$ she computes $\alpha_h = SubByte(s_h \oplus \varepsilon_h) \oplus SubByte(s_h)$ and tests if the following equality holds:

$$\alpha_h \oplus \texttt{SNLF}(\alpha_h) = (C \oplus \widetilde{C})_{13} \quad .$$

With one pair of correct/faulty ciphertexts the attacker reduces the space of possible key and error byte values from $2^8(2^8-1)$ to 2^9 on average. By using a constant unknown byte fault model (i.e. the same unknown error is assumed to occur repeatedly during different executions), the intersection of the results of the attack on three pairs of correct/faulty ciphertexts is likely to reveal the value of the corresponding key byte with high probability. By iterating the attack on the other bytes of the three lower rows of the last round state and with a 4-byte brute-force she can retrieve the full AES-128 key.

The random execution of dummy rounds can still disturb the attacker as she won't be able to always target the last round of the effective computation. However one may note that for a fault injected before the last round or in a dummy round, the differential $\widetilde{C} \oplus C$ will be nonzero in more indexes than those expected, she can thus discard the corresponding faulty output. Furthermore, the same observation allows the attacker to deduce which byte was faulted, thus she does not need control on fault localization.

2.4.3 Simulations

The attack was simulated by injecting a constant unknown byte fault on the last three rows of the input of round 10. For our simulations we set the SNLF to be the byte-wise applied AES-inversion as suggested in [88]. We computed over 100 000 experiments the success rate to recover the corresponding last three rows of the last round key, the 4

missing bytes being then recovered by performing a fast exhaustive search. Simulations were performed in three different contexts depending on the attacker's ability to inject the same constant fault. Namely, we performed simulations with a probability of injecting the same error of 100%, 90% and 75% respectively. The corresponding results are depicted in Figure 2.2.

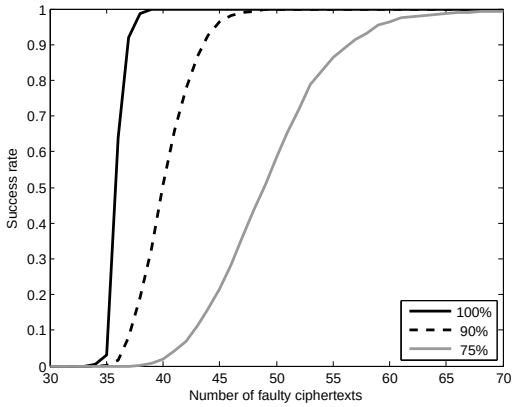


Figure 2.2 – Success rate to recover 12 bytes of the last round key vs number of faulted ciphertexts depending on the attacker's capability to induce a constant error.

2.4.4 Conclusion

This section described a fault attack on the symmetric infective countermeasure based on redundant and dummy rounds suggested in [88, 89]. We observe that if a fault is induced during the last round of the algorithm, the infection of the error is limited. This section shows that by exploiting this weakness one can retrieve the full AES-128 key with 36 faults on average.

Sec. 2.5 discusses an improvement of the infective countermeasure that thwarts the attack described in this section. Our discussion will focus on the security of such new version of the countermeasure and on the difficulty to correctly implement it in the presence of fault attacks. This is a joint work with Christophe Giraud.

2.5 Analysis of the CHES 2014 AES Infective Countermeasure

At CHES 2014 an interesting evolution of the symmetric infective countermeasure of [88] was presented. In [165] Tsupamudre *et al.* presented some improvements of the attacks discovered in [15] and suggested modifications of the original infective countermeasure in order to withstand the attacks found. In this section we study the proposition of [165] and show several flaws in its design. We suggest four different attack paths that allow the attacker to bypass the infective countermeasure by using three different fault models.

2.5.1 Countermeasure Presentation

The infective countermeasure suggested at CHES 2014 by Tupsamudre *et al.* [165] is based on the work presented at LatinCrypt 2012 by Gierlichs *et al.* [88] and it is meant to resist the attack presented in Sec. 2.4. For the sake of simplicity we recall in Alg. 6 the CHES 2014 countermeasure suggested in [165] applied to AES-128. For more information about AES, the reader can refer to [81].

Algorithm 6: CHES 2014 Countermeasure applied on AES-128

Inputs : Plaintext P, round keys k^j for $j \in \{1, \ldots, 11\}$, pair (β, k^0), security level $t \geq 22$
Output: Ciphertext $C = \text{AES-128}(P,K)$

State $R_0 \leftarrow P$; Redundant state $R_1 \leftarrow P$; Dummy state $R_2 \leftarrow \beta$
$i \leftarrow 1$, $q \leftarrow 1$
$rstr \leftarrow \{0,1\}^t$ // $\#1(rstr) = 22, \#0(rstr) = t - 22$
while $q \leq t$ **do**
  $\lambda \leftarrow rstr[q]$ // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow (i \wedge \lambda) \oplus 2(\neg\lambda)$
  $\zeta \leftarrow \lambda \cdot \lceil i/2 \rceil$ // ζ is actual round counter, // $\zeta = 0$ is for dummy round
  $R_\kappa \leftarrow \texttt{RoundFunction}(R_\kappa, k^\zeta)$
  $\gamma \leftarrow \lambda(\neg(i \wedge 1)) \cdot \texttt{BLFN}(R_0 \oplus R_1)$
  $\delta \leftarrow (\neg\lambda) \cdot \texttt{BLFN}(R_2 \oplus \beta)$
  $R_0 \leftarrow (\neg(\gamma \vee \delta) \cdot R_0) \oplus ((\gamma \vee \delta) \cdot R_2)$
  $i \leftarrow i + \lambda$
  $q \leftarrow q + 1$
end
return (R_0)

Alg. 6 uses three states denoted R_0, R_1 and R_2 for the cipher, the redundant and the dummy rounds respectively. The execution order of these rounds is given by a random bit string $rstr$ generated at the beginning of the algorithm. Each "0" on the string encodes a dummy round, while a "1" encodes a redundant or cipher round. Each time a "1" occurs, an index i is incremented and a redundant round (resp. a cipher round) is executed if i is odd (resp. even). Alg. 6 thus executes a loop over the $rstr$ string bits and executes a cipher, redundant or dummy round accordingly. One may note that Alg. 6 computes the redundant round before the cipher round all along the algorithm and dummy rounds can happen randomly at any time.

Dummy rounds are executed over a dummy state R_2 which is initialized to a random 128-bit value β and by using the round key k^0 which is computed such that:

$$\texttt{RoundFunction}(\beta, k^0) = \beta \ . \tag{2.5}$$

The number of dummy rounds is parameterized by a security level t chosen by the developer. More precisely, this parameter represents the whole number of cipher, redundant and dummy rounds performed during Alg. 6 execution. For instance in the case of AES-128, $t - 22$ dummy rounds will be performed.

Regarding the infective part, a first infection is activated after each cipher round if its state R_0 is different from the redundant state R_1 (Steps 9 and 11). Moreover another infection occurs if $R_2 \neq \beta$ after the execution of a dummy round (Steps 10 and

11). These infections consist in replacing the cipher state R_0 with the random value R_2, leaving no chance to the attacker to obtain information on the secret key once the infection is applied. To do so, a Boolean function `BLFN` is used which maps non-zero 128-bit values to 1 and outputs 0 for a null input.

Compared to the original LatinCrypt 2012 proposal, the CHES 2014 infective countermeasure differs by the way of dealing with the sequence of cipher, redundant and dummy rounds which is now done by using a random string $rstr$ and by the way the infection is performed which is now fully random.

Despite the security analysis of Alg. 6 presented in [165], we show in the next section that it may be insecure if implemented as such. Furthermore we show that an attacker can recover the full secret key for each of the three most popular fault models used in literature.

2.5.2 Attacks

In this section we firstly present the principle of our attacks which are based on the fact that the variables dealing with the number of rounds to perform are not protected. We then exploit this remark to suggest four different attacks that use different fault models such as the instruction skip, the stuck-at and the random error fault model.

Principle of Our Attacks

Due to the improvements of Alg. 6 compared with the original LatinCrypt 2012 countermeasure, it is impossible for the attacker to obtain any information on the secret key once the infection has occurred. In order to thwart this countermeasure, we thus investigate the possibility to disturb the number of executed rounds since the corresponding variable is not protected in integrity. Indeed, if the attacker succeeds in disturbing the number of rounds she may be able to retrieve the secret key from the corresponding faulty ciphertext [48, 72]. In the remainder of this section, we show how such an attack works if the last round of an AES-128 has been skipped.

If the attacker knows a correct and a faulty ciphertext obtained by skipping the last AES round then it is equivalent to know the input S^{10} and the output S^{11} of the last round. Due to the lack of *MixColumns* transformation during the last AES round, the last round key k^{11} can be recovered byte per byte by XORing the corresponding bytes of S^{10} and S^{11}:

$$k_i^{11} = S_i^{11} \oplus \texttt{SBox}(S^{10}_{\mathrm{SR}^{-1}(i)}) \ , \ \ \forall i \in \{1, \ldots, 16\}, \tag{2.6}$$

where SR corresponds to the byte index permutation induced by the transformation *ShiftRows*. In such a case, the attacker can recover the full AES-128 key from only one pair of correct and faulty ciphertexts.

One may note that this attack works similarly if the attacker knows the input and the output of the first round. For further details on the first round attack the reader can refer to [48].

In the following, we describe different ways of disturbing Alg. 6 by using several fault models such that it does not perform the AES with the correct number of rounds whereas no infection is performed. In our description we make use of AES-128 as a reference, however our attacks can apply straightforwardly to other key sizes.

Attack 1 by Using Instruction Skip Fault Model

The attack presented in this section exploits the fact that whenever the variable i is odd and $\lambda = 1$ then a redundant round is executed and that this kind of round does not involve any infection. To exploit this remark, we assume that the attacker is able to skip an instruction of its choice by means of a fault injection.

Description If the attacker skips Step 12 of Alg. 6 after the last redundant round then the increment of i is not performed. Therefore i stays odd so the last cipher round is replaced by another redundant round. As no infection is involved for redundant rounds, the algorithm returns the output of the penultimate round. The attacker can thus take advantage of such an output to recover the secret key as explained in Sec. 2.5.2.

Efficiency As explained in Sec. A.2, the probability of skipping the last cipher round and thus to recover the AES key after disturbing r different AES executions by skipping Step 12 during the q-th loop is given by:

$$\mathcal{P}_r = 1 - \left(1 - \frac{\binom{q-1}{20}\binom{t-q}{1}}{\binom{t}{22}}\right)^r \quad . \tag{2.7}$$

where t is the total number of rounds performed during Algorithm 6, i.e. the number of while loops.

Some numerical values of (2.7) are given in Table 2.2 for t equal to 30, 40 and 50, $q = t-3, \cdots, t-1$ and $r = 1, \cdots, 4$. One can notice that if the fault is injected when q equals t then the attack does not work because all the rounds have already been executed.

t	q	Number r of faults			
		1	2	3	4
30	27	11.80%	22.21%	31.39%	39.49%
	28	30.34%	51.48%	66.20%	76.46%
	29	53.10%	78.01%	89.69%	95.16%
40	37	19.34%	34.93%	47.52%	57.66%
	38	28.06%	48.24%	62.76%	73.21%
	39	29.62%	50.46%	65.13%	75.46%
50	47	18.96%	34.32%	46.77%	56.86%
	48	22.00%	39.16%	52.54%	62.98%
	49	18.86%	34.16%	46.57%	56.65%

Table 2.2 – Probability of obtaining a useful faulty ciphertext by skipping Step 12 during the q-th loop of Alg. 6.

By analyzing Table 2.2, one can deduce the best strategy for the attacker. For example if $t = 30$ then the attacker should target the 29-th loop in order to obtain the best chances of retrieving the key with the minimal number of fault injections.

Experiments The attack described in this section has been simulated for $t = 30$ and for each q between 25 and 29. The experiment has been repeated 3 000 times for each configuration. The results of our tests are depicted in Figure 2.3.

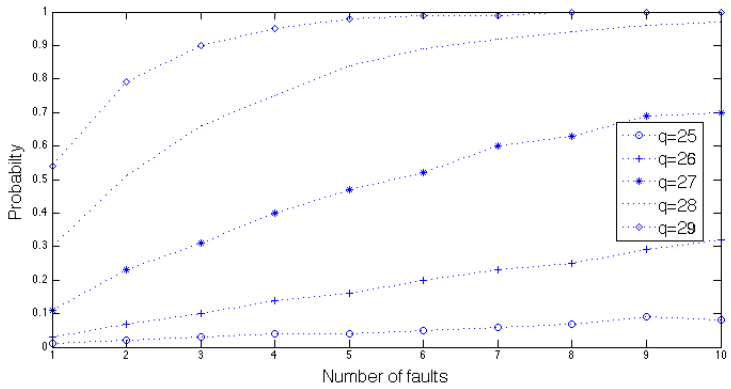


Figure 2.3 – Experimental probability of obtaining a useful faulty ciphertext by skipping Step 12 during the q-th loop of Alg. 6 for $t = 30$.

By comparing Figure 2.3 and the row $t = 30$ of Table 2.2, one can notice that the experiments perfectly match with the theoretical results.

Attack 2 by Using Stuck-at 0 Fault Model

In this section we use the *stuck-at 0* fault model where we assume that the attacker can set to zero a variable of her choice. As for the attack presented in Sec. 2.5.2, the goal of the attacker is to skip the execution of the last cipher round.

Description To avoid the execution of the last cipher round by using a stuck-at 0 fault model without activating an infection, the attacker can set to zero the variable λ right after Step 5 during the loop involving the last "1" of $rstr$, i.e. during the loop dealing with the last cipher round. The computation of the last cipher round is thus skipped since $\lambda = 0$ implies a dummy round. The attacker thus retrieves an exploitable faulty ciphertext that can be used to retrieve the secret key as described in Sec. 2.5.2. As no consistency check is performed on λ, $rstr$ nor on the number of cipher rounds executed, Alg. 6 does not detect the fault.

Efficiency We detail in Sec. A.2 the reasoning to compute the probability of obtaining at least one useful faulty ciphertext after disturbing r different AES executions by setting λ to 0 after Step 5 of the q-th loop. Such a probability is given by:

$$\mathcal{P}_r = 1 - \left(1 - \frac{\binom{q-1}{21}}{\binom{t}{22}}\right)^r . \tag{2.8}$$

Some numerical values of (2.8) are given in Table 2.3 for t equal to 30, 40 and 50, $q = t - 2, \cdots, t$ and $r = 1, \cdots, 4$.

By comparing Table 2.3 with Table 2.2, one may note that the attack presented in this section is more efficient than the one presented in Sec. 2.5.2, especially when the attacker targets the last loop execution.

Experiments We simulated the attack for $t = 30$ and for q from 27 to 30. For each value of q we performed 3 000 tests with random $rstr$. The results of such experiments are depicted in Figure 2.4.

t	q	Number r of faults			
		1	2	3	4
30	28	5.06%	9.86%	14.42%	18.75%
	29	20.23%	36.37%	49.24%	59.51%
	30	73.33%	92.89%	98.10%	99.49%
40	38	11.36%	21.42%	30.35%	38.26%
	39	25.38%	44.33%	58.46%	69.00%
	40	55.00%	79.75%	90.89%	95.90%
50	48	14.14%	26.29%	36.71%	45.66%
	49	25.14%	43.96%	58.05%	68.60%
	50	44.00%	68.64%	82.44%	90.17%

Table 2.3 – Probability of obtaining a useful faulty ciphertext by sticking λ at 0 during the q-th loop of Alg. 6.

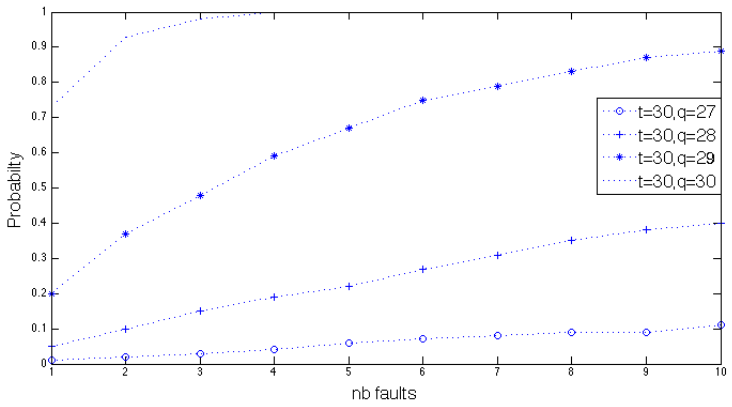


Figure 2.4 – Experimental probability of obtaining a useful faulty ciphertext by sticking λ at 0 during the q-th loop of Alg. 6 for $t = 30$.

Attack 3 by Using Random Error Fault Model

We show in this section how the attacker can use the random error fault model to obtain a useful faulty ciphertext. In this fault model, we assume that the attacker can change the value of a chosen internal variable into a random value.

Description Due to its central role in the infection and scheduling, string $rstr$ is very sensitive. However, the authors of [165] do not suggest any mean of ensuring its integrity. We thus investigated this path and we noticed that an attacker can disturb the generation of $rstr$ at Step 3 of Alg. 6 such that it does not contain 22 "1" anymore. If the fault disturbs the string $rstr$ such that it contains only 21 (resp. 20) "1" then Alg. 6 does not execute the last cipher round (resp. the last redundant and cipher rounds). In both cases no infection is performed allowing the attacker to exploit the corresponding faulty ciphertext to recover the secret key as explained in Sec. 2.5.2.

Efficiency The probability to obtain at least one useful faulty ciphertext after disturbing r different AES executions by randomly modifying the least significant byte of $rstr$ during Step 3 is given by:

$$\mathcal{P}_r = 1 - \left(1 - \left(\sum_{i=1}^{8} \frac{\binom{t-8}{22-i}\binom{8}{i}}{\binom{t}{22}} \sum_{j=1}^{i} \frac{\binom{i}{j}\binom{8-i}{j-1}}{255} + \sum_{i=2}^{8} \frac{\binom{t-8}{22-i}\binom{8}{i}}{\binom{t}{22}} \sum_{j=2}^{i} \frac{\binom{i}{j}\binom{8-i}{j-2}}{255}\right)\right)^r . \quad (2.9)$$

For more details about the computation of this probability, the reader can refer to Sec. A.2.

Table 2.4 gives the probability to obtain a useful faulty ciphertext for t equal to 30, 40 and 50.

t	Number r of faults			
	1	2	3	4
30	41.63%	65.93%	80.11%	88.39%
40	34.72%	57.39%	72.18%	81.84%
50	24.60%	43.15%	57.13%	67.67%

Table 2.4 – Probability of obtaining a useful faulty ciphertext by disturbing Step 3 of Alg. 6.

Experiments Figure 2.5 shows the results obtained by simulating the attack described above. The simulations have been performed by generating a random string *rstr* and disturbing it with an 8-bit random error. The test has been performed 3 000 times for each t equal to 30, 40 and 50.

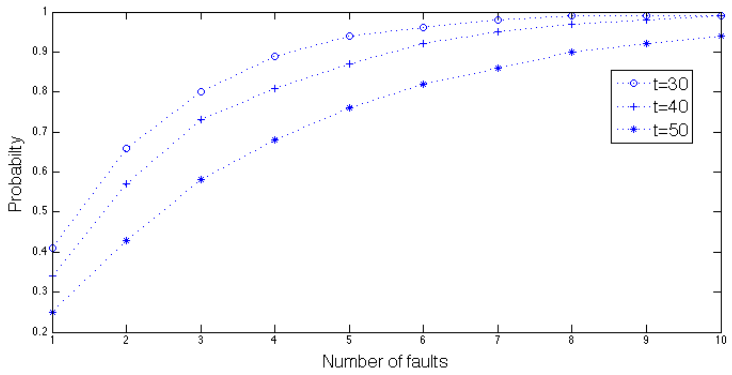


Figure 2.5 – Experimental probability of obtaining a useful faulty ciphertext by disturbing Step 3 of Alg. 6.

Attack 4 by Using Random Error Fault Model

This section describes a second attack that can be mounted by using the random error fault model.

Description The idea of the attack is to disturb the increment of index q at Step 13 of Alg. 6 during the execution of the first cipher round. We noticed that if the disturbance produces an error e such that $q \oplus e > t$ then the evaluation at Step 4 is false and the algorithm returns. If the algorithm computes only one cipher round then the attacker can use such an output to retrieve the first round key, cf. [48]. It is important to notice that in order to retrieve a useful output, the attacker needs to disturb the execution during the first cipher round and not after a redundant or dummy round.

Efficiency As detailed in Sec. A.2, the probability to obtain at least one useful faulty ciphertext after disturbing r different AES executions by injecting a random error during Step 13 of the q-th loop is given by:

$$\mathcal{P}_r = 1 - \left(1 - \frac{2^8 - t}{2^8} \sum_{i=2}^{3} \frac{\binom{q}{i}\binom{t-q}{22-i}}{\binom{t}{22}}\right)^r . \tag{2.10}$$

We give in Table 2.5 the probability that the attacker retrieves a useful faulty ciphertext for t equal to 30, 40 and 50 and for q from 2 to 4.

t	q	Number of faults			
		1	2	3	4
30	2	46.88%	71.78%	85.01%	92.04%
	3	73.67%	93.07%	98.17%	99.52%
	4	60.52%	84.42%	93.85%	97.57%
40	2	24.99%	43.73%	57.79%	68.34%
	3	48.66%	73.64%	86.47%	93.05%
	4	58.22%	82.55%	92.71%	96.95%
50	2	15.17%	28.05%	38.96%	48.23%
	3	32.88%	54.95%	69.76%	79.70%
	4	45.58%	70.38%	83.88%	91.23%

Table 2.5 – Probability of obtaining a useful faulty ciphertext by injecting a random error fault on Step 13 of Alg. 6.

The attacker can use Table 2.5 to choose the best strategy for her attack. For example for $t = 30$, one obtains the best chances to retrieve a useful faulty ciphertext by attacking the third loop. Furthermore when comparing the efficiency of our four attacks, the attack presented in this section is the most efficient one.

Experiments We mounted several simulations where we disturbed the Step 13 of the q-th loop with a random byte error e. We mounted the experiments for $t = 30$ and for q from 2 to 6. For each different q we repeated the experiment 3 000 times. The results of such experiments are shown in Figure 2.6.

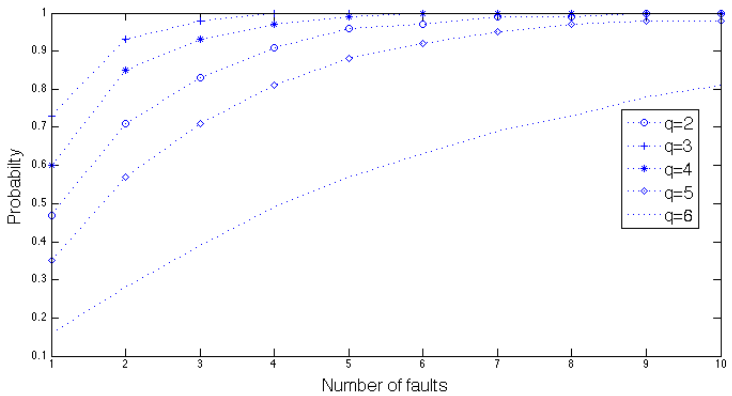


Figure 2.6 – Experimental probability of obtaining a useful faulty ciphertext by a injecting random error fault on Step 13 of Alg. 6 for $t = 30$.

The simulations shows that this attack has a remarkable success rate. For example for $t = 30$, an attacker that reiterates the fault injection only twice during the third loop has a probability of retrieving a useful faulty ciphertext greater than 90%.

2.5.3 Conclusion

This section presents a security analysis of the infective countermeasure of CHES 2014. Our attack targets variables and instructions that have been explicitly added to the original countermeasure in order to thwart fault attacks. While the new infective countermeasure gives no information to the attacker once the infection is applied, we discovered that it does not protect the number of cipher rounds effectively executed. By using this remark we applied the three most popular fault models and found four different attack paths that allow an attacker to recover the secret key of the underlying cryptosystem. For each attack we studied the success probability and performed simulations that validated our theoretical results.

With this work we also remark that the lack of formal security proofs in this field is clearly an issue. We hope that new ideas may pave the way to formally prove the security of cryptosystems against fault-based cryptanalysis.

In the next section we change the target of our analysis. We abandon the infective countermeasure and analyze generic ECC implementations countermeasures against faults. The results of this work were published at the COSADE 2014 workshop.

2.6 Common Points Attack on ECC

In this section we present a fault attack against elliptic curve cryptosystems that exploits particular points presents on some elliptic curves. We start by introducing the concept of common points, afterwards we show how to use them in a fault attack against elliptic curve cryptosystems to retrieve sensitive values. The basic idea of our attack is to input a point that lies on a family of curves that includes a weak one (i.e. a smooth one). The attacker then forces the computation on the weak curve by means of a fault injection. She can then use the output to solve the ECDLP and retrieve the secret. The advantage in using common points is that they allow the faulty values to successfully pass the point on curve test even if a parameter has been faulted.

2.6.1 Common Points

Definition 88 (Common Point)**.** *Let $\mathcal{F}$ be a family of elliptic curves, a point P is a* common point *for the family of curves $\mathcal{F}$, iff $P \in \mathcal{E}, \forall\, \mathcal{E} \in \mathcal{F}$.*

Example 3 (Common Point)**.** *Consider for example Eq. 1.1 over a field $\mathbb{K}$ with $Char(\mathbb{K}) \neq 2, 3$, and let $b = g^2 \in \mathbb{K}$:*

$$y^2 = x^3 + ax + g^2 \tag{2.11}$$

The family of curves obtained by varying parameter $a \in \mathbb{K}$ has two common points, namely the points $(0, \pm g)$ satisfy the curve equation for any a.

Among the whole family described in Ex. 3, most curves have composite order. Furthermore some of them have smooth order, allowing an attacker to solve the ECDLP

in a reasonable time. Common points are thus contained in any curve whose b is a quadratic residue for the underlying field. Among the curves proposed in FIPS 186-4 [80], only P-224 does not, and four out of eight curves proposed in SECG [163] can be attacked. In the following we present our new fault attack that makes use of these points.

2.6.2 Fault Attack Using Common Points

In order to apply our attack it is necessary for the attacker to be able to choose the input point of the scalar multiplication and to obtain the corresponding result. The reader will notice that these assumptions are weaker than the ones of [74, 93], as we need no leakage hypothesis nor a particular fault model.
Our attack is divided in four steps as detailed below:

- Step 1: the attacker sends a point $P = (0, \pm g)$ to the embedded device, and disturbs the curve parameter a into $\widetilde{a}$ at the beginning of the scalar multiplication.
- Step 2: the device performs the scalar multiplication on the faulted curve $\mathcal{E}'$: $y^2 = x^3 + \widetilde{a} \cdot x + g^2$, and return the value $\widetilde{Q} = [d]P \in \mathcal{E}'$.
- Step 3: From $\widetilde{Q} = (x_q, y_q)$, the attacker can recover the value $\widetilde{a}$ by solving the equation $y_q^2 = x_q^3 + \widetilde{a} \cdot x_q + g^2$.
- Step 4: the attacker can then solve the ECDLP on the curve $\mathcal{E}'$, which will work if the order o of $\widetilde{Q}$ is smooth.

We remark that in Step 1 the attacker does not need a particular fault model to perform the attack. Furthermore, when the order of $\mathcal{E}'$ is smooth, our attack does involve a single fault, which means that a single, successful execution is sufficient to completely break the cryptosystem. This property is particularly useful for two reasons. First of all our attack does not suffer from the fact that the secret scalar may be freshly regenerated at each execution. Furthermore it works in environments where the number of executions is limited, while other attacks that need to be reiterated a considerable number of times like [32, 75] may be thwarted.

The simulations carried out in Sec. 2.6.3 show that the order of $\mathcal{E}'$ is smooth 45% of the time. We show in the following that the other 55% of the results also provide very useful information on the secret scalar.

Attack Application on Not-so-Smooth Orders

When the order o of $\widetilde{Q}$ is not smooth enough to use square root attacks [139] for instance, the attacker can still gain information on the secret scalar value. Let us assume that the secret scalar is fixed for multiple executions. For an order o that contains a big factor of ℓ bits, the attacker can use Pohlig-Hellman decomposition [116] and retrieve the secret scalar modulo small factors of size $\log(o) - \ell$ bits on each execution. Thus by performing the attack on $\frac{\log(o)}{\log(o)-\ell}$ executions on average, she can accumulate the information thanks to the Chinese Remainder Theorem and completely reveal the secret scalar.

Comparison with Existing Attacks

In the following, we compare the efficiency of our attack with previously published ones [23, 32, 49, 74]. The attack of [49] uses a fault similar to the one we use to solve the ECDLP, however it is thwarted by simply testing if the point is on the curve after the scalar multiplication. As the authors explain in their work, by faulting parameter a into $\widetilde{a}$, the input point P does not lie on the curve $\widetilde{\mathcal{E}}$ given by the equation:

$$\widetilde{\mathcal{E}} : y^2 = x^3 + \widetilde{a} \cdot x + b$$

There exists instead a value b' such that P lies on the curve $\widetilde{\mathcal{E}}'$ defined by:

$$\widetilde{\mathcal{E}}' : y^2 = x^3 + \widetilde{a} \cdot x + b'$$

As the curve operation formulae do not involve curve parameter b, the scalar multiplication is in fact done on $\widetilde{\mathcal{E}}'$. At the end of the multiplication, the result point will also lie on $\widetilde{\mathcal{E}}'$, and thus will not satisfy the equation of curve $\widetilde{\mathcal{E}}$. As b has not been modified, by testing if the result point lies on the curve, the attack is thwarted. On the contrary, due to the use of a common point, our attack is not detected by this countermeasure. Indeed the common point P satisfies the curve equation for any a. In particular the input point lies on the curve $\widetilde{\mathcal{E}}$, thus the scalar multiplication is performed on it. This implies that the result of the multiplication $\widetilde{Q}$ lies on the same curve. Thus our attack is not detected by testing if $\widetilde{P}$ or $\widetilde{Q}$ satisfies the equation for $\widetilde{\mathcal{E}}$ if a is not reloaded before the final point on curve test.

The fault attacks of [23, 32] need the faulted result of several executions to retrieve the secret scalar value. Hence, implementations using random scalar nonces, or with a limited number of executions may not be vulnerable to them. The point on curve countermeasure suggested by [23] does not detect our attack as explained above, and the verification countermeasure suggested in [32] is not effective as we do not disturb the scalar multiplication.

The authors of [74] use a fault to drop the regular input point into a weaker curve (characterized by a wrong b') where the faulted point will leak information by SSCA. However they need a strong hypothesis on the fault model and on the leakage for the attack to work. As a common point lies on all curves with the same b, the fault model required by our attack is very relaxed. Indeed, any random fault on the parameter a provides information on the secret scalar. We insist on the fact that our attack needs no SCA hypothesis, which make it work on implementations that do not leak information if the point at infinity is handled during the multiplication, contrary to the attack presented in [74].

Attack Scenario

Unfortunately, to the best of our knowledge the attack scenario that we describe is not applicable to any protocol. However our context is similar to the one of [23] Section 4.1 or [82] Section 3 for example, where the attacker can choose the input point P, and get the output Q of the scalar multiplication. While no actual protocol is concerned, we stress that embedded cryptographic developers should choose their countermeasures in order to avoid the attack that we present here, and new cryptographic protocols should be conceived in order to avoid falling into our attack scenario.

2.6.3 Simulations

In order to validate the efficiency of our attack, simulations have been performed on Pari/gp software [136] by using the standard elliptic curve P-192 proposed in FIPS 186-4 [80].

The attack was mounted by using input point $P = (0, \sqrt{b})$. Afterwards, one byte of the value a was disturbed with an error before the multiplication. For each byte j of a, and for each value e between 1 and $2^8 - 1$, the error was simulated by assigning $a \leftarrow a \oplus 2^{8*j}e$. The scalar multiplication $\widetilde{Q} = [k]P$ was computed for a random value k and the faulty output point $\widetilde{Q}$ was returned. Afterwards for each output point the cardinality o of the new curve was computed by using the Pari/gp implementation of the SEA algorithm [155].

In order to study the probability of revealing the whole secret k with a single fault, the size of the order o was collected for all faulted executions. At most $\log_2(o)$ bits of information can be obtained on the value k for a point $\widetilde{Q}$ resulting from a faulted scalar multiplication. Thus the secret k can be fully revealed only if the bit-size of o and k are similar. Figure 2.7 shows the occurrence probability of the different order sizes obtained during the campaign. From the results it is clear that once the value of k is retrieved with one application of our attack, the remaining secret bits can be obtained by brute force since at most 18 bits are missing.

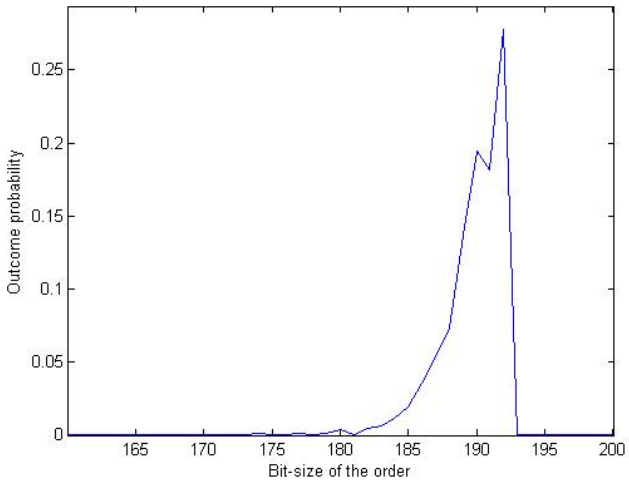


Figure 2.7 – Outcome probability for each bit-size of the result point order.

The second analysis concerns the probability that the resulting ECDLP is smooth enough to be solved by modern computers. For each result the cardinality of the faulted curve is factored to extract the bit-size of its biggest prime factor. Figure 2.8 displays the probability that the faulted curve cardinality contains a biggest factor of a certain bit-size. To better understand this result in Figure 2.9 is plotted the cumulative outcome probability of the previous distribution. In other words, Figure 2.9 shows the probability that the biggest factor of the faulted curve cardinality is smaller than a certain bit-size. We assume that the computation limits of modern parallel Pollard's rho implementations are bound to 112 bits complexity [36], thus as "smooth" orders we consider orders whose biggest prime factor is smaller than 112 bits. From the experiments it can be observed that the probability to obtain sufficiently small sizes for the biggest factor exceeds 45%.

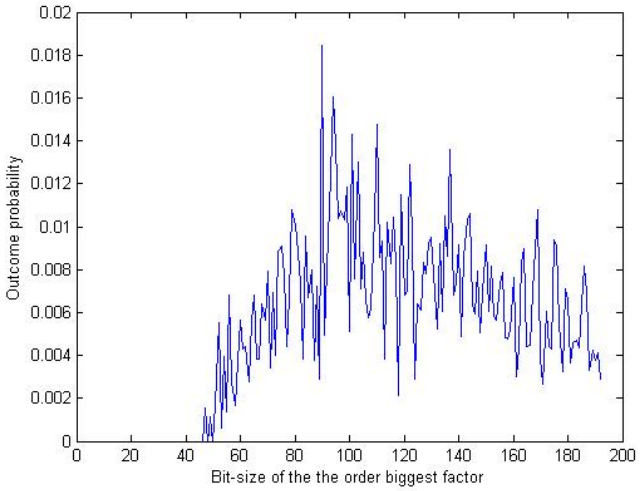


Figure 2.8 – Outcome probability for the bit-size of each result point order greatest factor.

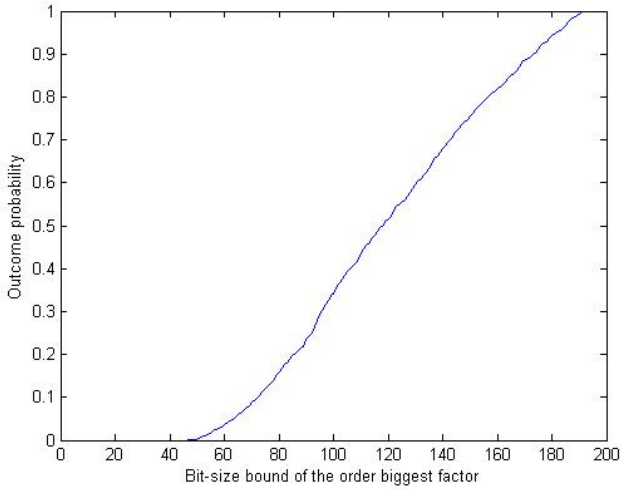


Figure 2.9 – Cumulative outcome probability for each point order biggest factor bit-size.

Finally, in order to obtain information on the time required to solve the logarithm, the attack was mounted on faults that produced a smooth order result. It has been chosen to mount the ECDLP on results whose order biggest factor was 42, 52, and 62 bits. These sizes have been chosen starting by the smallest obtained, in order to speed up the computation time to collect more results. The ECDLP was solved by using Pari/gp library function *elllog* without optimization. The computation times were evaluated on a 3 .10GHz personal computer with 8GB RAM. The average time was computed over 500 executions for the 42-bit ECDLP, over 205 for the 52-bit, and over 4 for the 62-bit ECDLP. For random values of the secret key k the results are shown in Table 2.6, for example for a curve whose order biggest factor is 42 bits, the required time is 53 seconds on average. Thus it is clear that our attack is definitely practical on smooth order curves.

Table 2.6 – Average time to solve the ECDLP for different bit-sizes of the biggest factor of the order of the point by using NIST P-192 as original curve.

Biggest factor bit-size	#Tests	Avg time
42	500	53 sec
52	205	20.6 min
62	4	19.21 hours

2.6.4 Countermeasures

Three conditions are necessary to mount the attack described in this section, namely to control the input point, to know the result of the scalar multiplication, and to have a quadratic residue as parameter b of the curve. In the following, various state-of-the-art countermeasures are analyzed. In addition to standard FA countermeasures we also focus on an SCA countermeasure which provides a good level of resistance against attacks. Finally, Table 2.7 resumes the results of the countermeasure analysis provided in this section.

Initial and Final Checks

Intuitively, initial and final parameters checking could thwart the attack described here , as the curve parameters are disturbed, and thus no more consistent. As already remarked in [74], the initial checks are ineffective if the fault is injected after the countermeasure application. However for our attack to work, the adversary needs to retrieve the output, which is not returned if the final checks fail. The authors of [49], suggest that parameters checking can be performed by means of integrity check or by point on curve testing, and that the two mechanisms offer the same security level. This work shows that this claim is false if parameter a is not reloaded from memory before the final checks. Indeed it is straightforward to see that the initial point on curve test does not detect the disturbance of a due to the fact that the input common point lies also on the disturbed curve. Furthermore, if a is not reloaded from memory before the final point on curve test, as the scalar multiplication is performed on the faulted curve, the output point $\widetilde{Q}$ lies on it. Thus the final point on curve test succeeds and the result is output.

Combined Curve

The countermeasure proposed in [32] against the sign change attack is applicable to curves over prime fields. The authors suggest to generate a random prime t and a new random curve $\mathcal{E}_t := \mathcal{E}(\mathbb{F}_t)$. They obtain then a "combined" elliptic curve $\mathcal{E}_{pt} := \mathcal{E}(\mathbb{Z}_{pt})$ which contains $\mathcal{E}_t$ and the original curve. By performing the scalar multiplication on $\mathcal{E}_{pt}$ and $\mathcal{E}_t$ they can then verify the result by using a modular reduction. This countermeasure is ineffective against our attack as we do not disturb the scalar multiplication.

Point Blinding

Intuitively a DSCA countermeasure hardly thwarts a fault attack, however the second countermeasure proposed in [57] performs well against faults, as remarked in [74]. The author of [57] suggests to thwart DSCA by pre-computing $S = [k]R$ for a random point R, then computing $Q = [k](P + R)$ and returning $Q - S$. Then at each execution a

random bit b is generated and the random points are updated with $R \leftarrow (-1)^b[2]R$ and $S \leftarrow (-1)^b[2]S$. Such a countermeasure counteracts efficiently our attack if the attacker has no control on the fault. However, if she can inject the same error twice, such a countermeasure can be bypassed. Indeed, let us assume that for two consecutive executions the bit b equals 0 (the case $b = 1$ is similar). For the same fault injected in the two executions she will thus obtain two results: $Q_1 = [k](P + 2R) - [2]S$ and $Q_2 = [k](P + 4R) - [4]S$. As the value S is pre-computed, the scalar multiplications $[2]S$ and $[4]S$ are done over $\mathcal{E}$, while the computation of $[k](P + [j]R)$ for $j = 1, 2$ is done over $\widetilde{\mathcal{E}}$. Furthermore if the fault is injected before the update of R and S, the doubling of these two values is also done over $\widetilde{\mathcal{E}}$. The attacker thus computes over $\widetilde{\mathcal{E}}$, the value $Q_2 - [2]Q_1 = [k]P + [4k]R - [4]S - [2k]P - [4k]R + [4]S$, by simplifying she obtains $Q_2 - [2]Q_1 = [k](P - [2]P)$, and thus she can mount the ECDLP on the faulted curve as the result is independent of R.

Table 2.7 – Countermeasures effectiveness against our new attack. For the point blinding countermeasure we assume that the same fault can be injected in two executions.

Countermeasure	Result
point on curve before multiplication	Ineffective
Integrity Check before multiplication	Ineffective
point on curve after multiplication	Ineffective
Integrity Check after multiplication	Effective
Combined Curve	Ineffective
Point Blinding	Ineffective

2.6.5 Conclusion

This section introduced the concept of common points, i.e. points that lie on a whole family of curves, which have never been remarked before. Then we show how to exploit their properties in our new attack to thwart state-of-the-art secure implementations with a single fault. By forcing the computation to be performed on another curve of the family, an attacker can try to solve the ECDLP on this new curve which is expected to be weaker than the standard one. The use of common points overcome the drawbacks of other known attacks, such as the need for high accuracy on the injected error. Furthermore it was claimed that one of the most complete still less expensive solutions to thwart fault attacks is to test that the used point lies on the curve. Our new attack shows that implementations relying on this countermeasure are in fact in a great danger. Simulations of our attack show that it is practical on most standard curves, and that the probability of success for a single execution exceeds 45%. By analyzing state-of-the-art countermeasures, we show that our attack is thwarted only by verifying the integrity of the curve parameter a at the end of the scalar multiplication. Last but not least, we exploited common points to mount a very efficient fault attack but one should wonder if there is no other way to make the most of these points by producing other kind of attacks or countermeasures. While there exists no actual protocol allowing the attacker to choose the input point and retrieve the result of a scalar multiplication, we stress that our work should be interpreted as a warning and a reason to avoid such kind of protocols. In particular we remark that some recent work did not take into account our admonishment and may be broken by using our strategy [144].

2.7 Conclusion

In this chapter we have introduced the concept of fault attacks which represents one of the most challenging threats to embedded cryptography. We have described in Sec. 2.1 some of the classical and more powerful fault attacks both on asymmetric and symmetric cryptosystems. In particular we have described the effect of fault attacks on the CRT-RSA cryptosystem, together with three of the most effective countermeasures used in practice. We have afterwards detailed the effects of faults on ECCs, and we have presented the countermeasures that have been suggested by researchers to thwart such attacks. Concerning symmetric cryptosystem we have focused particularly on AES, where we have detailed how differential fault attacks work. Among the various countermeasures for both symmetric and asymmetric cryptosystems we have presented the idea of infective countermeasures, which has been initially suggested for CRT-RSA but various authors adapted it to symmetric cryptography. This second chapter presents some of our work on fault attacks and infective countermeasures.

In Sec. 2.2 we analyze a recent proposal for translating redundant countermeasures into infective ones. We show that the two countermeasures obtained by using such approach are not resistant. Despite the translation method seems promising we suggest that more research should be devoted to the subject before practical application should rely on it.

Afterwards we have analyzed recent proposals for symmetric infective countermeasures. In particular we analyze the security of the first published infective countermeasure [117] for symmetric cryptosystems in Sec. 2.3. Our analysis reveals how a little bias on the infection data allows an attacker to break the countermeasure with a reasonable number of faults.

Further works on infective symmetric countermeasures has been presented in Sec. 2.4. The authors of [88] suggest a countermeasure based on the use of dummy and redundant rounds to protect symmetric implementations. Our contribution revealed flaws on the countermeasure design that allows an attacker to retrieve sensitive information despite an interesting ad promising approach to security.

Our work on the analysis of [88] seems to have motivated others to perform further research within the infective computation domain. Indeed soon after our analysis of [88], another version of the countermeasure has been published [165], which enjoys an information theoretic proof of security. We present in Sec. 2.5.1 our analysis of the new countermeasure where we suggest some fault path that has not been detected by their authors as well as possible countermeasures.

On the subject of pure fault attacks we also present our work on ECCs in Sec. 2.6. We have suggested the concept of common points on elliptic curves to the community and a detailed attack against state of the art countermeasures. We were the first to provide reasons why ECC cryptosystems should not allow the user to choose both the input point and retrieve the resulting point of a scalar multiplication. Our remark is of importance for modern protocols, for example some of the countermeasures suggested in [144] can be broken by using our common points attack.

Chapter 3

Side-Channel Attacks

"Asymptotically we're all dead."

Jacques Patarin

Contents

As it has been explained in Chap. 2, the interaction between an embedded device executing a cryptographic algorithm and its environment may reveal sensitive information about the cryptosystem. In Chap. 2 we have investigated some of the threats posed by an attacker who physically interacts with the hardware in order to modify its internal state. In this chapter we are going to investigate some of the attacks that can be mounted by passively observing the physical emissions of embedded devices. The term Side-Channel Analysis (SCA) is commonly used to denote this kind of attack.

The remainder of this chapter is organized as follows. In Sec. 3.1 we provide a introduction to side-channels. We recall the best known side-channel vectors and we introduce the theory of leakage models. We afterwards provide examples of simple and statistical side-channel attacks and countermeasures. Afterwards we present in Sec. 3.2 our recent result in horizontal side-channel attacks [14]. We show that indefinitely increasing the number of shares used to protect implementations from side-channel attacks may, under particular conditions, be counter-productive and provide advantages to the attacker. This joint work with Jean-Sebastien Coron, Emmanuel Prouff and Rina Zeitoun has been published at the CHES 2016 conference. In Sec. 3.3 we discuss a recent suggestion for a new class of side-channel and fault attacks countermeasure. We present our analysis of the countermeasure against side-channel attacks [9]. This is a joint work with Guillaume Barbu and is still unpublished. In Sec. 3.4 we present an attack that uses a combination of faults and side-channel analysis to break state-of-the-art CRT-RSA countermeasures secured against fault and side-channel attacks. The results of this joint work with Guillaume Barbu, Guillaume Dabosville, Christophe Giraud, Guenael Renault, Soline Renner and Rina Zeitoun, have been published at the PKC 2013 conference. Finally Sec. 3.5 concludes the chapter.

3.1 Side-Channels

We have seen in Chap. 2 that the classical concept of *black-box* cryptography is no longer sufficient to model the reality of attackers. Fault attacks are, however, very invasive, as the attacker must often be able to reach very sensitive zones of the hardware to inject his fault. Also, the disturbances introduced by fault vectors may be detected by the cryptosystem, which may try, under particular circumstances, to kill itself instead of revealing sensitive information. Whenever such drawbacks cannot be afforded by the attacker, an alternate solution exists. In 1998 Kocher *et al.* [110–112] showed that, as in many other physical processes, the real execution of an algorithm generates physical observables (leakages) that can be measured by an attacker and that reveal additional information on the execution internals. In this section we present different kinds of physical observations that may reveal information on the sensitive data manipulated by an algorithm. Furthermore, many different methods allow to extract information from leakage traces. Which one to choose depends on many factors, among them complexity and effectiveness. We introduce the most common of them, together with a brief introduction to the theoretical models that have been developed in the literature and some of the classical simple and statistical side-channel attacks and countermeasures.

3.1.1 Side-Channel Vectors

In order to capture side-channel information several different physical interactions exist that can be observed and measured by the attacker, depending both on its resources and on the physical access to the device. On their first known historical appearances [83,177], side-channel attacks exploited sound and radio-frequencies leakages to retrieve secret information. However, attacker techniques and instruments evolved with the continuous research and interest in the field. Nowadays a wide range of physical interactions can be measured and analyzed by the attacker, each one with its own advantages. We list hereafter the known physical observables that have been reported in the literature. We provide examples of use and, where possible, the first appearance in the literature of the discussed vector.

- *Acoustic*: Acoustic cryptanalysis consists in measuring the frequencies in the audible spectrum, between 20Hz and 20KHz produced by the device while running a given algorithm. The measure can be performed by using a simple microphone and can provide astonishing information. The first known examples of its use can be dated back to the cold war, around 1955, when the British MI5 used a microphone to decode the secret code information used by the Egyptian and French Hagelin embassy during the Suez crisis communications [177]. Although it may seem a too simple attack vector to be useful against modern cryptography, researchers have recently demonstrated its effectiveness in retrieving the RSA key used by a remote desktop computer [86].

- *Time*: (TA) Measuring the time taken to perform an operation has been one of the first attack vectors used in the literature. Its first appearance is due to Kocher *et al.* [111,112] which used timing analysis to break CRT-RSA cryptosystems. Recent works demonstrated its effectiveness against modern cryptography as in [19,20].

- *Temperature*: Temperature analysis aims at measuring the temperature variances of the hardware during an operation. Probably due to intrinsic inertia of the

temperature and the consequent slow evolution this attack vector has not been extensively used to mount attacks on cryptosystems.

- *Power Consumption*: Power consumption or power analysis (PA) is one of the the most widely known and used attack vectors nowadays. Originally introduced by Kocher *et al.* [111, 112]. It mainly consists in measuring the evolution of the power absorbed by the device under test during operations. Historically the most powerful source of side-channel information, its name is often used to denote attacks using side-channel information like Simple Power Analysis and Differential Power Analysis.

- *Electro-Magnetic Emissions*: Similarly to power consumption, electromagnetic analysis (EMA) turned out to be one of the most informative vectors for the side-channel attackers. The main idea is to use an electromagnetic probe to capture wide band electromagnetic emissions generated by the digital components of the hardware when switching states. The main difference with power analysis is the locality of EMA. While the power absorbed by the chip is generated by all of its components, an electromagnetic probe can be placed on a specific component (i.e: ALU, co-processors, RAM, etc..) and provide information generated only by the element located below the probe. Electromagnetic side-channels were introduced by Kocher *et al.* [111, 112].

- *Photonic Emissions*: It has been recently observed that CMOS technology can release data-dependent energy in the form of photon emissions [78, 152]. These emissions can be captured by using dedicated hardware, for example by using Near InfraRed (NIR) cameras, to retrieve sensitive data. This kind of attack mainly focuses on retrieving sensitive information from memories, which presents slow refresh rates, better fitting the requirements of commodity NIR cameras.

The side-channel vectors are used to acquire additional information on the instructions executed by a given hardware. However, the precise form of the information retrieved is relatively hard to define. Several theories have thus been suggested to model the leaked information. In the following section we describe such theoretical models.

3.1.2 Towards a Leakage Model

The theory of side-channel attacks and countermeasures has known a fast evolution in the last few decades due to its effectiveness in breaking cryptosystems that were not conceived to withstand physical attacks. The need to provide proofs of security in this new framework, together with the need for a rigorous approach to model side-channel attacks and countermeasures, motivated theoreticians to search for a suitable model. A model has to be simple, yet it has to provide a good approximation of reality (which is itself usually very complex) and, as remarked by Micali and Reyzin in [124], non-trivial work must be achievable within its confines. As a first (not chronological) and general remark, we recall the paradigm suggested in [124]. They suggested that the model should take into account only data manipulated during computations, which is nowadays known as the *only computation leaks* paradigm. This axiom states that for each leakage sample the amount of information retrieved by the attacker is limited to the sole data and instructions that have been used during the observed computation. Similarly to the lack of a formal definition of *elementary step* encountered in Sec. 1.1.3, here the term *single computation* is not well defined, hence it can be adapted to the

granularity required for the expression of the algorithm. The paradigm is justified by the fact that data which are not manipulated have no reason to absorb or release energy. Some objection to this constraint has been moved [98]. In particular the model cannot take into account, for example the so called cold-boot attacks [97]. However, it has been fruitfully used during the last years to construct and prove side-channel countermeasures and to simulate side-channel attacks [6, 42, 58, 59, 85, 95, 142, 146]. We list hereafter some of the best known models that are used by different works to prove very interesting results in the only computation leaks paradigm. We finally remark that leakage models allows studying particularly complex protocols by proceeding from basic blocks up to higher representation levels.

Noisy Leakage Model

The first model has been suggested by Chari *et al.* in [44]. The idea of Chari *et al.* is to assume that when observing a physical phenomenon produced by the hardware under test, the attacker is provided with an observation modeled as a value depending on the internal state of the hardware, plus a tunable Gaussian noise component.

The model as suggested by Chari *et al.* has been extended and we thus assume that the information $\mathcal{L}(x)$ provided by physical observables to the attacker during the manipulation by the hardware of the value x is of the form:

$$\mathcal{L}(x) = f(x) + \mathcal{N}(\mu, \sigma^2) \tag{3.1}$$

where $\mathcal{N}(\mu, \sigma^2)$ is a Gaussian noise of mean μ and standard deviation σ. The Gaussian noise parameters can be tuned in order to allow simulation of more or less informative leakages. The function f takes into account the fact that the hardware may only leak a part of the value manipulated. While many choices are possible for f, like for example the Hamming distance or higher degree combinations of the bits of x, the Hamming weight is the de-facto standard for the research in the field, justified by its simplicity yet realistic approximation of reality. We recall that given a value x on n bits (i.e $x = x_0, \ldots, x_{n-1}$), its Hamming weight is given by the following equation:

$$HW(x) = \sum_{i=0}^{n-1} x_i$$

In the following we will refer to this model as the *noisy Hamming weight leakage model* where we want to stress that f corresponds to the Hamming weight. By using a leakage model it is possible for example, to analyze an algorithm execution by substituting x with the different values that are taken by the algorithm variables during the execution.

Threshold Probing Model

Among the other models that have been suggested in the literature we recall in particular the *t-threshold probing* model by Ishai *et al.*. The authors of [103] provide a completely different approach than that of Chari *et al.* [44]. Informally speaking, the t-threshold probing model suggests that an adversary can probe no more than t intermediate values produced during the execution of the algorithm. The adversary can freely choose the instant of acquisition for each probe, such that, for example, she may probe the t intermediate values at once, or at different times. However, probes cannot be moved during one execution, and each probe can be used only once per execution.

More formally, the t-threshold probing model can be defined as follows. Let an algorithm $\mathcal{A}$ be defined as a set $\mathcal{X} = \{x_1, \ldots, x_\ell\}$ of ℓ intermediate results. On attacking, the adversary specifies a set of indexes $\mathcal{I} = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, \ell\}$ for $t \leq \ell$. Afterwards, the algorithm executes and returns to the adversary, additionally to the classical input and outputs, the sequence x_i for each i in $\mathcal{I}$.

We also recall that recently it has been shown in [69] that it is possible to reduce the security in (a slightly different notion as described in [140] of) the noisy leakage model to the security in the t-threshold probing model.

Random Probing Model

We remark that a known variant of this model is the *ϵ-random probing* model. Let us consider a function f as the one which best approximates the leakage of the device under test. Thus In the *ϵ-random probing* model, instead of selecting the set of indexes to probe, each index in $\{1, \ldots \ell\}$ is probed and returns $f(x_i)$ with probability $\epsilon_i \leq \epsilon$. In [69], the authors also show that by using the ϵ-random probing model, it is possible to reduce noisy leakage security to t-threshold security.

These models are used to describe attacks and countermeasures in the remainder of this chapter. In our following description we abuse the term power analysis to describe attacks that exploits any side-channel vector, not only power consumption. In the next section we describe the first side-channel attack on cryptosystems that appeared in the literature, originally described by Kocher *et al.* in [111]. We also describe in particular its adaptation to the AES, and some of the most effective countermeasures that have been conceived to protect implementations in the presence of side-channel leakages.

3.1.3 Simple Side-Channel Attacks

Simple side-channel attacks try to recover sensitive information by observation of single leakage traces. They are often denoted by the acronym SPA, standing for Simple Power Analysis. Two main strategies exist to recover sensitive information by simple analysis. The first strategy tries to identify dependencies between instructions and leakage traces, while the second one tries directly to exploit dependencies between leakages and data. However, as a first example, we remark that simple power analysis may provide useful reverse engineering information on the algorithm to the attacker.

Instruction dependency

Among simple attacks the most intuitive is probably instruction-data dependency attacks. This kind of attack is characterized by the attacker trying to retrieve sensitive information thanks to the execution of portions of code that depends on the sensitive variable's value. In order to illustrate how this attack works, we recall here a classical example about the verification of a PIN (Personal Identification Number). We assume that the attacker is given an hardware that stores a secret 10-digits PIN. Furthermore she is provided with a keyboard to enter a user PIN. If she finds the value of the secret PIN then she wins and evil things happen. We provide in Alg. 7 an example pseudo-code that may be internally used by the hardware to test the user input PIN.

Without further countermeasures, a brute-force attack requires at most $10^{10} \simeq 2^{30}$ tests to disclose the secret PIN. On the other hand, we observe that such an algorithm presents a weakness in the way it handles wrong PINs. We assume that each loop in Alg. 7 takes equal time to execute. By measuring the time taken by the device to

Algorithm 7: Simple PIN test

Inputs : 10 digit user-PIN
Output: "Yes" or "No"

```
1 for i = 0 to 9 do
2 |   if user-PIN[i] ≠ PIN[i] then return ("No");
3 end
4 return ("Yes")
```

answer a given PIN, the attacker can greatly reduce the attack complexity. She can, for example, try all the 10 different values for the first digit of the PIN. She can decide which of the values is correct by comparing the execution time taken to answer each query. Due to the early exit at Step 2, all wrong values will return upon comparing the first digit, where the correct value requires a comparison of at least the two first digits, thus taking at least twice the time of the others. By using this technique she can iterate the same attack on all 10 digits, and reduce the total number of steps required to retrieve the PIN to $10 * 10 \simeq 2^7 \ll 2^{30}$. We have chosen to present the reader with a simple example in order to better understand instruction-data dependency attacks. Other interesting and more sophisticated attacks of the same philosophy exist and allow, for example, to break unprotected implementations of the square-and-multiply exponentiation algorithm, or equivalently multiply-and-add scalar multiplication algorithms on ECCs [113]. It is almost impossible to be exhaustive due to the large body of work produced, thus we have provided here only some examples of simple attacks. However, we remark that whenever the execution of an instruction depends on some sensitive data value, then there exists a potential threat for the implementation.

Data dependency

Simple side-channel attacks may also provide information in other settings. We discuss here how to retrieve information on sensitive variables when only data-leakages dependencies are available. Let us for example assume that the power consumption during an implementation execution carries information on the values manipulated by the algorithm (due to a noisy leakage model, for example). Furthermore it may be possible, for example, to retrieve a good approximation of the Hamming weight of intermediate variables. While it has been shown that finding out the Hamming weights on algorithms like the DES or the AES alone may be of little help, this information may be exploited to mount an attack. For example, Goubin [93] demonstrated that detecting long sequences of zeros may allow to break unprotected scalar multiplication algorithms. More recent works relies on the fact that a long sequence of zeros or ones may be better recognizable by simple side-channels to break an AES-GCM [19, 20].

Countermeasures

Since the introduction of simple power analysis by Kocher *et al.* [111,112], several sound countermeasures have been identified. In the following we describe some of the countermeasures that have been found in the literature to thwart simple side-channel attacks. The first and most important countermeasure against simple side-channel attacks is to construct algorithms whose flow is independent of the sensitive data manipulated. In the PIN test example (Alg. 7), instruction-data independence can be achieved quite

easily. Indeed we show in Alg. 8 another version of the same algorithm that is secure against the timing attack described before.

Algorithm 8: SPA Secure PIN test

Inputs : 10 digit user-PIN
Output: "Yes" or "No"

out = "Yes";
for *i = 0 to 9* **do**
  if *user-PIN[i]* $\neq$ *PIN[i]* **then** out ="No";
end
return (out)

Several authors suggested algorithms that trade off speed versus security, this is the case for example of the so called "regular algorithms" like the *square-and-multiply always* or the *Montgomery Ladder exponentiation* algorithms for both RSA and ECC [57, 91, 106]. Another well known and interesting example is the *Side-Channel Atomicity* introduced by Chevallier-Mames *et al.* [47]. The countermeasure consists in re-writing an algorithm, in particular exponentiation and scalar multiplications, by using common building blocks of code in order to break dependency between instructions and data.

In the next section we present advanced attacks that use statistical treatments to reduce the noise present on the traces and to retrieve the sensitive variables manipulated.

3.1.4 Differential Side-Channel Analysis

When simple side-channel attacks are not sufficient to break cryptographic implementations, attackers can resort to Differential side-channel Analysis. Often denoted Differential Power Analysis (DPA) for historical reasons, it has been first described by Kocher *et al.* [111] in 1998. This kind of attack is based on statistical analysis of a set of leakage traces acquired on the hardware under test. Differential power analysis has received great interest since its introduction, probably due to two main factors. The first reason is its impressive efficiency to break cryptosystem implementations and SPA protected implementations. The second is probably due to the early development of useful models that allowed to produce formal research [103, 124]. In the following we use the AES as an example to describe how DPA may be applied to a cryptographic algorithm.

Differential Power Analysis

In order to explain how DPA works, we use the example of AES-128. The algorithm, as described in Chap. 1, Sec. 1.2.3, takes a 128-bit input message, and combines it with a 128-bit key to obtain the ciphertext. In the first round of the algorithm each message byte is combined (XOR) with one byte of the first round key. Afterwards each byte of the result is passed through a nonlinear function denoted S-Box. The aim of the attacker is to retrieve the secret round key. She can count on a set on known input messages m_i and a set of leakage traces c_i corresponding to the treatment of the first round. In the following we assume that the side-channel observations of the attacker can be modeled by a noisy leakage model. A statistical method that allows to detect data-dependent statistical correlations among a set of measurements is commonly denoted

DPA. The original method based on difference of means consists in dividing the set of measurements into two subsets, then computing the average of each subset and the difference between the averages of these subsets. Then two possibilities arise, either the choice which guided the assignment of each trace to a set is uncorrelated with the measurement, in this case the difference of averages will asymptotically should approach zero. Otherwise, in the case of a correct assignment of the traces, the difference of the averages asymptotically approaches a non-zero value. The *selection function* used to assign traces to a set can be chosen among those relating the message byte and the key byte. For example, by using one of the intermediate functions of the first round like the output of the XOR or the one of non linear transformations like the S-Box. We provide in the following a brief list of different statistical distinguishers that have been suggested in the literature to mount statistical side-channel attacks.

Other Statistical Distinguishers

Since the introduction of DPA, more sophisticated attacks have been suggested in the literature. They are mainly based on the choice of different statistical distinguishers that are better suited to detect statistical correlations between measurements in particular circumstances, or that provide better tolerance to noise.

Correlation Power Analysis The Correlation Power Analysis (CPA), based on the Pearson Correlation coefficient has been introduced in [39] in order to provide an alternative distinguisher to DPA. The CPA coefficient of a set of traces C and a set of hypothesis H is given by:

$$\rho(C, H) = \frac{cov(C, H)}{\sigma_C \sigma_H}$$

CPA appears to be more tolerant to noise than DPA and generally provides better experimental results by revealing correlations with a lower number of traces. This is probably due to the fact that its formula takes into account both the mean and the variance of the measurements. Moreover its inclusion of a leakage model on the formulas allows for more precise results (as long as the model is consistent with the observations). However the main drawback of the CPA is its need for a leakage model. The attacker must indeed guess the model of the leakage of the hardware for the attack to be successful. If the real leakage model is too different from the theoretical guessed model, then the distinguisher may converge too slowly or not work at all.

Mutual Information Analysis The Mutual Information Analysis (MIA), is based on the estimation of the mutual information between curves and hypotheses. It has been suggested by Gierlichs *et al.* in [87].

In its basic version, the attacker builds, for a given leakage model, a set of theoretical distributions conditioned by the key values hypotheses. Afterwards she collects leakages which are used to estimate the distribution of the samples in each point. By comparing the estimated distribution against the theoretical ones, the retained key guess is the one which minimizes the distance between the observed and the theoretical distributions.

Depending on the metric used to compute the distance between the distributions one can trade computational complexity for detection efficiency. MIA mainly differs from other distinguishers in that it can detect correlations between measurements and non-linear functions of the leakage model. Another fundamental difference between

MIA and other distinguishers is that MIA can detect leakages to any statistical order (see Sec. 3.1.5). For more details we refer the reader to [12, 87, 170].

Linear Regression Analysis Linear Regression Analysis has been introduced in [67, 151]. Among the advantages provided by this distinguisher is the fact that together with the best correlating hypothesis, it finds a set of coefficients describing a leakage model that (for a chosen metric) minimizes the distance between the hypotheses and the observations. Thus the attacker retrieves both the best correlating key hypothesis, and the best matching leakage model in each observation sample. The obtained leakage model can then be plugged into attacks that require less computational resources than LRA, like for example CPA. Further details can be found in [63, 67, 151].

Maximum Likelihood (Template) Analysis Template attacks, initially suggested by Chari *et al.* are the strongest possible form of side-channel attack in an information theoretic sense [45]. As their authors observe, instead of trying to remove or reduce the noise in traces, template attacks tries to precisely model the noise in order to fully extract information in a single sample. Template attacks often require access to so called *open-samples*, an identical experimental device to the one under attack that can be programmed at will by the attacker. Attacks that can exploits open-samples are also called *profiled* attacks, in contrast with *non-profiled* attacks where the weaker adversary is only able to observe the targeted device behavior. The idea of template attack is to use the maximum likelihood approach together with details of the cryptographic function being attacked. We provide here the idea of the maximum likelihood approach, while a more precise and formal definition will be given in Sec. 3.2. We assume that the attacker knows a set of samples $x_0, \ldots, x_n$ of i.i.d observations from a distribution with unknown pdf $f_0(\cdot)$. Let f_0 belongs to a family of distribution functions parametrized by a parameter $\theta_i \in \Theta$ (i.e.: $f_0(\cdot) = f(\cdot \mid \theta_0)$. The attacker wants to find an estimator that gets as close to θ_0 as possible. The idea is to find the value θ_i which maximizes the value:

$$\mathcal{L}(\theta_i) = \prod_{i=0}^{n} f(x_i \mid \theta_i)$$

In practice this boils down to find the θ_i for which the set of observations are the most probable.

3.1.5 On the Order of Side-Channel Attacks

We briefly discuss in this section different terminologies used in the literature to address the order of side-channel attacks. In the literature we can find different definitions, sometimes confusing, about the order of side-channel countermeasures and attacks (see for example [58, 63, 95, 123, 129, 142]. The same terminology is indeed used to define two different concepts, hence the confusion. Let us describe them in the following.

Multi-variate Order

The attacks that we have presented in Sec. 3.1.4 exploits leakage information in one particular point of the curve to find dependencies between manipulated data and observations. However, it has been observed that it is interesting to combine (with a carefully selected function) two leakage points together and mount the attack on the

resulting value [123]. It is common, for example on leakage observations made of several time samples, to multiply different time samples on the same leakage trace. The hope is that such computation reveal information concealed by particular side-channel countermeasures (c.f. Sec. 3.1.6).

However, the attacker may not be aware of which points are to be combined, hence she may try to compute all combinations of pairs of points. Consequently, given an initial curve of n points, she obtains a resulting curve of n^2 points. Hence this approach presents two main drawbacks: the number of points to analyze for each observation grows exponentially and the number of curves required to retrieve information from them grows. However, in some circumstances this strategy may pay off and break (first-order) side-channel secure implementations [123]. It has been proved that some combination functions are better suited in particular circumstances (see [141]).

Statistical order

In the literature there also exists another notion of attack order, which comes from statistical analysis, and that concerns the order of the statistical distinguisher used in the attack. As we have seen, the DPA attack reveals differences on the mean value of carefully selected sets of curves. As the mean is the 1-st order statistical moment of a random variable, DPA is defined, in this context, as a 1-st order statistical attack. It is also possible to mount DPA attacks based on the variance of the leakages. As the variance is the 2-nd statistical moment of a random variable, hence variance-DPA is a 2-nd order statistical attack. As one may guess countermeasures exist that are secure against t-order side-channel attacks for large t, but that are thwart by a "simple" bivariate side-channel attack.

Thus in the remainder of this manuscript we adopt the standard notation of *high-order* for both notions, while we identify if it is multi-variate or statistical when necessary to avoid confusion. After this clarification on our notation, we can describe in the rest of this section some of the best known side-channel countermeasures.

3.1.6 Countermeasures

We divide the countermeasures in two categories, those which use heuristic methods to increase traces noise (for example reducing the SNR), whose objective is to reduce attacks efficiency, and the proven countermeasures which can provably thwart side-channel attacks under reasonable assumptions.

Heuristic

Heuristics countermeasures consist in strategies to reduce the Signal-to-Noise Ratio (SNR) of the traces in order to reduce the efficiency of statistical attacks. Among them we can find hardware countermeasures such as the insertion of random clock skews (jitter) during computation with the purpose of breaking traces synchronization, such that the same point on all traces does not carry the information about the same computation. Other hardware countermeasures consist in activating unused hardware modules which consume power during a sensitive computation, such that the power absorbed by the sensitive computation will be masked by the power absorbed by the other hardware modules. Other hardware countermeasures devoted to thwart EMA, involve the insertion of additional metal layers to shield and disturb the electro-magnetic emissions

of the device [38, 51]. Similar to hardware countermeasures there exist software countermeasures that can achieve similar results. This is the case for example of the jitter countermeasure, which can be achieved in software by random addition of idempotent instructions. Similar concepts can be applied at higher levels by executing several times the sensitive algorithm on dummy data. The real execution can then be randomly executed among all the dummy ones. This countermeasure, like all the other presented above can only slow down attackers, but a sufficiently motivated hacker can just acquire more traces to reduce the noise introduced by the heuristic countermeasure.

Proven

Another approach to thwart side-channel attacks consists in finding sound countermeasures that can be proven to be safe with some (reasonable) assumption.

Various techniques to protect algorithms from leakage have been inspired by *secret sharing* [29, 156] or *multi-party computation* [46]. Furthermore the notion of *t-order* security in the multi-variate connotation has the same origins.

The basic idea is to randomly split a secret into several shares such that the adversary needs all of them to reconstruct the secret. Masking was soon identified as a sound countermeasure when side-channel attacks appeared in the literature [44,94]. For example, given a sensitive variable x, it can be split (shared) into t shares by generating $t-1$ random variables $\{x_0, \ldots, x_{t-2}\}$. The last share is computed as $x_{t-1} = x \oplus x_0 \oplus \ldots \oplus x_{t-2}$. The sensitive computation can then be performed on each share separately. Afterwars, the sensitive result value can be obtained as: $x = x_0 \oplus \ldots \oplus x_t$. More formally, for two positive integers n and d, a (n, d)-*sharing* of a variable x defined over some finite field $\mathbb{F}_{2^k}$ is a random vector $(x_1, x_2, \ldots, x_n)$ over $\mathbb{F}_{2^k}$ such that $x = \sum_{i=1}^{n} x_i$ holds (*completeness equality*) and any tuple of $d-1$ shares x_i is a uniform random vector over $(\mathbb{F}_{2^k})^{d-1}$. If $n = d$, the terminology simplifies to n-*sharing*. Since then, many works have been published to address the practical implementation and/or the security of masking for various ciphers. This strategy provides the correct result as long as the function computed on the shares is linear with respect to the sharing method, i.e: XOR in our example. However, as soon as a non linear function needs to be performed on the sensitive data, further ideas must be found. In particular the AES S-Box involve a non-linear step (with respect to the XOR), and its computation requires several multiplications of the shares. One solution consists in using precomputed tables to obtain the result of the non linear step [58,62,65,131]. However, in order to reduce the memory footprint of the countermeasure, other solutions have been suggested. This is the case for example of the ISW *secure multiplication scheme* over $\mathbb{F}_2$ introduced by Ishai *et al.* in [103]. We present in the following its extension to any field $\mathbb{F}_{2^k}$ proposed by Rivain and Prouff in 2010 [146].

It has been shown that the ISW scheme is secure against a $\lfloor n/2 \rfloor$–order attack in the probing security model, see [103]. Furthermore, it has been shown [61, 70, 140] that the Rivain-Prouff scheme of [61, 146] achieves security at order $\lfloor n/2 \rfloor$ in the probing security model.

In the next section we suggest an analysis of the Rivain-Prouff scheme against horizontal attacks. We show the counter intuitive result that in particular settings increasing the number of shares may reduce the system security instead of augmenting it.

Algorithm 9: SecMult

Require: the n-sharings $(x_i)_{i\in[1..n]}$ and $(y_j)_{j\in[1..n]}$ of $x^\star$ and $y^\star$ respectively
Ensure: the n-sharing $(c_i)_{i\in[1..n]}$ of $x^\star \cdot y^\star$

1: **for** $i = 1$ **to** n **do**
2: **for** $j = i + 1$ **to** n **do**
3: $r_{i,j} \leftarrow^{\$} \mathbb{F}_{2^k}$
4: $r_{j,i} \leftarrow (r_{i,j} + x_i \cdot y_j) + x_j \cdot y_i$
5: **end for**
6: **end for**
7: **for** $i = 1$ **to** n **do**
8: $c_i \leftarrow x_i \cdot y_i$
9: **for** $j = 1$ **to** n, $j \neq i$ **do** $c_i \leftarrow c_i + r_{i,j}$
10: **end for**
11: **return** $(c_1, c_1, \ldots, c_n)$

3.2 Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme

Using secret sharing at the implementation level enables one to achieve provable security in the so-called *probing security model* [103]. In this model, it is assumed that an adversary can recover information on a limited number of intermediate variables of the computation. This model has been argued to be practically relevant to address so-called *higher-order* (i.e.: multi-variate) side-channel attacks and it has been the basis of several efficient schemes to protect block ciphers [6, 42, 58, 59, 85, 95, 142, 146]. More recently, it has been shown in [69] that the probing security of an implementation actually implies its security in the more realistic *noisy leakage model* introduced in [140]. More precisely, if an implementation obtained by applying the compiler in [103] is secure at order n in the probing model, then [70, Theorem 3] shows that the success probability of distinguishing the correct key among $|\mathbb{K}|$ candidates is bounded above by $|\mathbb{K}| \cdot 2^{-n/9}$ if the leakage L_i on each intermediate variable X_i satisfies:

$$\mathrm{I}(X_i; L_i) \leqslant 2 \cdot (|\mathbb{K}| \cdot (28n + 16))^{-2} \ ,$$

where $\mathrm{I}(\cdot;\cdot)$ denotes the *mutual information* and where the index i ranges from 1 to the total number of intermediate variables.

In this section we investigate what happens when the above condition is not satisfied. Since the above mutual information $I(X_i; L_i)$ can be approximated by $k/(8\sigma^2)$ in the Hamming weight model in $\mathbb{F}_{2^k}$, where σ is the noise in the measurement (see Sec. A.3), this amounts to investigating the security of Ishai-Sahai-Wagner's (ISW) implementations when the number of shares n satisfies:

$$n > c \cdot \sigma$$

As already observed in previous works [52, 169], the fact that the same share (or more generally several data depending on the same sensitive value) is manipulated several times may open the door to new attacks which are not taken into account in the probing model. Those attacks, sometimes called *horizontal* [52] or *(Template) algebraic* [135, 169] exploit the algebraic dependency between several intermediate results to discriminate key hypotheses.

In this paper, we exhibit two (horizontal) side channel attacks against the ISW multiplication algorithm. These attacks show that the use of this algorithm (and its extension proposed by Rivain and Prouff in [146]) may introduce a weakness with respect to horizontal side channel attacks if the sharing order n is such that $n > c \cdot \sigma^2$, where σ is the measurement noise. While the first attack is too costly (even for low noise contexts) to make it applicable in practice, the second attack, which essentially iterates the first one until it achieves a satisfying likelihood, shows very good performances. For instance, when the leakages are simulated by noisy Hamming weights computed over $\mathbb{F}_{2^8}$ with $\sigma = 1$, it recovers all the shares of a 21-sharing. We also confirm the practicality of our attack with a real life experiment on a development platform embedding the `ATMega328` processor (see Section 3.2.7). Actually, in this context where the leakages are multivariate and not univariate as in our theoretical analyses and simulations, the attack appears to be more efficient than expected and recovers all the shares of a n-sharing when $n \geqslant 40$.

We also describe a variant of Rivain-Prouff's multiplication that is still provably secure in the original ISW model, and also heuristically secure against our new attacks. Our new countermeasure is similar to the countermeasure in [76], in that it can be divided in two steps: a "matrix" step in which starting from the input shares x_i and y_j, one obtains a matrix $x_i \cdot y_j$ with n^2 elements, and a "compression" step in which one uses some randomness to get back to a n-sharing c_i. Assuming a leak-free component, the countermeasure in [76] is proven secure in the noisy leakage model, in which the leakage function reveals all the bits of the internal state of the circuit, perturbed by independent binomial noise. Our countermeasure does not use any leak-free component, but is only heuristically secure in the noisy leakage model (see Section 3.2.8 for our security analysis).

Eventually, we describe in Section 3.2.9 a new mask refreshing algorithm with complexity $\mathcal{O}(n \cdot \log n)$ instead of $\mathcal{O}(n^2)$ for the classical algorithm in [11]. A completely different mask refreshing algorithm is described in [118], with complexity only $\mathcal{O}(n)$; however our new algorithm is significantly simpler. Our new mask refreshing algorithm enables to decrease the complexity of the previous variant of Rivain-Prouff's multiplication from $\mathcal{O}(n^2 \log n)$ to $\mathcal{O}(n^2)$, hence the same complexity as the original Rivain-Prouff multiplication.

3.2.1 Notation

Calligraphic letters, like $\mathcal{X}$, are used to denote finite sets (*e.g.* $\mathbb{F}_{2^n}$). The corresponding large letter X denotes a random variable over $\mathcal{X}$, while the lower-case letter x a value over $\mathcal{X}$. The probability of an event ev is denoted by $\Pr[ev]$. The *probability distribution function* (pdf for short) of a *continuous* random variable X is denoted by $f_X(\cdot)$. It will sometimes be denoted by $\mathrm{p}_X(\cdot)$ if X is discrete. The pdf of the random variable $X|Y$ is denoted by $f_{X|Y}(\cdot)$. The expectation and the variance of a random variable X are respectively denoted by $\mathbb{E}[X]$ and $\mathrm{Var}[X]$. The covariance between two random variables X and Y is denoted by $\mathrm{Cov}[X, Y]$. The *Signal to Noise Ratio* (SNR) of a univariate noisy observation L of a random variable X defined as *the signal*, is defined as $\mathrm{SNR} \doteq \frac{\mathrm{Var}[\mathbb{E}[L|X]]}{\mathbb{E}[\mathrm{Var}[L|X]]}$ (where we recall that $\mathbb{E}[L \mid X]$ and $\mathrm{Var}[L \mid X]$ are both viewed as functions of the random variable X).

The Gaussian distribution of dimension t with t-size expectation vector μ and $t \times t$ covariance matrix Σ is denoted by $\mathcal{N}(\mu, \Sigma)$. We recall that the corresponding proba-

bility density function (pdf) is defined for every $\ell \in \mathbb{R}^t$ as:

$$f(\ell) = \frac{1}{\sqrt{(2\pi)^t \det(\Sigma)}} \exp\left(-\frac{1}{2}(\ell-\mu)' \cdot \Sigma^{-1} \cdot (\ell-\mu)\right) \ , \tag{3.2}$$

where $(\cdot)'$ denotes the transposition operation and $\det(\cdot)$ denotes the matrix determinant. The corresponding cumulative distribution function (cdf) F is defined for every $(a_i, b_i)_{i\in[1..t]} \in ((\mathbb{R} \cup \{-\infty, +\infty\})^2)^t$ by

$$F(\mathbf{a}, \mathbf{b}) = \int_{a_t}^{b_t} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(\ell_1, \ell_2, \cdots, \ell_t)\ d\ell_1 d\ell_2 \cdots d\ell_t \doteq \int_{\mathbf{a}}^{\mathbf{b}} f(\ell)\ d\ell \ , \tag{3.3}$$

with $\ell \doteq (\ell_1, \ell_2, \cdots, \ell_t)$, $\mathbf{a} \doteq (a_1, a_2, \cdots, a_t)$ and $\mathbf{b} \doteq (b_1, b_2, \cdots, b_t)$.

If the dimension t equals 1, then the Gaussian distribution is said to be *univariate* and its covariance matrix is reduced to the variable variance denoted σ^2. Otherwise, it is said to be *multivariate*.

The *entropy* $\mathbb{H}(X)$ of a discrete r.v. X defined over $\mathbb{F}_{2^k}$ aims at measuring the amount of information provided by an observation of X. It is defined by $\mathbb{H}(X) = -\sum_{x\in\mathbb{F}_{2^k}} f_X(x) \log f_X(x)$. The *differential entropy* extends the notion of entropy to continuous, and possibly t-dimensional, r.v. Contrary to the entropy, the differential entropy can be negative. In the case of a real valued random variable $\vec{L}$, it is defined by:

$$\mathbb{H}(\vec{L}) = -\int_{\ell\in\mathbb{R}^t} f_{\vec{L}}(\ell) \log(f_{\vec{L}}(\ell)) d\ell \ . \tag{3.4}$$

If $\vec{L}$ is a t-dimensional Gaussian r.v. with covariance matrix Σ (*i.e.* its pdf is defined by (3.33)), then its entropy satisfies the following equality:

$$\mathbb{H}(\vec{L}) = \frac{1}{2} \log((2\pi e)^t \det(\Sigma)) \ . \tag{3.5}$$

In the general case, there is no analytical expression for the differential entropy of a r.v. X whose pdf mixes more than one Gaussian pdf. However, upper and lower bounds can be derived [43].

An algorithm with domain $(\mathbb{F}_{2^k})^n$ is said to be $(n-1)$th-order secure in the probing model if on input an n-sharing $(x_1, x_2, \ldots, x_n)$ of some variable x, it admits no tuple of $n-1$ or less intermediate variables that depends on x. An algorithm achieving such a security is resistant to the class of (vertical) $(n-1)$th-order side-channel attacks under the *only computation leaks* assumption [124].

3.2.2 Horizontal DPA Attack

Problem description.

Let $(x_i)_{i\in[1..n]}$ and $(y_i)_{i\in[1..n]}$ be respectively the n-sharings of $x^\star$ and $y^\star$ (namely, we have $x^\star = x_1 + \cdots + x_n$ and $y^\star = y_1 + \cdots + y_n$). We assume that an adversary gets, during the processing of Algorithm 9, a single observation of each of the following random variables for $1 \leq i, j \leq n$:

$$L_i = \varphi(x_i) + B_i \tag{3.6}$$
$$L'_j = \varphi(y_j) + B'_j \tag{3.7}$$
$$L''_{ij} = \varphi(x_i \cdot y_j) + B''_{ij} \tag{3.8}$$

where φ is an unknown function which depends on the device architecture, where B_i, B'_j are Gaussian noise of standard deviation $\sigma/\sqrt{n}$, and B''_{ij} is Gaussian noise with standard deviation σ. Namely we assume that each x_i and y_j is processed n times, so by averaging the standard deviation is divided by a factor $\sqrt{n}$, which gives $\sigma/\sqrt{n}$ if we assume that the initial noise standard deviation is σ. The random variables associated to the ith share x_i and the jth share y_j are respectively denoted by X_i and Y_j. Our goal is to recover the secret variable $x^\star$ (and/or $y^\star$).

3.2.3 Complexity Lower Bound: Entropy Analysis of Noisy Hamming Weight Leakage

For simplicity, we first restrict ourselves to a leakage function φ equal to the Hamming weight of the variable being manipulated. In that case, the mutual information $\mathrm{I}(X;L)$ between the Hamming weight of a uniform random variable X defined over $\mathbb{F}_{2^k}$ and a noisy observation L of this Hamming weight can be approximated as:

$$\mathrm{I}(X;L) \simeq \frac{k}{8\sigma^2} \ , \tag{3.9}$$

if the noise being modeled by a Gaussian random variable has standard deviation σ. This approximation, whose derivation is given inSec. A.3, is only true for large σ.

To recover a total of $2n$ shares (n shares of $x^\star$ and $y^\star$ respectively) from $3n^2$ Hamming weight leakages (namely each manipulation leaks according to (3.6)-(3.8) with $\varphi = \mathrm{HW}$), the total amount of information to be recovered is $2n \cdot k$ if we assume that the shares are i.i.d. with uniform distribution over $\mathbb{F}_{2^k}$. Therefore, since we have a total of $3n^2$ observations during the execution of Algorithm 9, we obtain from (3.9) that the noise standard deviation σ and the sharing order n must satisfy the following inequality for a side channel attack to be feasible:

$$3 \cdot n^2 \cdot \frac{k}{8\sigma^2} > 2n \cdot k \ . \tag{3.10}$$

We obtain an equality of the form $n > c \cdot \sigma^2$ for some constant c, as in a classical (vertical) side channel attack trying to recover $x^\star$ from n observations of intermediate variables depending on $x^\star$ [44]. This analogy between horizontal and vertical attacks has already been noticed in previous papers like [52] or [18]. Note that in principle the constant c is independent of the field degree k (which has also been observed in previous papers, see for instance [161]).

Attack With Perfect Hamming Weight Observations

We consider the particular case of perfect Hamming weight measurements (no noise). We show that even with perfect observations of the Hamming weight, depending on the finite-field representation, we are not always guaranteed to recover the secret variable $x^\star$.

More precisely, we consider the pdf of the random variable corresponding to perfect Hamming weight measurements:

$$\vec{H} \mid X_i \doteq (\mathrm{HW}(X_i), \mathrm{HW}(Y_j), \mathrm{HW}(X_i \cdot Y_j)) \mid X_i$$

defined for every $x_i \in \mathbb{F}_{2^k}$ and every triplet $(h_1, h_2, h_3) \in [0..k]^3$ by:

$$f_{\vec{H}|X_i}((h_1,h_2,h_3), x_i) \doteq \Pr[\mathrm{HW}(X_i) = h_1, \mathrm{HW}(Y_j) = h_2, \mathrm{HW}(X_i \cdot Y_j) = h_3 \mid X_i = x_i] \ . \tag{3.11}$$

If the probability distributions $f_{\vec{H}|X_i=x_i}$ are distinct for every $x_i \in \mathbb{F}_{2^k}$, then one can recover the secret variable $x^\star$ with overwhelming probability from enough measurements. However this property depends on the finite field $\mathbb{F}_{2^k}$ and its representation. For example in $\mathbb{F}_{2^4}$, if the representation $\mathbb{F}_{2^4} \simeq \mathbb{F}_2[t]/(t^4+t+1)$ is used then it may be checked that the finite field elements $x_i = t+1$ and $x_i' = t^3+t^2$ are associated to identical distributions f_{x_i} and $f_{x_i'}$; so we cannot distinguish between x_i and x_i' (the other field elements have distinct probability distributions). In $\mathbb{F}_{2^8}$, for the representation $\mathbb{F}_{2^8} \simeq \mathbb{F}_2[t]/(t^8+t^4+t^3+t+1)$ (as used in AES), all finite field elements have distinct probability distributions, but this is not always the case with other irreducible polynomials.

In summary, even with perfect observations of the Hamming weight, depending on the finite-field representation, we are not always guaranteed to recover the secret variable $x^\star$; however for the finite field representation used in AES the attack enables to recover the secret $x^\star$ for a large enough number of observations.

Maximum Likelihood Attack: Theoretical Attack with the full ISW State

For most field representations and leakage functions, the maximum likelihood approach used in the previous section (in particular in (3.11)), recovers the i-th share of $x^\star$ from an observation of L_i and an observation of (L'_j, L''_{ij}) for every $j \in [1..n]$. It extends straightforwardly to noisy scenarios and we shall detail this extension in Section 3.2.4. However, the disadvantage of this approach is that it recovers each share separately, before rebuilding $x^\star$ and $y^\star$ from them. From a pure information theoretic point of view this is suboptimal since (1) the final purpose is not to recover all the shares perfectly but only the shared values and (2) only $3n$ observations are used to recover each share whereas the full tuple of $3n^2$ observations brings more information. Actually, the most efficient attack in terms of leakage exploitation consists in using the joint distribution of $(L_i, L'_j, L''_{ij})_{i,j\in[1..n]}$ to distinguish the correct hypothesis about $x^\star = x_1 + x_2 + \cdots + x_n$ and $y^\star = y_1 + y_2 + \cdots + y_n$.

As already observed in Section 3.1.6, during the processing of Algorithm 9, the adversary may get a tuple $(\ell_{ij})_{j\in[1..n]}$ (resp. $(\ell'_{ij})_{i\in[1..n]}$) of n observations for each L_i (resp. each L'_j) and one observation ℓ''_{ij} for each L''_{ij}. The full tuple of observations $(\ell_{ij}, \ell'_{ij}, \ell''_{ij})_{i,j}$ is denoted by $\vec{\ell}$, and we denote by $\vec{L}$ the corresponding random variable [1]. Then, to recover $(x^\star, y^\star)$ from $\vec{\ell}$, the *maximum likelihood approach* starts by estimating the pdfs $f_{\vec{L}|X^\star=x^\star,Y^\star=y^\star}$ for every possible $(x^\star, y^\star)$, and then estimates the following vector of distinguisher values for every hypothesis (x, y):

$$\mathrm{d}^\star_{\mathrm{ML}}(\vec{\ell}) \doteq \left(f_{\vec{L}|(X^\star,Y^\star)}(\vec{\ell}, (x,y))\right)_{(x,y)\in\mathbb{F}^2_{2^k}} \tag{3.12}$$

The pair (x, y) maximizing the above probability is eventually chosen.

At a first glance, the estimation of the pdfs $f_{\vec{L}|X^\star=x^\star,Y^\star=y^\star}$ seems to be challenging. However, it can be deduced from the estimations of the pdfs associated to the manipulations of the shares. Indeed, after denoting by $\mathrm{p}_{x,y}$ each probability value in the right-hand side of (3.12), and by using the law of total probability Theorem 7 together

[1] In (3.6)-(3.8), it is assumed that the observations $(\ell_{ij})_{j\in[1..n]}$ and $(\ell'_{ij})_{i\in[1..n]}$ are averaged to build a single observation with noise divided by $\sqrt{n}$. This assumption is not done here in order to stay as general as possible.

with the fact that the noises are independent, we get:

$$2^{2kn} \cdot \mathrm{p}_{x,y} = \sum_{\substack{x_1,\cdots,x_n \in \mathbb{F}_{2^k} \\ x = x_1 + \cdots + x_n}} \sum_{\substack{y_1,\cdots,y_n \in \mathbb{F}_{2^k} \\ y = y_1 + \cdots + y_n}} \prod_{i,j=1}^{n} f_{L_i|X_i}(\ell_{ij}, x_i) \cdot f_{L'_j|Y_j}(\ell'_{ij}, y_j) \cdot f_{L''_{ij}|X_iY_j}(\ell''_{ij}, x_i y_j) \ .$$

Unfortunately, even if the equation above shows how to deduce the pdfs $f_{\vec{L}|(X^\star,Y^\star)}(\cdot,(x^\star,y^\star))$ from characterizations of the shares' manipulations, a direct processing of the probability has complexity $O(2^{2nk})$. By representing the sum over the x_i's as a sequence of convolution products, and thanks to Walsh transforms processing, the complexity can be easily reduced to $O(n2^{n(k+1)})$. The latter complexity stays however too high, even for small values of n and k, which led us to look at alternatives to this attack.

3.2.4 First Attack: ML Attack on a Single Matrix Row

Attack Description.

In this section, we explain how to recover each share x_i of $x^\star$ separately, by observing the processing of Algorithm 9. Applying this attack against all the shares leads to the full recovery of the sensitive value $x^\star$ with some success probability, which is essentially the product of the success probabilities of the attack on each share separately.

Given a share x_i, the attack consists in collecting the leakages on $(y_j, x_i \cdot y_j)$ for every $j \in [1..n]$. Therefore the attack is essentially a horizontal version of the classical (vertical) second-order side-channel attack, where each share x_i is multiplicatively masked over $\mathbb{F}_{2^k}$ by a random y_j for $j \in [1..n]$.

The most efficient attack to maximize the amount of information recovered on X_i from a tuple of observations $\vec{\ell} \doteq \ell_i, (\ell'_j, \ell''_{ij})_{j\in[1..n]} \hookleftarrow \vec{L} \doteq L_i, (L'_j, L''_{ij})_{j\in[1..n]}$ consists in applying a *maximum likelihood approach* [44, 96], which amounts to computing the following vector of distinguisher values:

$$\mathrm{d}_{\mathrm{ML}}(\vec{\ell}) \doteq \left(f_{\vec{L}|X_i}(\vec{\ell}, \hat{x}_i)\right)_{\hat{x}_i \in \mathbb{F}_{2^k}} \tag{3.13}$$

and in choosing the candidate $\hat{x}_i$ which maximizes the probability.

Let us respectively denote by $f(\cdot,\cdot)$, $f'(\cdot,\cdot)$ and $f''(\cdot,\cdot)$ the pdfs $f_{L_i|X_i}(\cdot,\cdot)$, $f_{L'_j|Y_j}(\cdot,\cdot)$ and $f_{L''_{ij}|X_iY_j}(\cdot,\cdot)$. Since the variables $L_i \mid X_i = \hat{x}_i$ and all the variables $(L'_j, L''_{ij} \mid X_i = \hat{x}_i)$ are mutually independent whatever $\hat{x}_i \in \mathbb{F}_{2^k}$, we have:

$$f_{\vec{L}|X_i}(\vec{\ell}, \hat{x}_i) = f(\ell_i, \hat{x}_i) \prod_{j=1}^{n} f_{(L'_j,L''_{ij})|X_i}((\ell'_j, \ell''_{ij}), \hat{x}_i) \ . \tag{3.14}$$

Applying the law of total probability, the pdf of $(L'_j, L''_{ij}) \mid X_i = \hat{x}_i$ can moreover be developed such that:

$$f_{(L'_j,L''_{ij})|X_i}((\ell'_j, \ell''_{ij}), \hat{x}_i) = \sum_{y \in \mathbb{F}_{2^k}} f_{(L'_j,L''_{ij})|(X_i,Y_j)}((\ell'_j, \ell''_{ij}), (\hat{x}_i, y)) \cdot \mathrm{p}_{Y_j}(y) \ , \tag{3.15}$$

that is

$$f_{(L'_j,L''_{ij})|X_i}((\ell'_j, \ell''_{ij}), \hat{x}_i) = 2^{-k} \sum_{y \in \mathbb{F}_{2^k}} f'(\ell'_j, y) \cdot f''(\ell''_{ij}, \hat{x}_i \cdot y) \ , \tag{3.16}$$

since the Y_j's are assumed to have uniform distribution and since the variables $L'_j \mid Y_j = y$ and $L''_{ij} \mid X_iY_j = \hat{x}_i \cdot y$ are independent.

Eventually, each score $f_{\vec{L}|X_i}(\vec{\ell}, \hat{x}_i)$ in (3.13) may be computed based on the following expression:

$$f_{\vec{L}|X_i}(\vec{\ell}, \hat{x}_i) = 2^{-nk} f(\ell_i, \hat{x}_i) \cdot \prod_{j=1}^{n} \Big(\sum_{y \in \mathbb{F}_{2^k}} f'(\ell'_j, y) \cdot f''(\ell''_{ij}, \hat{x}_i \cdot y) \Big) \ ,$$

where all the distributions are Gaussian ones (and hence can be easily evaluated). Hence, an approximation of the pdf in (3.13) can be deduced from the approximations (aka *templates*) of the distributions associated to the manipulations of the shares X_i, Y_j and $X_i Y_j$.

In practice, one often makes use of the equivalent (averaged) log-likelihood distinguisher $\mathrm{d}'_{\mathrm{ML}}(\cdot)$ which, in our case, may be defined as:

$$\mathrm{d}'_{\mathrm{ML}}(\vec{\ell}) = \frac{1}{n} \log \mathrm{d}_{\mathrm{ML}}(\vec{\ell}) + k \log 2 \tag{3.17}$$

$$\simeq \left(\frac{1}{n} \Big(\log f(\ell_i, \hat{x}_i) + \sum_{j=1}^{n} \log \{ \sum_{y \in \mathbb{F}_{2^k}} f'(\ell'_j, y) \cdot f''(\ell''_{ij}, \hat{x}_i \cdot y) \} \Big) \right)_{\hat{x}_i \in \mathbb{F}_{2^k}} \tag{3.18}$$

Remark 1. *The same approach described in this section can be applied to iteratively recover each share y_j of y. The attack description can be straightforwardly deduced by exchanging the roles of X_i and Y_j (and the indices i and j). For instance,* (3.15) *becomes:*

$$f_{(L_i, L''_{ij})|Y_j}((\ell_i, \ell''_{ij}), \hat{y}_j) = \sum_{x \in \mathbb{F}_{2^k}} f_{(L_i, L''_{ij})|(X_i, Y_j)}((\ell_i, \ell''_{ij}), (x, \hat{y}_j)) \cdot \mathrm{p}_{X_i}(x) \ . \tag{3.19}$$

Complexity Analysis

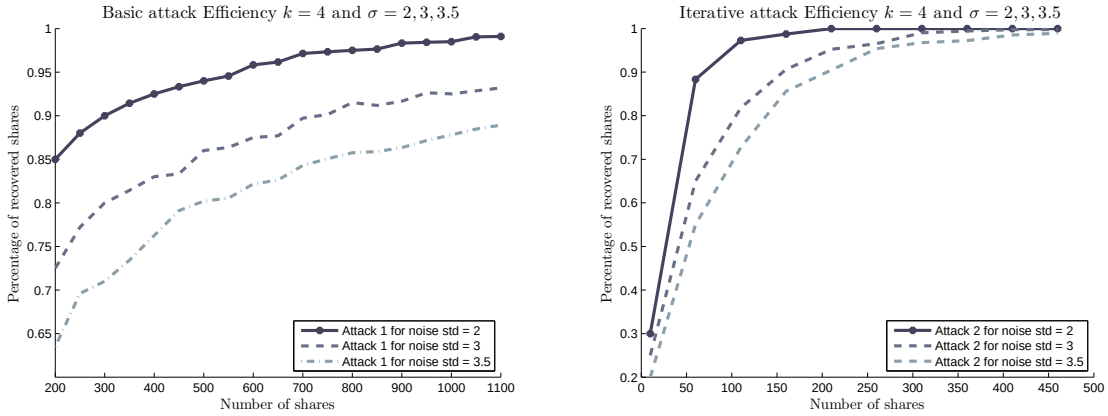
As mentioned previously, given a share x_i, the attack consists in collecting the leakages on $(y_j, x_i \cdot y_j)$ for every $j \in [1..n]$. Therefore the attack is essentially an horizontal version of the classical (vertical) second-order side-channel attack. In principle the number n of leakage samples needed to recover x_i with good probability (aka the attack complexity) should consequently be $n = \mathcal{O}(\sigma^4)$ [44, 96, 161]. This holds when multiplying two leakages both with noise having σ as standard deviation. However here the leakage on y_j has a noise with a standard deviation $\sigma/\sqrt{n}$ instead of σ (thanks to the averaging step). Therefore the noise of the product becomes $\sigma^2/\sqrt{n}$ (instead of σ^2), which gives after averaging with n measurements a standard deviation of σ^2/n, and therefore an attack complexity satisfying $n = \mathcal{O}(\sigma^2)$, as in a classical first-order side-channel attack.

Numerical Experiments

The attack presented in Sect. 3.2.4 has been implemented against each share x_i of a value x, with the leakages being simulated according to (3.6)-(3.8) with $\varphi = \mathrm{HW}$. For the noise standard deviation σ and the sharing order n, different values have been tested to enlighten the relation between these two parameters. We stated that an attack succeeds iff the totality of the n shares x_i have been recovered, which leads to the full recovery of $x^\star$. We recall that, since the shares x_i are manipulated n times, measurements for the leakages L_i and L'_j have noise standard deviations $\sigma/\sqrt{n}$ instead of σ. For efficiency reasons, we have chosen to work in the finite field $\mathbb{F}_{2^4}$ (namely $k = 4$ in previous analyses).

σ (SNR)	0 ($+\infty$)	0.2 (25)	0.4 (6.25)	0.6 (2.77)	0.8 (1.56)	1 (1)
n	12	14	30	73	160	284

Table 3.1 – First attack: number of shares n as a function of the noise σ to succeed with probability > 0.5



(a) Basic Attack

(b) Iterative Attack

Figure 3.1 – Percentage of recovered shares with respect to n for $\sigma = 2, 3, 3.5$ and $k = 4$

For various noise standard deviations σ with SNR $= k(2\sigma)^{-2}$ (*i.e.* SNR $= \sigma^{-2}$ for $k = 4$), Table 3.1 gives the average minimum number n of shares required for the attack to succeed with probability strictly greater than 0.5 (the averaging being computed over 300 attack iterations). The attack complexity $n = \mathcal{O}(\sigma^2)$ argued in Sect. 3.2.4 is confirmed by the trend of these numerical experiments. Undeniably, this efficiency is quickly too poor for practical applications where n is small (clearly lower than 10) and the SNR is high (smaller than 1). However, it must be noticed that the attack quickly recovers 90% of the shares even for $\sigma = 2, 3$ (see Figure 3.1a), and the shares which are not recovered (because they are not given the maximum likelihood) have a good ranking. Consequently, combining this attack with one of the Key Enumeration (KEA) techniques recently developed (see *e.g.* [92,119]) should significantly increase the attack efficiency.

3.2.5 Second Attack: Iterative ML Attack

Attack Description

From the discussions in Sect. 3.2.3, and in view of the poor efficiency of the previous attack, we investigated another strategy which targets all the shares simultaneously. Essentially, the core idea of our second attack described below is to apply several attacks recursively on the x_i's and y_j's, and to refine step by step the likelihood of each candidate for the tuple of shares. Namely, we start by applying the attack described in Section 3.2.4 in order to compute, for every i, a likelihood probability for each hypothesis $X_i = x$ (x ranging over $\mathbb{F}_{2^k}$); then we apply the same attack in order to compute, for every j, a likelihood probability for each hypothesis $Y_j = y$ (y ranging over $\mathbb{F}_{2^k}$) with the single difference that the probability $\mathrm{p}_{X_i}(x)$ in (3.19) is replaced by

the likelihood probability which was just computed[2]. Then, one reiterates the attack to refine the likelihood probabilities $(\mathrm{p}_{X_i}(x))_{x\in\mathbb{F}_{2^k}}$, by evaluating (3.15) with the uniform distribution $\mathrm{p}_{Y_j}(y)$ being replaced by the likelihood probability new-$\mathrm{p}_{Y_j}(y)$ which has been previously computed. The scheme is afterwards repeated until the maximum taken by the pdfs of each share X_i and Y_j is greater than some threshold β. In order to have better results, we perform the whole attack a second time, by starting with the computation of the likelihood probability for each hypothesis $Y_j = y$ instead of starting by $X_i = x$.

We give the formal description of the attack processing in Algorithm 10 (in order to have the complete attack, one should perform the while loop a second time, by rather starting with the computation of new-$\mathrm{p}_{Y_j}(y)$ instead of new-$\mathrm{p}_{X_i}(x)$).

3.2.6 Numerical Experiments

The iterative attack described in Algorithm 10 has been tested against leakages simulations defined exactly as in Section 3.2.4. As previously we stated that an attack succeeds if the totality of the n shares x_i have been recovered, which leads to the full recovery of $x^\star$. For various noise standard deviations σ with SNR $= k(2\sigma)^{-2}$, Table 3.2 gives the average minimum number of shares n required for the attack to succeed with probability strictly greater than 0.5 (the averaging being computed over 300 attack iterations). The first row corresponds to $k = 4$, and the second row to $k = 8$ (the corresponding SNRs are $\mathrm{SNR}_4 = \sigma^{-2}$ and $\mathrm{SNR}_8 = (\sqrt{2}\sigma^2)^{-1}$). Numerical experiments yield greatly improved results in comparison to those obtained by running the basic attack. Namely, in $\mathbb{F}_{2^4}$, for a noise $\sigma = 0$, the number of shares required is 2, while 12 shares were needed for the basic attack, and the improvement is even more confirmed with a growing σ: for a noise $\sigma = 1$, the number of shares required is 25, while 284 shares were needed for the basic attack. It can also be observed that the results for shares in $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^8}$ are relatively close, the number of shares being most likely slightly smaller for shares in $\mathbb{F}_{2^4}$ than in $\mathbb{F}_{2^8}$. This observation is in-line with the lower bound in (3.10), where the cardinality 2^k of the finite field plays no role. Once again, it may be observed that the attack quickly recovers 90% of the shares even for $\sigma \in \{2, 3, 3.5\}$ (see Figure 3.1b), and the shares which are not recovered (because they are not given the maximum likelihood) have a good ranking. Consequently, combining this attack with KEA should still increase the attack efficiency.

σ $(\mathrm{SNR}_4, \mathrm{SNR}_8)$	0 $(+\infty, +\infty)$	0.2 $(25, 17.67)$	0.4 $(6.25, 4.41)$	0.6 $(2.77, 1.96)$	0.8 $(1.56, 1.10)$	1 $(1, 0.7071)$
n (for $\mathbb{F}_{2^4}$)	2	2	3	6	13	25
n (for $\mathbb{F}_{2^8}$)	5	6	8	11	16	21

Table 3.2 – Iterative attack: number of shares n as a function of the noise σ to succeed with probability > 0.5 in $\mathbb{F}_{2^4}$ (first row) and in $\mathbb{F}_{2^8}$ (second row).

3.2.7 Practical Results

We provide in this section practical experiments for the attacks described in Sec. 3.2.4 and 3.2.5 on a commercial development board.

[2]Actually to get the probability of $X_i \mid \vec{L}$ instead of $\vec{L} \mid X_i$, Bayes' Formula is applied which explains the division by the sum of probabilities in the lines 14 and 19 in Algorithm 10.

Algorithm 10: Iterative Maximum Likelihood Attack

Require: a threshold β, an observation ℓ_i of each L_i, an observation ℓ'_j of each L'_j and one observation ℓ''_{ij} of each L''_{ij} (the random variables being defined as in (3.6)-(3.8))

Ensure: a n-tuple of pdfs $(\mathrm{p}_{X_i})_i$ (resp. $(\mathrm{p}_{Y_i})_i$) such that, for every $i \in [1..n]$, at least one $\hat{x}_i$ (resp. $\hat{y}_j$) satisfies $\mathrm{p}_{X_i}(\hat{x}_i) \geqslant \beta$ (resp. $\mathrm{p}_{Y_i}(\hat{y}_j) \geqslant \beta$)

for $i = 1$ **to** n **do**
  for $x \in \mathbb{F}_{2^k}$ **do** {Initialize the likelihood of each candidate for X_i}
    $\mathrm{p}_{X_i}(x) = f_{L_i|X_i}(\ell_i, x)$
  end for
  for $y \in \mathbb{F}_{2^k}$ **do** {Initialize the likelihood of each candidate for Y_i}
    $\mathrm{p}_{Y_i}(y) = f_{\vec{L}'_i|Y_i}(\ell'_i, y_i)$
    new-$\mathrm{p}_{Y_i}(y) = \mathrm{p}_{Y_i}(y)$
  end for
end for

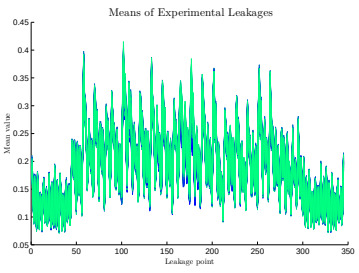
while `end` $\neq n$ **do**
  `end` $\leftarrow 0$
  for $i = 1$ **to** n **do**
    for $x \in \mathbb{F}_{2^k}$ **do** {Compute/Update the likelihood of each candidate for X_i}
      new-$\mathrm{p}_{X_i}(x) =$
      $2^{-(2n+1)k} \frac{\mathrm{p}_{X_i}(x)}{\sum_{x' \in \mathbb{F}_{2^k}} \mathrm{p}_{X_i}(x')} \prod_{j=1}^{n} \sum_{y \in \mathbb{F}_{2^k}} \frac{\text{new-}\mathrm{p}_{Y_j}(y)}{\sum_{y' \in \mathbb{F}_{2^k}} \text{new-}\mathrm{p}_{Y_j}(y')} \cdot f_{L''_{ij}|X_iY_j}(\ell''_{ij}, x \cdot y)$
    end for
  end for
  for $i = 1$ **to** n **do**
    for $y \in \mathbb{F}_{2^k}$ **do** {Compute/Update the likelihood of each candidate for Y_i}
      new-$\mathrm{p}_{Y_i}(y) =$
      $2^{-(2n+1)k} \frac{\mathrm{p}_{Y_i}(y)}{\sum_{y' \in \mathbb{F}_{2^k}} \mathrm{p}_{Y_i}(y')} \prod_{j=1}^{n} \sum_{x \in \mathbb{F}_{2^k}} \frac{\text{new-}\mathrm{p}_{X_j}(x)}{\sum_{x' \in \mathbb{F}_{2^k}} \text{new-}\mathrm{p}_{X_j}(x')} \cdot f_{L''_{ij}|X_iY_j}(\ell''_{ij}, x \cdot y)$
    end for
  end for
  for $i = 1$ **to** n **do**
    if $\max_x(\text{new-}\mathrm{p}_{X_i}(x)) \geqslant \beta$ and $\max_y(\text{new-}\mathrm{p}_{Y_i}(y)) \geqslant \beta$ **then**
      `end` $++$
    end if
  end for
end while

Setup In order to provide real life experiments of the attack described in this work we have mounted it against a development platform embedding the `ATMega328` processor. The presence of decoupling capacitors between the ground pins of the processor and the reference ground did not allow us to correctly measure the power consumption of the processor. We therefore de-soldered the ground pins of the processor and connected them to a 20 OHm resistor whose other end was connected to the reference ground. After this preparation, we used a passive probe connected to an oscilloscope in order to register the current absorbed by the processor by measuring the difference of potential at the ends of the resistor. Thanks to a probe bandwidth of 500MHz, we pre-filtered all the frequencies higher than 200MHz. We moreover used a sampling rate of 100MHz on the oscilloscope as the best compromise between accuracy and measurement trace size.

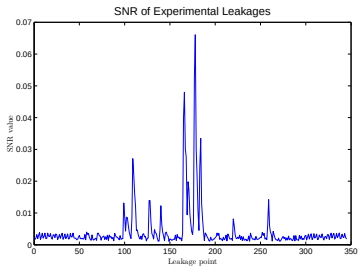
Leakage Characterization

For our implementation of Alg. 9, the `mov` instruction is used to manipulate the shares x_i, y_j and the multiplication results $x_i \cdot y_j$. We therefore chose to target it in our attack experiments. An advantage of targeting a single instruction that manipulates all the values is that we obtain homogeneous leakages for all the manipulated data. Furthermore, the `mov` instruction may be found in many different architectures and we therefore think that our attack can be reproduced quite easily.

For the leakage characterization phase, we measured $200,000$ leakages of the `mov` instruction parametrized with randomly generated values. Each measure was composed of 340 points, which is essentially the size of a relevant sample of instantaneous measures for the `mov` instruction in our setup. This campaign allowed us to characterize the leakage related to the processing `mov y,x` for `x` ranging between 0 and 255 and for `y` being constant (our implementation uses the same destination register for all the shares); more precisely, each `x` was associated to a mean vector $\mu_{\mathtt{x}} \in \mathbb{R}^{340}$ and a covariance matrix $\Sigma_{\mathtt{x}} \in \mathbb{R}^{340} \times \mathbb{R}^{340}$. The 256 means are plotted in Figure 3.2a. To reduce the dimension of our *templates*, we afterwards estimated the signal-to-noise ratio of the acquisitions at each point in order to identify the best points of interest for our attack. The results are plotted in Figure 3.2b.



(a) Mean vectors of all 256 classes.



(b) SNR on each leakage point.

The peaks in Figure 3.2b allowed us to choose a first set of points. In addition, a simple maximum likelihood attack on each of these points separately has also been mounted in order to find those who provided the most information. This step finally provided us with a set of 11 points to mount our attack. It may be observed that the best SNR obtained with our setup is around 0.07 which, for our simulations, corresponds to the case $k = 16$ and $\sigma = 3.75$, and $k = 256$ and $\sigma = 3.2$. Our experiments should

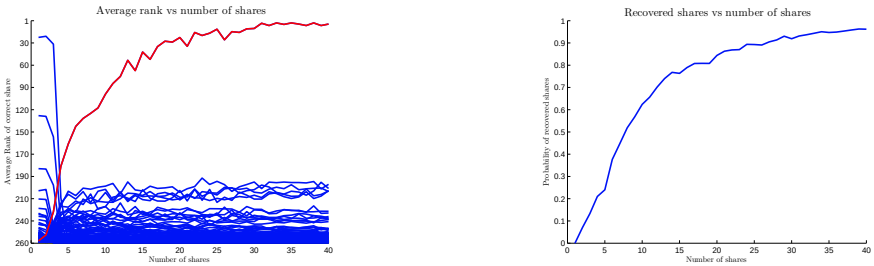
therefore be compared to the simulations plotted in the gray dot-and-dashed curve in Figure 3.1a (for the basic attack) and in 3.1b (for the iterative attack).

Attacks

Thanks to the characterization described in the previous section, we performed the attacks of Sec. 3.2.4 and 3.2.5 against our implementation of Alg. 9 parametrized with different sharing orders n.

Maximum Likelihood Attack Experiments As a first experiment we tested the attack of Sec. 3.2.4 in order to evaluate the evolution of the rank of the correct hypothesis on a single share x_i when n varies. In Figure 3.3a we plot the ranking of the correct hypothesis, in red, among all 256 byte values, for each n between 1 and 40. The results have been obtained by averaging the result among 100 repetitions of the same attack for each order. From Figure 3.3a we can observe that the correct hypothesis is ranked among the first 50 values as soon as $n > 10$. We also observe that we need $n > 35$ for the correct hypothesis to be firmly ranked first by the basic attack.

During the same attack we have also evaluated the average number of shares of $x^\star$ correctly retrieved for each order n. This allowed us to obtain an estimation of the minimum sharing order n required to successfully mount the attack. The result of such evaluation is depicted in Figure 3.3b. Even if $n = 40$ appears to be a necessary condition to directly recover all the shares, the post-processing of our attack results with a KEA algorithm [92, 119] should allow to recover them for smaller n ($n = 10$ seems to be achievable at a reasonable cost).



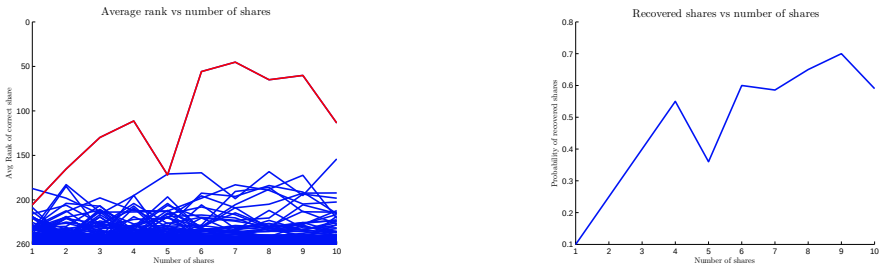
(a) Avg rank of the correct hypothesis vs number of shares.

(b) Avg number of correctly retrieved shares.

Figure 3.3 – Results for the attack of Sec. 3.2.4 when n varies between 1 and 40.

Iterative Attack Experiments We have afterwards mounted the attack described in Sec. 3.2.5 in order to measure the success probability with respect to the masking order. As before we provide the average ranking of the correct hypothesis (in red) for different values of the order n. Due to time constraints only 10 repetitions of the attack have been averaged. The evolution of the rank of the correct hypothesis is depicted in Figure 3.4a. As before, we also provide the average number of shares correctly retrieved for each order n in Figure 3.4b.

Even if we did not get results as smooth as for previous experiments, we can observe an overall better detection rate for the iterative attack if we compare to the basic one (as actually expected). Furthermore, we have averaged the results of the above attack on 10 repetitions when $n = 20, 30, 40$. For such order we have obtained an average number of



(a) Avg rank of the correct hypothesis vs number of shares.

(b) Avg number of correctly retrieved shares.

Figure 3.4 – Results for the attack of Sec. 3.2.5 when n varies between 1 and 10.

correctly retrieved shares of 0.88, 0.93 and 0.97, respectively. We thus remark again that by using KEA post-processing on the results of our attack the correct hypothesis should be recovered with a reasonable number of shares. In particular it seems reasonable to assume that $n = 10$ provides better results and lower costs with respect to the basic attack.

Experiments Conclusions

We have successfully proved the effectiveness of our attack on a real implementation using the `ATMega328` processor. We obtain better results on our experimentations than those predicted by theory in Sec. 3.2.4 and Sec. 3.2.5 for similar SNR values. We have investigated such behavior and we conjecture that the better results are due to the use of a multi-variate attack on 11 points for the experimental attacks, where the theoretical results are computed for a mono-variate attack. The more points allows for an improvement of the global SNR which can explain the disparity between theory and practice success probabilities.

3.2.8 A Countermeasure against the previous Attacks

Description

In the following, we describe a countermeasure against the previous attack against the Rivain-Prouff algorithm. More precisely, we describe a variant of Algorithm 9, called RefSecMult, to compute an n-sharing of $c = x^\star \cdot y^\star$ from $(x_i)_{i\in[1..n]}$ and $(y_i)_{i\in[1..n]}$. Our new algorithm is still provably secure in the original ISW probing model, and heuristically secure against the horizontal side-channel attacks described in the previous sections.

As observed in [76], the ISW and Rivain-Prouff countermeasures can be divided in two steps: a "matrix" step in which starting from the input shares x_i and y_j, one obtains a matrix $x_i \cdot y_j$ with n^2 elements, and a "compression" step in which one uses some randomness to get back to an n-sharing c_i. Namely the matrix elements $(x_i \cdot y_j)_{1\leq i,j\leq n}$ form an n^2-sharing of $x^\star \cdot y^\star$:

$$x^\star \cdot y^\star = \left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{j=1}^{n} y_j\right) = \sum_{1\leq i,j\leq n} x_i \cdot y_j \tag{3.20}$$

and the goal of the compression step is to securely go from such an n^2-sharing of $x^\star \cdot y^\star$ to an n-sharing of $x^\star \cdot y^\star$.

Algorithm 11: RefSecMult

Require: n-sharings $(x_i)_{i\in[1..n]}$ and $(y_j)_{j\in[1..n]}$ of $x^\star$ and $y^\star$ respectively
Ensure: an n-sharing $(c_i)_{i\in[1..n]}$ of $x^\star \cdot y^\star$

$M_{ij} \leftarrow \mathsf{MatMult}((x_1,\ldots,x_n),(y_1,\ldots,y_n))$
for $i = 1$ **to** n **do**
  for $j = i+1$ **to** n **do**
    $r_{i,j} \leftarrow^{\$} \mathbb{F}_{2^k}$
    $r_{j,i} \leftarrow (r_{i,j} + M_{ij}) + M_{ji}$
  end for
end for
for $i = 1$ **to** n **do**
  $c_i \leftarrow M_{ii}$
  for $j = 1$ **to** n, $j \neq i$ **do** $c_i \leftarrow c_i + r_{i,j}$
end for
return $(c_1, c_1, \ldots, c_n)$

Our new countermeasure (Algorithm 11) uses the same compression step as Rivain-Prouff, but with a different matrix step, called MatMult (Algorithm 12), so that the shares x_i and y_j are not used multiple times (as when computing the matrix elements $x_i \cdot y_j$ in Rivain-Prouff). Eventually the MatMult algorithm outputs a matrix $(M_{ij})_{1\leq i,j\leq n}$ which is still an n^2-sharing of $x^\star \cdot y^\star$, as in (3.20); therefore using the same compression step as Rivain-Prouff, Algorithm 11 outputs an n-sharing of $x^\star \cdot y^\star$, as required.

Algorithm 12: MatMult

Require: the n-sharings $(x_i)_{i\in[1..n]}$ and $(y_j)_{j\in[1..n]}$ of $x^\star$ and $y^\star$ respectively
Ensure: the n^2-sharing $(M_{ij})_{i\in[1..n],j\in[1..n]}$ of $x^\star \cdot y^\star$

if $n = 1$ **then**
  $\vec{M} \leftarrow [x_1 \cdot y_1]$
else
  $\vec{X}^{(1)} \leftarrow (x_1,\ldots,x_{n/2})$, $\vec{X}^{(2)} \leftarrow (x_{n/2+1},\ldots,x_n)$
  $\vec{Y}^{(1)} \leftarrow (y_1,\ldots,y_{n/2})$, $\vec{Y}^{(2)} \leftarrow (y_{n/2+1},\ldots,y_n)$
  $\vec{M}^{(1,1)} \leftarrow \mathsf{MatMult}(\vec{X}^{(1)}, \vec{Y}^{(1)})$
  $\vec{X}^{(1)} \leftarrow \mathsf{RefreshMasks}(\vec{X}^{(1)})$, $\vec{Y}^{(1)} \leftarrow \mathsf{RefreshMasks}(\vec{Y}^{(1)})$
  $\vec{M}^{(1,2)} \leftarrow \mathsf{MatMult}(\vec{X}^{(1)}, \vec{Y}^{(2)})$
  $\vec{M}^{(2,1)} \leftarrow \mathsf{MatMult}(\vec{X}^{(2)}, \vec{Y}^{(1)})$
  $\vec{X}^{(2)} \leftarrow \mathsf{RefreshMasks}(\vec{X}^{(2)})$, $\vec{Y}^{(2)} \leftarrow \mathsf{RefreshMasks}(\vec{Y}^{(2)})$
  $\vec{M}^{(2,2)} \leftarrow \mathsf{MatMult}(\vec{X}^{(2)}, \vec{Y}^{(2)})$
  $\vec{M} \leftarrow \begin{bmatrix} \vec{M}^{(1,1)} & \vec{M}^{(1,2)} \\ \vec{M}^{(2,1)} & \vec{M}^{(2,2)} \end{bmatrix}$
end if
return $\vec{M}$

As illustrated in Fig. 3.5, the MatMult algorithm is recursive and computes the $n \times n$ matrix in four sub-matrix blocs. This is done by splitting the input shares x_i and y_j in two parts, namely $\vec{X}^{(1)} = (x_1,\ldots,x_{n/2})$ and $\vec{X}^{(2)} = (x_{n/2+1},\ldots,x_n)$, and similarly $\vec{Y}^{(1)} = (y_1,\ldots,y_{n/2})$ and $\vec{Y}^{(2)} = (y_{n/2+1},\ldots,y_n)$, and recursively processing the four sub-matrix blocs corresponding to $\vec{X}^{(u)} \times \vec{Y}^{(v)}$ for $1 \leq u, v \leq 2$. To prevent

the same share x_i from being used twice, each input block $\vec{X}^{(u)}$ and $\vec{Y}^{(v)}$ is refreshed before being used a second time, using a mask refreshing algorithm. An example of such mask refreshing, hereafter called RefreshMasks, can for instance be found in [69]; see Algorithm 13. Since the mask refreshing does not modify the xor of the input $n/2$-vectors $\vec{X}^{(u)}$ and $\vec{Y}^{(v)}$, each sub-matrix block $\vec{M}^{(u,v)}$ is still an $n^2/4$-sharing of $(\oplus\vec{X}^{(u)}) \cdot (\oplus\vec{X}^{(v)})$, and therefore the output matrix $\vec{M}$ is still an n^2-sharing of $x^\star \cdot y^\star$, as required. Note that without the RefreshMasks, we would have $M_{ij} = x_i \cdot y_j$ as in Rivain-Prouff.

Algorithm 13: RefreshMasks

Input: $a_1, \ldots, a_n$
Output: $c_1, \ldots, c_n$ such that $\sum_{i=1}^n c_i = \sum_{i=1}^n a_i$

1: **for** $i = 1$ **to** n **do** $c_i \leftarrow a_i$
2: **for** $i = 1$ to n **do**
3: **for** $j = i + 1$ to n **do**
4: $r \leftarrow \{0,1\}^k$
5: $c_i \leftarrow c_i + r$
6: $c_j \leftarrow c_j + r$
7: **end for**
8: **end for**
9: **return** $c_1, \ldots, c_n$

Since the RefreshMasks algorithm has complexity $\mathcal{O}(n^2)$, it is easy to see that the complexity of our RefSecMult algorithm is $\mathcal{O}(n^2 \log n)$ (instead of $\mathcal{O}(n^2)$ for the original Rivain-Prouff countermeasure in Alg. 9). Therefore for a circuit of size $|C|$ the complexity is $\mathcal{O}(|C| \cdot n^2 \log n)$, instead of $\mathcal{O}(|C| \cdot n^2)$ for Rivain-Prouff. The following lemma shows the soundness of our RefSecMult countermeasure.

Lemma 1 (Soundness of RefSecMult)**.** *The* RefSecMult *algorithm, on input n-sharings $(x_i)_{i\in[1..n]}$ and $(y_j)_{j\in[1..n]}$ of $x^\star$ and $y^\star$ respectively, outputs an n-sharing $(c_i)_{i\in[1..n]}$ of $x^\star \cdot y^\star$.*

Proof. We prove recursively that the MatMult algorithm, taking as input n-sharings $(x_i)_{i\in[1..n]}$ and $(y_j)_{j\in[1..n]}$ of $x^\star$ and $y^\star$ respectively, outputs an n^2-sharing M_{ij} of $x^\star \cdot y^\star$. The lemma for RefSecMult will follow, since as in Rivain-Prouff the lines 2 to 12 of Alg. 11 transform an n^2-sharing M_{ij} of $x^\star \cdot y^\star$ into an n-sharing of $x^\star \cdot y^\star$.

The property clearly holds for $n = 1$. Assuming that it holds for $n/2$, since the RefreshMasks does not change the xor of the input $n/2$-vectors $\vec{X}^{(u)}$ and $\vec{Y}^{(v)}$, each sub-matrix block $\vec{M}^{(u,v)}$ is still an $n^2/4$-sharing of $(\oplus\vec{X}^{(u)}) \cdot (\oplus\vec{X}^{(v)})$, and therefore the output matrix $\vec{M}$ is still an n^2-sharing of $x^\star \cdot y^\star$, as required. This proves the lemma. □ □

Remark 2. *The description of our countermeasure requires that n is a power of 2, but it is easy to modify the countermeasure to handle any value of n. Namely in Algorithm 12, for odd n it suffices to split the inputs x_i and y_j in two parts of size $(n-1)/2$ and $(n+1)/2$ respectively, instead of $n/2$.*

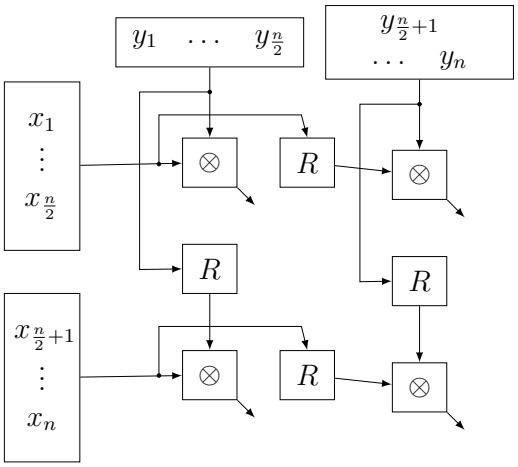


Figure 3.5 – The recursive MatMult algorithm, where R represents the RefreshMasks Algorithm, and $\otimes$ represents a recursive call to the MatMult algorithm.

Security Analysis

Proven security in the ISW probing model.

Lemma 2 (t-SNI of RefSecMult)**.** *Let $(x_i)_{1\leq i\leq n}$ and $(y_i)_{1\leq i\leq n}$ be the input shares of the* SecMult *operation, and let $(c_i)_{1\leq i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < n$, there exist two subsets I and J of indices with $|I| \leq t_1$ and $|J| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$ and $y_{|J}$.*

Heuristic security against horizontal-DPA attacks. We stress that the previous lemma only proves the security of our countermeasure against t probes for $n \geq t+1$, so it does not prove that our countermeasure is secure against the horizontal-DPA attacks described in previous sections, since such attacks use information about n^2 intermediate variables instead of only $n-1$.

As illustrated in Fig. 3.5, the main difference between the new RefSecMult algorithm and the original SecMult algorithm (Alg. 9) is that we keep refreshing the x_i shares and the y_j shares blockwise between the processing of the finite field multiplications $x_i \cdot y_j$. Therefore, as opposed to what happens in SecMult, we never have the same x_i being multiplied by all y_j's for $1 \leq j \leq n$. Therefore an attacker cannot accumulate information about a specific share x_i, which heuristically prevents the attacks described in this paper.

3.2.9 Mask Refreshing with Complexity $\mathcal{O}(n \cdot \log n)$

In this section we describe a new mask refreshing algorithm with complexity $\mathcal{O}(n \cdot \log n)$ instead of $\mathcal{O}(n^2)$ for the classical algorithm recalled in previous section. For simplicity we first assume that n is a power of 2; we refer to Appendix A.5 for a generalization to arbitrary n. Note that a completely different mask refreshing algorithm is described in [118], with complexity only $\mathcal{O}(n)$; however our algorithm is simpler.

As illustrated in Fig. 3.6, our algorithm is defined recursively. First, a pre-processing layer L_I is applied on the n input shares a_i, corresponding to lines 5 to 9 of Algorithm 14 below. Then the RefreshMasks algorithm is applied recursively on the two halves

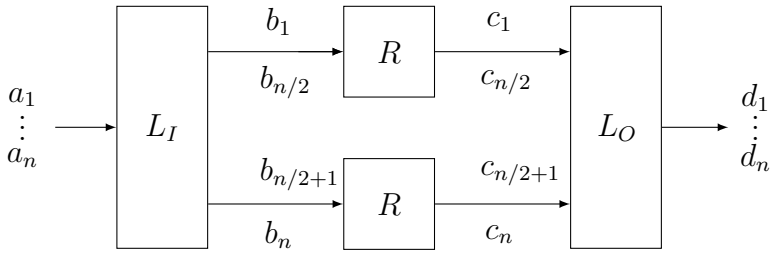


Figure 3.6 – Recursive definition of RefreshMasks, with the pre-processing layer L_I, the two recursive applications of RefreshMasks to the two halves of the shares, and the post-processing layer L_O.

of the shares. Eventually a post-processing layer L_O is applied (same as L_I), before outputting the output shares d_i (it corresponds to Lines 12 to 16 in Alg. 14).

Algorithm 14: RefreshMasks

Require: $a_1, \ldots, a_n$
Ensure: $d_1, \ldots, d_n$ such that $\bigoplus_{i=1}^{n} d_i = \bigoplus_{i=1}^{n} a_i$

1: **if** $n = 2$ **then**
2: $\quad r \leftarrow^{\$} \{0,1\}^k$
3: $\quad$ **return** $(a_1 \oplus r, a_2 \oplus r)$
4: **end if**
5: **for** $i = 1$ **to** $n/2$ **do**
6: $\quad r_i \leftarrow^{\$} \{0,1\}^k$
7: $\quad b_i \leftarrow a_i \oplus r_i$
8: $\quad b_{n/2+i} \leftarrow a_{n/2+i} \oplus r_i$ $\{b_i \oplus b_{n/2+i} = a_i \oplus a_{n/2+i}\}$
9: **end for**
10: $(c_1, \ldots, c_{n/2}) \leftarrow$ RefreshMasks$(b_1, \ldots, b_{n/2})$
11: $(c_{n/2+1}, \ldots, c_n) \leftarrow$ RefreshMasks$(b_{n/2+1}, \ldots, b_n)$
12: **for** $i = 1$ **to** $n/2$ **do**
13: $\quad r_i \leftarrow^{\$} \{0,1\}^k$
14: $\quad d_i \leftarrow c_i \oplus r_i$
15: $\quad d_{n/2+i} \leftarrow c_{n/2+i} \oplus r_i$ $\{d_i \oplus d_{n/2+i} = c_i \oplus c_{n/2+i}\}$
16: **end for**
17: **return** $(d_1, \ldots, d_n)$

Correctness.

The correctness of Algorithm 14 is straightforward to verify recursively. Namely for all 4 blocks in Fig. 3.6, the xor of the outputs is the same as the xor of the inputs. Therefore globally we must have $\bigoplus_{i=1}^{n} d_i = \bigoplus_{i=1}^{n} a_i$.

Complexity.

Let $T(n)$ be the complexity of RefreshMasks. We have $T(n) \leq 2 \cdot T(n/2) + c \cdot n$ for some constant c. One can show recursively that $T(n) \leq 2^i \cdot T(n/2^i) + i \cdot c \cdot n$, which implies $T(n) = \mathcal{O}(n \cdot \log n)$.

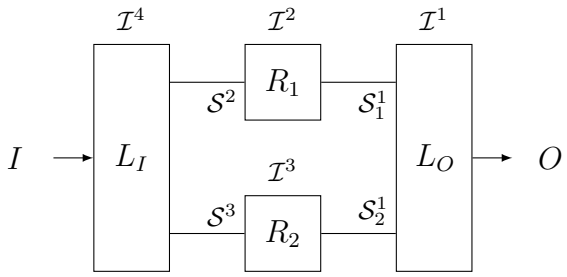


Figure 3.7 – RefreshMasks as composition of gadgets

Security.

The following lemma shows that our new RefreshMasks algorithm achieves the same property as the previous RefreshMasks, namely t-SNI security.

Lemma 3 (t-SNI of RefreshMasks)**.** *Let $(a_i)_{1\leq i\leq n}$ be the input shares of the* RefreshMasks *algorithm, and let $(d_i)_{1\leq i\leq n}$ be the output shares. For any set of t intermediate variables and any subset $|O| \leq t_O$ of output shares such that $t + t_O < n$, there exists a subset I of indices with $|I| \leq t$, such that the distribution of those t intermediate variables as well as the output shares $d_{|O}$ can be perfectly simulated from $a_{|I}$.*

Proof. The proof is based on the two following simple lemmas.

Lemma 4. *Let $a_1, a_2 \in \{0,1\}^k$ be inputs, and let $r \leftarrow^{\$} \{0,1\}^k$. Let $\mathcal{I}$ be a subset of the variables $\{a_1, a_2, r\}$ and let $\mathcal{O}$ be a subset of the variables $\{a_1 \oplus r, a_2 \oplus r\}$. Then the variables in $\mathcal{I} \cup \mathcal{O}$ can be perfectly simulated from $I_1 \in \{\emptyset, \{a_1\}\}$ and $I_2 \in \{\emptyset, \{a_2\}\}$, with $|I_1| \leq t_1 + |\mathcal{O}|/2$ and $|I_2| \leq t_2 + |\mathcal{O}|/2$, for some positive integers t_1, t_2, with $t_1 + t_2 \leq |\mathcal{I}|$.*

Proof. If $|\mathcal{O}| = 2$ or $|\mathcal{I}| \geq 2$ then we can let $I_1 = \{a_1\}$ and $I_2 = \{a_2\}$. If $|\mathcal{O}| = 1$ and $|\mathcal{I}| = 0$ then the variable in $\mathcal{O}$ can be perfectly simulated by a random value. If $|\mathcal{O}| = 1$ and $|\mathcal{I}| = 1$, assume without loss of generality that $\mathcal{O} = \{a_1 \oplus r\}$; then if $\mathcal{I} = \{a_1\}$ or $\mathcal{I} = \{r\}$, we let $I_1 = \{a_1\}$; if $\mathcal{I} = \{a_2\}$, we let $I_2 = \{a_2\}$, and we can perfectly simulate $a_1 \oplus r$ with a random value. If $|\mathcal{O}| = 0$ then the simulation is straightforward. □ □

Lemma 5. *Let $a_1, a_2 \in \{0,1\}^k$ be inputs, and let $r \leftarrow^{\$} \{0,1\}^k$. Let $\mathcal{I}$ be a subset of the variables $\{a_1, a_2, r\}$ and $\mathcal{O} \in \{\emptyset, \{a_1 \oplus r\}\}$. Then the variables in $\mathcal{I} \cup \mathcal{O} \cup \{a_2 \oplus r\}$ can be perfectly simulated from $I \subset \{a_1, a_2\}$, with $|I| \leq 2 \cdot |\mathcal{O}| + |\mathcal{I}|$.*

Proof. If $|\mathcal{O}| = 1$ or $|\mathcal{I}| \geq 2$ we can take $I = \{a_1, a_2\}$. If $|\mathcal{O}| = 0$ and $|\mathcal{I}| = 0$, we can simulate $a_2 \oplus r$ with a random value. If $|\mathcal{O}| = 0$ and $|\mathcal{I}| = 1$, if $\mathcal{I} = \{a_1\}$ we let $I = \{a_1\}$ and we can again simulate $a_2 \oplus r$ with a random value; if $\mathcal{I} = \{r\}$ or $\mathcal{I} = \{a_2\}$ then we let $I = \{a_2\}$. □ □

We proceed with the proof of Lemma 3, by recurrence on the number n of input shares. The case $n = 2$ is straightforward, and similar to Lemma 4. Namely if $t_O = 1$ then $t = 0$ and thanks to the random mask r at Line 2 we can simulate the corresponding output variable with a random value; if $t = 1$ then $t_O = 0$ and the simulation of the probed intermediate variable is straightforward.

We now consider the algorithm with n shares. We label the gadgets from 1 to 4 starting from right to left, with $\mathcal{I}^i$ the corresponding probed intermediate variables (see Fig. 3.7). We start with the rightmost gadget L_O, with $\mathcal{I}^1$ the corresponding set

of probed intermediate variables. For simplicity we can assume that within $\mathcal{I}^1$ only the intermediate variables r_i, c_i and $c_{n/2+i}$ are probed, and not the output variables $c_i \oplus r_i$ and $c_{n/2+i} \oplus r_i$, since such output variables can be equivalently obtained from $\mathcal{O}$, for a smaller value of t, and therefore a stronger bound for $|I|$.

By applying Lemma 4 on each set of intermediate variables $\{c_i, c_{n/2+i}, r_i\}$ and output variables $\{c_i \oplus r_i, c_{n/2+i} \oplus r_i\}$ for all $1 \leq i \leq n/2$, and summing the inequalities, we obtain that the gadget L_O can be perfectly simulated from two sets of input indices $\mathcal{S}_1^1 \subset \{1, \ldots, n/2\}$ and $\mathcal{S}_2^1 \subset \{n/2+1, \ldots, n\}$, such that

$$|\mathcal{S}_1^1| \leq t_1 + t_O/2, \quad |\mathcal{S}_2^1| \leq t_2 + t_O/2 \tag{3.21}$$

for some positive integers t_1, t_2, with $t_1 + t_2 \leq |\mathcal{I}^1|$.

We now consider the R_1 and R_2 gadgets, which are recursive applications of RefreshMasks with $n/2$ shares. The t-SNI condition $t + t_O < n$ for applying Lemma 3 on R_1 is therefore:

$$|\mathcal{I}^2| + |\mathcal{S}_1^1| < n/2 \tag{3.22}$$

and when such condition is satisfied we have from Lemma 3 that the probed intermediate variables in $\mathcal{I}^2$ and the output variables in $\mathcal{S}_1^1$ can be simulated from a subset of input indices $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |\mathcal{I}^2|$. Similarly the t-SNI condition for gadget R_2 is:

$$|\mathcal{I}^3| + |\mathcal{S}_2^1| < n/2 \tag{3.23}$$

When such condition is not satisfied for R_1 or R_2 we say that the corresponding gadget is *saturated*. In that case, to properly simulate the probed intermediate variables and output variables of the gadget, one must know *all* inputs of the gadget. As will be seen below, the proof of Lemma 3 is based on the fact that the two gadgets R_1 and R_2 cannot be both saturated, that is at least R_1 or R_2 must be non-saturated.

Namely consider the following two inequalities:

$$|\mathcal{I}^2| + t_1 + t_O/2 < n/2 \tag{3.24}$$
$$|\mathcal{I}^3| + t_2 + t_O/2 < n/2 \tag{3.25}$$

Using $|\mathcal{S}_1^1| \leq t_1 + t_O/2$, we have that Inequality (3.24) implies Condition (3.22) for gadget R_1; similarly using $|\mathcal{S}_2^1| \leq t_2 + t_O/2$, we have that Inequality (3.25) implies Condition (3.23) for gadget R_2. Moreover, at least one of the two inequalities (3.24) or (3.25) must be satisfied, since otherwise, using $t_1 + t_2 \leq |\mathcal{I}^1|$:

$$n \leq |\mathcal{I}^2| + t_1 + t_O/2 + |\mathcal{I}^3| + t_2 + t_O/2 \leq |\mathcal{I}^2| + |\mathcal{I}^3| + |\mathcal{I}^1| + t_O \leq t + t_O$$

which contradicts the bound $t + t_O < n$. As mentioned previously, this shows that the gadgets R_1 and R_2 cannot be both saturated.

If both inequalities (3.24) and (3.25) are satisfied, then both gadgets R_1 and R_2 are non-saturated, and by recursively applying Lemma 3 on both gadgets, we get $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ and $|\mathcal{S}^3| \leq |\mathcal{I}^3|$. One can therefore let $I = \mathcal{S}^2 \cup \mathcal{S}^3$ and simulate the L_I gadget as in the real circuit[3]; we then have as required:

$$|I| \leq |\mathcal{S}^2| + |\mathcal{S}^3| \leq |\mathcal{I}^2| + |\mathcal{I}^3| \leq t$$

Assume now wlog that (3.24) is satisfied and (3.25) is not. Then R_1 is non-saturated and we can apply Lemma 3 on R_1, which gives as previously $|\mathcal{S}^2| \leq |\mathcal{I}^2|$. For R_2 we

[3]When both R_1 and R_2 are non-saturated, the L_I gadget is actually useless.

cannot necessarily apply Lemma 3, so we must take $\mathcal{S}^3 = \{n/2+1, \ldots, n\}$, which means that all inputs of R_2 must be simulated. We now consider the L_I gadget. We can assume wlog that all intermediate variables probed in $\mathcal{I}^4$ are of the form a_i, $a_{n/2+i}$ and r_i, since all output variables $a_i \oplus r_i$ and $a_{n/2+i} \oplus r_i$ can be probed in $\mathcal{I}^2$ and $\mathcal{I}^3$. By applying Lemma 5 for all $1 \leq i \leq n/2$ on each set of intermediate variables $\{a_i, a_{n/2+i}, r_i\}$ and output variable $a_i \oplus r_i$, where all output variables $a_{n/2+i} \oplus r_i$ must be simulated (since R_2 is saturated), and by summing the inequalities, we construct $I \subset \{1, \ldots, n\}$ such that:

$$|I| \leq 2 \cdot |\mathcal{S}^2| + |\mathcal{I}^4| \leq 2 \cdot |\mathcal{I}^2| + |\mathcal{I}^4| \tag{3.26}$$

It remains to show that $|I| \leq t$. Since by assumption (3.24) is satisfied and (3.25) is not, we have:

$$|\mathcal{I}^2| + t_1 + t_O/2 < n/2 \leq |\mathcal{I}^3| + t_2 + t_O/2$$

which gives using $t_2 \leq |\mathcal{I}^1|$:

$$|\mathcal{I}^2| \leq |\mathcal{I}^3| + t_2 \leq |\mathcal{I}^3| + |\mathcal{I}^1|$$

Then from (3.26) we obtain:

$$|I| \leq 2 \cdot |\mathcal{I}^2| + |\mathcal{I}^4| \leq |\mathcal{I}^2| + |\mathcal{I}^3| + |\mathcal{I}^1| + |\mathcal{I}^4| \leq t$$

as required, which terminates the proof of Lemma 3. □ □

Remark 3. *One can argue that our construction is somewhat minimal. Namely if we remove the pre-processing layer L_I, then the adversary can let $\mathcal{O} = \{1, \ldots, n/4\} \cup \{n/2+1, \ldots, 3n/4\}$, which implies that the first $n/4$ output variables of R_1 must be simulated; the adversary can then probe the remaining $n/4$ output variables of R_1, which implies that R_1 is now saturated. Then without L_I one must have $|I| = n/2$; with $|\mathcal{I}| = n/4$ this contradicts the bound $|I| \leq |\mathcal{I}|$, while with $|\mathcal{O}| = n/2$ the condition $|\mathcal{I}| + |\mathcal{O}| < n$ is still satisfied.*

Similarly, if we remove the post-processing layer L_O, then the adversary can saturate R_1 by letting $\mathcal{O} = \{1, \ldots, n/2\}$. Then all inputs of R_1 must be known, and for each intermediate probe at the input of R_2, one must add two additional indices in I, which contradicts the bound $|I| \leq t$.

Remark 4. *Using the above* RefreshMasks *algorithm for our countermeasure of Section 3.2.8, its complexity becomes $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^2 \cdot \log n)$, hence the same complexity as the original ISW countermeasure.*

3.2.10 Conclusion

Since the publication of the private circuit transformers, several papers have investigated their security under some assumptions about the adversary capability and the amount of noise in the information leakages. This eventually led Duc *et al.* [70] to state at Eurocrypt 2015 that the number of leakage observations needed to significantly reduce the guessing entropy of the secret parameter of an implementation protected with sharing at order n is bounded below by $\mathcal{O}(\sigma^{2n})$, where σ is the minimum instantaneous standard deviation of the noise in the observations. However, for this security bound to apply, the amount of information leaking in each observation must decrease when n increases, which implies that the noise standard deviation must increase accordingly. This condition is a strong limitation when the amount of noise cannot be

accurately controlled by the security designers. This paper questions the security of an implementation when the above bound does not apply, which occurs when the number of shares n is greater than $c \cdot \sigma^2$ (which occurs *e.g.* when the noise is constant and n increases). In particular, we exhibit two (template) horizontal side-channel attacks against the Rivain-Prouff's secure multiplication scheme [146] and we analyze their efficiencies thanks to several simulations and experiments. We argue that they are efficient iff the number of shares n is greater than $c \cdot \sigma^2$ for some constant c, which implies that the condition in Duc *et al.*'s paper is not only sufficient but also necessary. Eventually, we propose an improved version of Rivain-Prouff's secure multiplication which is secure against our attacks.

In the next section we analyze a suggestion for a side-channel countermeasure for symmetric ciphers suggested by Bringer *et al.* in [40] that can achieve security against t-th order statistical attacks and security against faults at once. This work challenges the countermeasure security and shows practical attacks exploiting simple side-channel analysis and 1-st order statistical attacks.

3.3 Security Analysis of the Orthogonal Direct Sum Masking

In this section, we introduce the Orthogonal Direct Sum Masking (ODSM) countermeasure as it was defined in [40]. We then detail how the authors apply it in the case of an AES implementation.

In [41], Bringer *et al.* introduced a masking scheme based on a specific encoding of the sensitive data and the corresponding mask. Following this scheme, the masking of sensitive data x with the random quantity m is obtained by computing $z = xF \oplus mG$, where:

- G is a generator matrix of the binary linear code $\mathcal{C}$ of length n, dimension k and minimum distance d
- F is a $k \times n$ matrix with k rows in $\mathbf{F}_2^n$ all linearly independent of each other and not belonging to the binary linear code $\mathcal{C}$.

Recovering x from the encoded word z can be achieved by multiplying z by the parity-check matrix of $\mathcal{C}$: $zH^T = xFH^T \oplus mGH^T = xFH^T$. The authors then describe an application of this scheme to the AES.

However, as pointed out by Moradi in [128] certain limitations exist when choosing the matrix F in order to retrieve the sensitive value x from xFH^T. Namely, one should ensure that the application $x \mapsto xKH^T$ is bijective, *ie.* $(KH^T)^{-1}$ shall exist.

In addition, the author stresses that the application of the scheme to the AES requires a mask correction at each round of the cipher algorithm.

Finally, the work of Azzi *et al.* in [5] adapts the ODSM scheme to enhance the fault detection capability through non-linear functions. However this comes at the cost of additional computations with regards to the side-channel resistance property. We now describe the original ODSM scheme of Bringer *et al.*.

3.3.1 Orthogonal Direct Sum Masking Scheme

As previously stated, the construction of the ODSM lies on the fact that for the considered code $\mathcal{C}$, we have $\mathcal{C}^{\perp} \oplus \mathcal{C} = \mathbb{F}_2^n$. Indeed, in this case we have $G \cdot H^T = 0$ and H is the parity matrix of $\mathcal{C}$. Consequently, in (1.4) we can recover x and y from z:

$$x = zG^T(GG^T)^{-1} \tag{3.27}$$

$$y = zH^T(HH^T)^{-1} \tag{3.28}$$

The principle of the masking scheme consists in representing a sensitive k-bit data x by a n-bit data z according to (1.4), where y is an $(n-k)$-bit random mask.

The sensitive value x can be easily recovered from z by using (3.27). In addition, the integrity of the manipulated data z can be verified as often as required by checking the integrity of the mask y thanks to (3.28), which provides the security against fault injection.

Based on this principle, the authors suggest to perform computation within the encoded/masked representation. Actually they show how applying operations on the sensitive value x can be achieved by applying associated operations on the encoded value z. To reach this goal, they split the different operations required into three categories and show how to proceed in each one:

2-operand operations. In this case, they focus their attention on the xor operation. Actually this case is quite straightforward since it is only necessary to encode the operand and perform the xor. For instance, supposing we need to xor a round key k_i to x, we would have to compute $z' = z \oplus k_i G$. And we can check that $x \oplus k_i$ is computed within the masking scheme:

$$z' = z \oplus k_i G = (x \oplus k_i)G \oplus yH \tag{3.29}$$

Binary linear operations. Let L be the matrix corresponding to the desired binary linear operation, then they suggest to construct a so-called *masked* binary linear operation whose corresponding matrix L' is constructed as follows:

$$L' = G^T(GG^T)^{-1}LG \oplus H^T(HH^T)^{-1}H \tag{3.30}$$

And we can check again that xL is correctly computed within the masking scheme when zL' is computed.

Nonlinear transformations. In this case, a *masked* version S' of the transformation S can be computed:

$$\forall z \in \mathbb{F}_2^n,\ S'(z) = S(zG^T(GG^T)^{-1})G \oplus zH^T(HH^T)^{-1}H \tag{3.31}$$

$S(x)$ is correctly obtained within the masking scheme when computing $S'(z)$.

Following these guidelines, the authors claim that various computations can be carried out within the coset $\mathcal{C} \oplus d$ of the linear code $\mathcal{C}$, with d a mask randomly chosen in $\mathcal{D} = \mathcal{C}^{\perp}$.

ODSM in Practice: Application to the AES

The ODSM can be applied, in particular, to an implementation of the AES. For that purpose, the authors consider the 128-bit version of the cipher and propose to use the binary linear code of parameters [16,8,5] (meaning $n = 16$ and $k = 8$) which has a supplementary dual in $\mathbb{F}_2^{16}$.[4] Indeed, once the initial state has been encoded, the different operations of the AES can be straightforwardly constructed as previously described.

AddRoundKey. The round key bytes only need to be encoded before being added to the encoded state: $z = z \oplus kG$.

ShiftRows. The ShiftRows operation remains unchanged. It only processes 16-bit words instead of 8-bit.

MixColumns. MixColumns can be computed by using the linear application generated from the matrix of the XTIME application by (3.30).

SubBytes. For the SubBytes operation, two approaches were proposed in [40]. The first one is a look-up table approach which requires to precompute the 16-bit output $S'(z)$ for all z as per (3.31). We note that this method implies a quite heavy memory overhead as a 128kB-table needs to be stored (2^{16} 16-bit values). The second approach actually performs the SubBytes outside of the code and thus involves the recomputation of a new masked S' transformation for each encryption to ensure a proper masking. It is then necessary to compute for all x in $\mathbb{F}_2^k$:

$$S'_{recomp}(x) = S(x \oplus x') \oplus x'', \quad \textit{with } x' \textit{ and } x'' \textit{ randomly chosen in } \mathbb{F}_2^k \tag{3.32}$$

The SubBytes is then performed as described in Alg. 15. Our analysis is actually

Algorithm 15: Masked SubBytes transformation on $z = xG \oplus yH$

$z = z \oplus x'G$;
$x = zG^T(GG^T)^{-1}$;
$x = S'_{recomp}(x)$;
$z' = xG \oplus yH$;
$z' = z' \oplus x''G$;
return z';

independent of the choice of either of the two approaches.

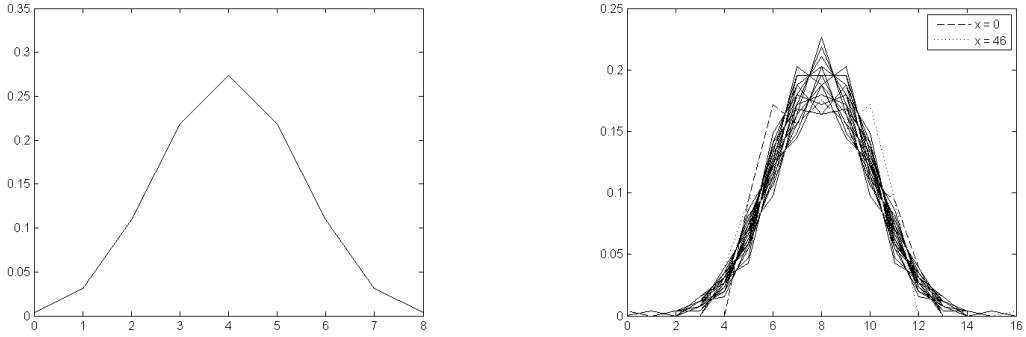
3.3.2 Side-Channel Analysis of the Masking Scheme

In this section we provide a deep analysis of the side-channel resistance of the masking scheme suggested in [40]. We demonstrate that it is possible to mount a first-order side-channel attack versus the countermeasure meant to be resistant to high-order attacks.

[4] For the generating and parity matrices G and H and for L and L' corresponding to the standard and *masked* versions of the XTIME linear application of this code, we refer the reader to [40].

Striking Differences

The authors of [40] proved the security of the ODSM scheme versus d-th order side-channel analysis, where $d+1$ is the distance of the dual code $\mathcal{C}^{\perp}$. Thus, with respect to the parameters of [40], the code is proved to be secure versus 4-th order attacks, 5 being the minimal distance of the dual code. The proof of security relies on the observation that the expected value of the leakage is independent of the sensitive value x (up to the 4^{th} order statistical moments), after the encoding $xG \oplus yH$ with mask y.



(a) Boolean masking pdf.

(b) ODSM masking pdf.

Figure 3.8 – pdf of the Hamming weights of the Boolean masking scheme vs ODSM masking scheme.

Figure 3.14 shows the expected probability density functions (pdf) of the Hamming weight (HW) of masked values for both boolean masking and the ODSM scheme. The results are obtained by collecting the distribution of the HW of $z = x \oplus y$ for Boolean masking (Figure 3.8a), and $z = xG \oplus yH$ for ODSM, where x is fixed, and y takes all values in $\mathbb{F}_2^k$ (Figure 3.8b).

As expected, in the boolean masking scheme the distributions are independent of the sensitive values, such that all distributions are superposed and only one curve is visible. On the other hand, in the ODSM scheme the distributions depend on the sensitive values and we can distinguish 22 different distributions, each one related to a particular set of sensitive values. In particular the distributions of the sensitive value $x = 0$ and $x = 46$ show striking differences. We remark that an encoded value with HW of 0 can only be produced by encoding the sensitive value $x = 0$ with a mask $y = 0$. Similarly a HW of 16 can only be obtained when the encoded sensitive value equals 46. From Figure 3.8b we thus observe that the extremum HW value 0 (resp. 16) is only present on the distribution of $x = 0$ (resp. $x = 46$). We further remark that for a sensitive encoded value equal to $x = 0$ (resp. $x = 46$), the HW of the encoded value can never be $4, 3, 2$ or 1 (resp. $12, 13, 14, 15$).

We show in the following that it is possible to exploit such striking differences by using 1^{st}-order statistics in order to retrieve the sensitive values.

Means-only attack on the ODSM distribution

We have noticed that the difference between the leakage distributions of the ODSM masking scheme and that of classical boolean masking scheme may lead to weaknesses that have not been taken into account in [40]. In this section we exhibit an actual attack that exploits these very differences to retrieve the key value by using only 1^{st}-order statistics on carefully selected leakages.

The basic idea of our attack comes by observing the distributions of Figure 3.8b. The distributions present a left skewness (i.e.: asymmetry about the mean) for $x = 0$, and a right skewness when $x = 46$. While such skewness do not bias the average of all values, it does when the average is computed only on the leakages below a given Hamming weight. In practice we exploit the fact that the skewness preserves the mean only when computed on all values, but produces detectable biases on subsets of them.

Observing Figure 3.8b one notice that the mean of all leakage values below 9 and those above 9 are not equals for all the 22 classes, in particular $z = 0G \oplus yH$ and $z = 46G \oplus yH$ should present remarkable differences due to the skewness. We thus argue that it provides a distinguisher for the values 0 and 46.

A more careful partitioning leads to even better and more accurate results. We divide the curves into two sets:

- a first set containing leakages with Hamming weight between 4 and 11,
- a second set for the remaining leakages.

The absolute difference of the two sets on theoretical distributions is depicted in Figure 3.9, for each message, for all masks.

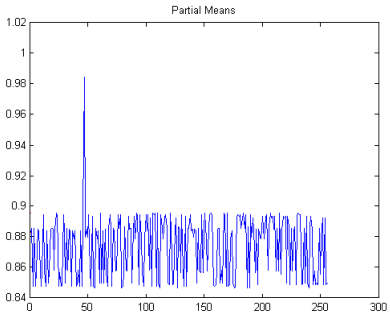


Figure 3.9 – Difference of the number of leakages between 4 and 11 and the rest.

We further remark that this choice for the two sets allows to retrieve only the value corresponding to 46, as the skewness on 0 is not captured by such partitioning.

Simulations

In this section we show the results of the application of our attack described in Section 3.3.2 to simulated leakages of the ODSM scheme. For our simulations we computed the value $z = \text{Sbox}(m \oplus k_i)G \oplus yH$, where y is in $\mathbb{F}_{2^k}$ freshly regenerated at each execution. For each value z we computed the corresponding leakage $\vec{\ell} = HW(z) + B$, where B is a Gaussian noise with standard deviation σ. In order to evaluate the success rate of our attack with different noise levels, we have performed different campaigns where σ varies from 0 to 2. For each campaign we have simulated the leakage of $10,000$ computations for each byte value.

We start our attack by computing the minimum and the maximum values among all leakages. Then define the range s of all leakages as the difference between the maximum and minimum values, divided by 16. We then use this value to split the leakages into the two sets, the first containing those leakages whose value falls between $s*4$ and $s*11$ and the second with the remaining leakages. We finally analyze the absolute difference of the two sets for each message. As observed in Figure 3.9, only the message m which

Figure 3.10 – Result of the attack with $\sigma = 0$ for $10,000$ leakages.

Figure 3.11 – Result of the attack with $\sigma = 1$ for $10,000$ leakages.

gives $\text{Sbox}(m \oplus k_i) = 46$ should produce a peak on the difference of means. Thus the peak found for a particular message m reveals $\text{Sbox}(m \oplus k_i) = 46$, so the attacker can retrieve the secret key byte $k_i = \text{Sbox}(46)^{-1} \oplus m$.

The key byte value used for our simulations is 43, thus we expect to obtain peaks for the message of value $233 = \text{Sbox}(46)^{-1} \oplus 43$. Figure 3.10 to Figure 3.12 show the results obtained by using $10,000$ noisy executions for each message m. The left part corresponds to the value of the absolute differences for each message (thus for each of the 256 values we depict the value of the difference of means). The right part depicts the maximum value of the absolute difference for each key, sampled after each 100 curves. For right-side figures, the correct message hypothesis (233) is depicted in red.

We present the pseudocode of our attack in Alg. 16.

We notice that our attack needs a huge number of curves to retrieve the key value even for relatively low noise simulations. For example, for $\sigma = 1$, we need about $384,000$ curves $(1500 * 256)$ in order to retrieve the correct key hypothesis. The need for a considerable number of curves can be interpreted as a consequence of the masked values living in $\mathbb{F}_{2^16}^2$, and thus far more samples are required to obtain a representative sample of the underlying distribution. However, we remark that as soon as there is no noise, very few hundred traces are necessary to retrieve the correct key.

We want to stress the fact that despite the security proof given in [40], our attack shows that it is possible to retrieve the secret values protected with the ODSM scheme by using a first-order statistic on carefully selected leakages.

Figure 3.12 – Result of the attack with $\sigma = 2$ for $10,000$ leakages.

Algorithm 16: MEANS ATTACK ON AES-128 ODSM SCHEME.

```
// Find min and max values
l_max = max_{m∈F_2^k, 0≤i<♯curves}(leakage(m, i));
l_min = min_{m∈F_2^k, 0≤i<♯curves}(leakage(m, i));
// Derive HW boundaries
leakage_size=(l_max - l_min) / 16;
set1_limit = leakage_size × 4;
set2_limit = leakage_size × 11;
// Separate curves
for m from 0 to 255 do
    for i from 0 to 10,000 do
        if set1_limit ≤ leakage(m, i) ≤ set2_limit then
            set2(m)+= leakage(m, i);
        else
            set1(m)+= leakage(m, i);
        end
    end
end
// Select best candidate based on difference of means
best_message = max_m(abs(set2(m) − set1(m))/10,000);
return k = Sbox(46)^{-1} ⊕ best_message
```

We finally insist on the fact that despite our attack applies here to the ODSM scheme with the parameters of [40], most choices of the code would succumb to such an attack.

3.3.3 Maximum likelihood attack

In this section we present a further attack to the countermeasure. As we have remarked in Sec. 3.3.2, the leakage corresponding to an HW of 0 can only be produced if both the sensitive value x and the mask y equal to 0. We thus suggest that it is possible to use a maximum likelihood (template) attack to distinguish the curves manipulating a variable z whose HW equals 0 from the others.

Template attacks are generally divided into two phases. In the first phase (profiling) the sensitive data is known to the attacker, for example she may employ an open sample, while in the second phase (attack) she tries to recover some unknown sensitive data by using new observations and the information collected during the profiling phase.

More formally, let us assume that the attacker retrieves a set of observations of the random variable z, where each observation has the form $\vec{\ell} = \varphi(z) + B$, where φ is an unknown function and B a Gaussian noise with standard deviation σ. She can estimate the expectation $\vec{\mu}_0$ and covariance Σ_0 of $\vec{\ell}$ when $z = 0$ and $\vec{\mu}_1, \Sigma_1$ when $z \neq 0$.

For a Gaussian distribution of expectation μ and covariance matrix Σ, the probability density function (pdf) of $\vec{\ell} \in \mathbb{R}^t$ is defined as:

$$f(\vec{\ell}) = \frac{1}{\sqrt{(2\pi)^t \det(\Sigma)}} \exp\left(-\frac{1}{2}(\vec{\ell} - \vec{\mu})' \cdot \Sigma^{-1} \cdot (\vec{\ell} - \vec{\mu})\right) . \tag{3.33}$$

So by evaluating $f_{l|z=0}$ and $f_{l|z\neq 0}$ she obtains the likelihood that $z = 0$ was the manipulated value.

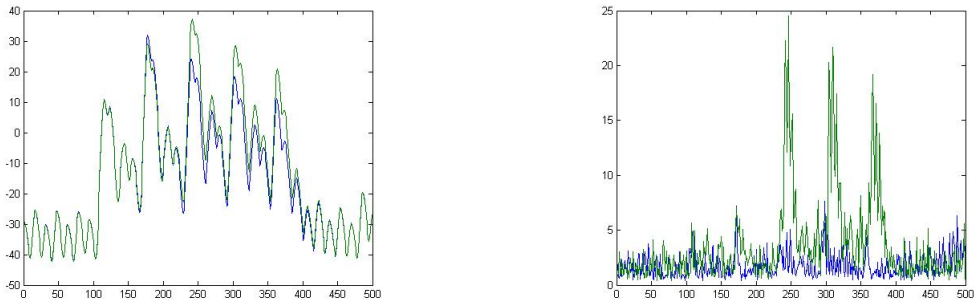


Figure 3.13 – Expectation and variance of real leakages when $z = 0$ and $z \neq 0$.

Experiments

We have tested the template attack against a real device and we provide in this section the results of our experiments.

The target of our experiment is an ATMega328P device. We have by-passed the decoupling capacitors of the device which may filter out useful signals. We have then connected the oscilloscope to the device and measured the difference of potential at the ends of a resistor placed between the ground pin of the ATMega328P and the ground of the device. We have finally pre-filtered the input of the oscilloscope at 20Mhz and sampled the data at 100Mhz. Our settings allow to obtain small curves while keeping as much information as possible.

We have then collected $250,000$ leakages where we controlled the value of the mask. A random bit was used to select if $z = 0$ or $z \neq 0$ was used by the implementation. Knowledge of the random bit allowed us to split the set of acquisition between those with $z = 0$ and those with $z \neq 0$ to build templates. We also used this knowledge to verify the confidence of the likelihood distinguisher.

We show in Figure 3.13 the differences between the expectation of the two sets. It is possible to distinguish important differences between them, in particular around points 250 and 300.

During attack phase we have acquired $250,000$ more curves and tried to separate them into two sets. The knowledge of the value of the random bit allowed us to verify the success rate of the distinguisher. We have used 80 points to estimate the pdfs of Eq. 3.33. These points were chosen as those providing the highest variance between the expectations of the two sets. In this settings we obtained 99.84% of correct detection rate.

Our attack thus demonstrates that even for real devices it is possible to break the countermeasure by using only first and second order statistical attacks.

3.3.4 Possible Fix-Ups and Residual Issues

As shown in Section 3.3.2 and Section 3.3.3 the differences between the distributions of the Hamming weight of masked values for different inputs can be exploited by an attacker to recover manipulated secrets. In this section we propose a method to improve the resistance of the scheme to the attacks that we have presented in this work while preserving the fault detection capability.

Conservative Shuffling

In this section we suggest how to add algorithmic noise to the ODSM scheme in order to defeat the attack introduced previously. We show that our countermeasure preserves the fault detection capability and the possibility to perform computation within the masking scheme.

Our method relies on shuffling the generating matrices G and H of the code $\mathcal{C}$ and $\mathcal{D}$, losing the systematic form of $\mathcal{C}$'s generating matrix. Successively applying permutations on the columns of these matrices allows to randomize the mappings between elements of $\mathbb{F}_2^k$ and $\mathcal{C}$ (resp. $\mathbb{F}_2^{n-k}$ and $\mathcal{D}$) by randomizing the codewords of $\mathcal{C}$ (resp. $\mathcal{D}$) itself. We can note that the properties of the associated codes remain unchanged as we only reorder the columns of the generating matrices. In particular the duality between $\mathcal{C}$ and $\mathcal{D}$ is preserved since we apply the same permutation on both matrices G and H. This can be seen by recalling that any permutation of n columns of a $k \times n$ matrix can be realized by multiplying from the right this matrix by a permutation matrix P. Further recalling that P is orthogonal ($PP^T = \mathbb{I}$), it comes straightforwardly that :

$$GP(HP)^T = GP(P^T H^T) = G(PP^T)H^T = GH^T = 0 \tag{3.34}$$

Such a process can be easily achieved at the cost of up to 12 bits of random used to select two columns to permute and an amount of circular shift. For the XOR operation, the permutation can be straightforwardly applied to the encoding of the operand: $z' = z \oplus k_i GP$. For the linear operation, the permutation needs to be reflected on the L' matrix of (3.30):

$$\begin{aligned} L'' &= (GP)^T(GP(GP)^T)^{-1}LGP \oplus (HP)^T(HP(HP)^T)^{-1}HP && (3.35)\\ &= P^T(G^T(GG^T)^{-1}LG)P \oplus P^T(H^T(HH^T)^{-1}H)P && (3.36)\\ &= P^T(G^T(GG^T)^{-1}LG \oplus H^T(HH^T)^{-1}H)P && (3.37)\\ &= P^T L' P && (3.38) \end{aligned}$$

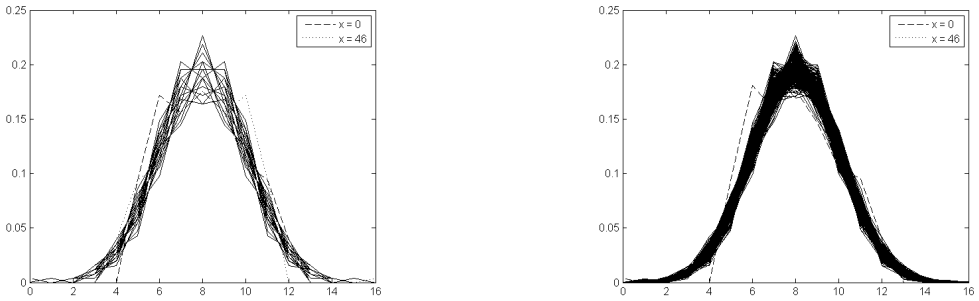
For the non-linear operation, only the table recomputation approach seems to be achievable with reasonable overhead as the look-up table approach would require to recompute the S' table for all z. Alg. 15 should then be adapted as exposed in Alg. 17.

Algorithm 17: Masked SubBytes transformation on $z = xGP \oplus yHP$

$z = z \oplus x'GP$;
$x = z(GP)^T(GG^T)^{-1}$;
$x = S'_{recomp}(x)$;
$z' = xGP \oplus yHP$;
$z' = z' \oplus x''GP$;
return z';

Using such shuffled matrices for each encryption, we obtain the distribution depicted in Figure 3.14b. Figure 3.14a is recalled for comparison purpose.

We can see that the distribution gives a much less explicit hint on the manipulated value compared to the original one. However we can still observe differences in the distributions for each value, supporting a residual weakness. Nevertheless, the attack described in Section 3.3.2 now fails even with a low noise level as can be seen in Figure 3.15.



(a) Original ODSM pdf. (b) Shuffled ODSM pdf.

Figure 3.14 – Hamming weights' pdf of encoded values for ODSM Vs Shuffled ODSM.

Figure 3.15 – Result of the attack with $\sigma = 0$ and $\sigma = 1$ for $10,000$ leakages.

Residual Issue: encoding 0

From the observation of Figure 3.14a and Figure 3.14b we can see that one potential weakness of the original scheme is not taken care of by our method. Indeed, we observe that the value $xG \oplus yH =$ 0x0000 can only be obtained when $x = 0x00$.

By assuming that the attacker can detect the manipulation of the value 0x0000 then she directly knows the corresponding value 0x00 of the internal AES state. Such a weakness is not present in traditional boolean masking, where all masked values can be produced by all secret values.

Such an attack may not be merely theoretical since the hypothesis of retrieving the Hamming weight of internal values of more than one byte by SPA has been exploited in recent publications [19, 20] in order to retrieve the operands of a 128-bit scalar multiplication.

We further remark that a similar SPA weakness would affect the ODSM scheme for any choice of code. Indeed, since the codes $\mathcal{C}$ and $\mathcal{D}$ are complementary duals and from the definition of $\mathcal{C}$ and $\mathcal{D}$ we know that:

$$\forall\ z\ \in\ \mathbb{F}_2^n,\ \exists!\ (x, y)\ \in\ \mathbb{F}_2^k \times \mathbb{F}_2^{n-k} \text{ such that } z = xG \oplus yH$$

This holds in particular when $z = 0$, which is thus equivalent to $xG = yH$ and to $x = y = 0$. Consequently even when randomizing G and H, we can only observe a value of null Hamming weight when the sensitive value x *and* the mask y are null. Unfortunately, as we have shown in Section 3.3.3 such weakness may be exploited by using template attacks. We can nevertheless stress that in case the attacker cannot control the value of the mask she may not be able to build the templates and consequently the attack should not work.

3.3.5 Conclusion

The definition of new countermeasures tackling in the same effort side-channel analysis and fault attacks is definitely a challenging task. The ODSM scheme succeeds in providing both a way to detect errors and ensuring the independency of the mean and the variance of the Hamming weight of masked data. However in this work we demonstrate that the distributions of Hamming weights of the ODSM encoded data are actually dependent on the sensitive values being manipulated, which renders the scheme helpless against a side-channel attack considering only 1^{st}-order statistical moment of the observed leakage. Furthermore we have shown that some measures can be taken in order to reduce the leakage exposed when observing the Hamming weight distributions for a given sensitive value, although it turns out that the scheme cannot be made totally SCA-resistant. Still, countermeasures based on coding theory appear as promising candidates to improve the resistance of cryptographic implementations against both side-channel and fault attacks. In particular, the definition of methods allowing to perform the complete execution of an algorithm under the protection of the code is an interesting line of research for future works.

The last work that we present in this chapter is an attack that exploits both fault and side-channel physical analysis. By using a combination of FA and SCA we show that it is possible to break a CRT-RSA countermeasure that has been conceived to resist each of them, but not both together.

3.4 A Combined Fault and Side-Channel Attack on CRT-RSA

The idea to combine SCA and FA first appeared in 2007 when Amiel *et al.* proposed a so-called *Combined Attack* (CA) on an RSA implementation protected against FA and SPA [2].

They noticed that by setting to zero one of the temporary registers used in the Montgomery ladder, its structure becomes unbalanced, revealing the value of the secret exponent by SPA.

Following this publication, three other works have been published taking advantage of this new way of defeating embedded security. Two of them present a CA against a secured AES implementation [148, 149]. The third one focuses on the elliptic curve scalar multiplication [74].

Despite its theoretical effectiveness, the combination of SCA and FA is very difficult in practice, explaining the lack of practical experiments in the current literature.

3.4.1 Context of the Attack

As stated in Sec. 2.1.4 and Sec. 3.1.4, several countermeasures have been developed to protect CRT-RSA embedded implementations against both SCA and FA. In the following, we consider an CRT-RSA algorithm protected:

- against SCA by using message and exponent blinding as suggested in [175], a regular exponentiation algorithm such as the Square Always [53] and a mask refreshing method along the exponentiation such as the one presented in [71]. Moreover, the blinding is kept all along the CRT-recombination.

- against FA by verifying the signature using the public exponent e [34]. In addition, we also use the approach presented in [68] which mainly consists in checking the result of the verification twice to counteract double FA attacks.

Fig. 3.16 depicts the main steps of such an implementation where the k_i's are random values (typically of 64 bits) generated at each execution of the algorithm and S'_p, S'_q and S' represent the blinded version of S_p, S_q and S respectively.

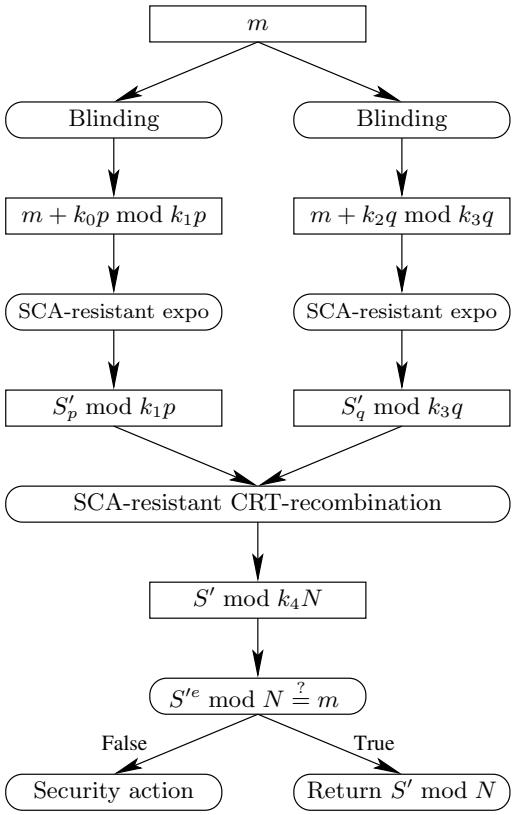


Figure 3.16 – Main steps of a CRT-RSA implementation secure against SCA and FA.

In the following, we assume that the fault injected by the attacker follows either the *bit-flip*, the *stuck-at* or the *unknown constant error* fault models (cf. Sec. 2.1.2). Moreover, we assume the attacker is able to choose which byte of the message is affected by the fault.

As mentioned in Sec. 2.1.4, injecting a fault during the signature computation leads to a faulty signature that allows the attacker to recover the private key. However in the implementation considered in this work, the verification with the public exponent detects such a disturbance and the faulty signature is never revealed to the attacker. The main contribution of this work is to show that in this case, an SCA can still allow the attacker to gain enough information on the faulty signature to recover the private key.

3.4.2 The attack

At first glance, it seems impossible to perform such an attack during the signature process due to the blinding countermeasure. However by observing Figure 3.16, one

may note that the faulty signature $\widetilde{S}$ remains blinded until the end of exponentiation with e modulo N. Therefore if we can express $\widetilde{S}^e \bmod N$ in terms of the message m and of the private key then we can perform an SCA on this value. In the next section, we exhibit such a relation allowing us to mount a CA on an SCA-FA-resistant CRT-RSA implementation.

A Useful Relation

Proposition 5. *If a fault ε is induced in m such that the faulty message $\widetilde{m}$ is equal to $m + \varepsilon$ at the very beginning of the computation of S_p then*

$$\widetilde{S}^e \equiv m + \varepsilon q i_q \bmod N \ , \tag{3.39}$$

where $\widetilde{S}$ corresponds to the faulty signature.

Proof. By definition of the CRT-RSA signature, we have:

$$\begin{cases} \widetilde{S} & \equiv & (m+\varepsilon)^d \bmod p \\ \widetilde{S} & \equiv & m^d \bmod q \end{cases} \tag{3.40}$$

It comes then straightforwardly that:

$$\begin{cases} \widetilde{S}^e & \equiv & m + \varepsilon \bmod p \\ \widetilde{S}^e & \equiv & m \bmod q \end{cases} \tag{3.41}$$

Finally, applying Gauss recombination to (3.41) leads to (3.39) since:

$$\begin{aligned} \widetilde{S}^e &\equiv p i_p m + q i_q (m+\varepsilon) \bmod N && (3.42)\\ &\equiv (p i_p m + q i_q m) + \varepsilon q i_q \bmod N && (3.43)\\ &\equiv m + \varepsilon q i_q \bmod N \ , && (3.44) \end{aligned}$$

where $i_p = p^{-1} \bmod q$. □

One may note that a similar relation holds if m is disturbed at the very beginning of S_q computation due to the symmetrical roles of p and q in both branches of the CRT-RSA. For the sake of simplicity, we will use the case where S_p computation is disturbed in the rest of this work.

Recovering the Private Key

Following the attack's principle depicted in Sec. 3.4.2 and using 5, we will now present in detail the main steps of our attack.

Firstly, the attacker asks the embedded device to sign several messages m_i through a CRT-RSA implemented as described in Sec. 3.4.1. For each signature, the computation of S_q is performed correctly and a constant additive error ε is injected on the message m_i at the beginning of each S_p computation. Then during each signature verification, the attacker monitors the corresponding side-channel leakage $\mathcal{L}_i$ which represents the manipulation of $\widetilde{S}_i^e \bmod N$.

From 5, we know that there exists a sensitive value k satisfying the relation $\widetilde{S}_i^e \bmod N = m_i + k$. Therefore, the attacker will perform a CPA to recover this sensitive value by computing $\rho_k(m_i + k, \mathcal{L}_i)$ for all the possible values of k (cf. Sec. 3.1.4).

Depending on the set $\{(m_i, \widetilde{S}_i^e \bmod N)\}_i$, it follows from Eq. 3.39 that k will be equal either to $\varepsilon q i_q \bmod N$ or to $\varepsilon q i_q \bmod N - N$.

Therefore, the value $\widehat{k}$ producing the strongest correlation at the end of the CPA will be one of these two values. Once $\widehat{k}$ recovered, the attacker must then compute the gcd between $\widehat{k}$ and N, which leads to the disclosure of q. From this value, the private key is straightforwardly computed.

Regarding the practicality of our fault model (i.e. a constant additive fault), one may note that by fixing a small part of the message (e.g. a byte), the disturbance of such a part in either the stuck-at, the bit-flip or the unknown constant fault model results in a constant additive error during the different signature computations. Therefore our fault model is definitely valid if the attacker can choose the messages to sign, or even if she can only have the knowledge of the messages and attack only those with a given common part.

Finally, one may note that it is not possible to perform a statistical attack targeting the full value of k at once due to its large size (i.e. $\lceil \log_2(N) \rceil$ bits). However, one can attack each subpart of this value, for instance by attacking byte per byte starting with the least significant one in order to be able to propagate easily the carry. It is worth noticing that CPA only applies when the corresponding part of the message varies. Therefore, if the attacker fixes the MSB of the message, then the corresponding set of measurements can be used to recover the whole but last byte of $\widehat{k}$. In such a case, a brute force search can be used to recover the missing byte.

In the next section, we present simulations of our attack which prove the efficiency of our method and which are based on the attacker's capability to inject the same fault and on the noise of the side-channel measurements.

3.4.3 Experiments

The success of the attack presented in Sec. 3.4.2 relies on the ability of the attacker to both measure the side-channel leakage of the system during the signature verification and induce the same fault ε on the different manipulated messages.

In order to evaluate the effectiveness of this attack, we have experimented it on simulated curves of the side-channel leakage $\mathcal{L}$, according to the Gaussian noise leakage model (Eq. 3.1). In the framework of our experiments, we consider that the processor manipulates 8-bit words and we use three different levels of noise, namely $\sigma = 0.1$, 1 and 5.

As well as the side-channel leakage, the faults were also simulated by setting the most significant word of the message m to all-0 at the very beginning of the S_p computation.

These faults were induced with a given success rate r, varying in our different experiment campaigns (namely 50%, 10% and 1%).

Depending on the experimental settings, all the different words of the secret value will be equivalently correlated with the simulated curves. The graphs presented in Figure 3.17 present the convergence of the correlation for each possible value k of one particular byte (the 5^{th} least-significant byte) of the secret depending on the number of side-channel measurements with different simulation settings σ and r.

As exposed in Figure 3.17, the number of traces required to recover the secret value depends essentially on the fault injection success rate. This comes from the fact that every wrongly-faulted computation can be considered as noise in the scope of our statistical analysis. The number of curves required to retrieve the secret word grows

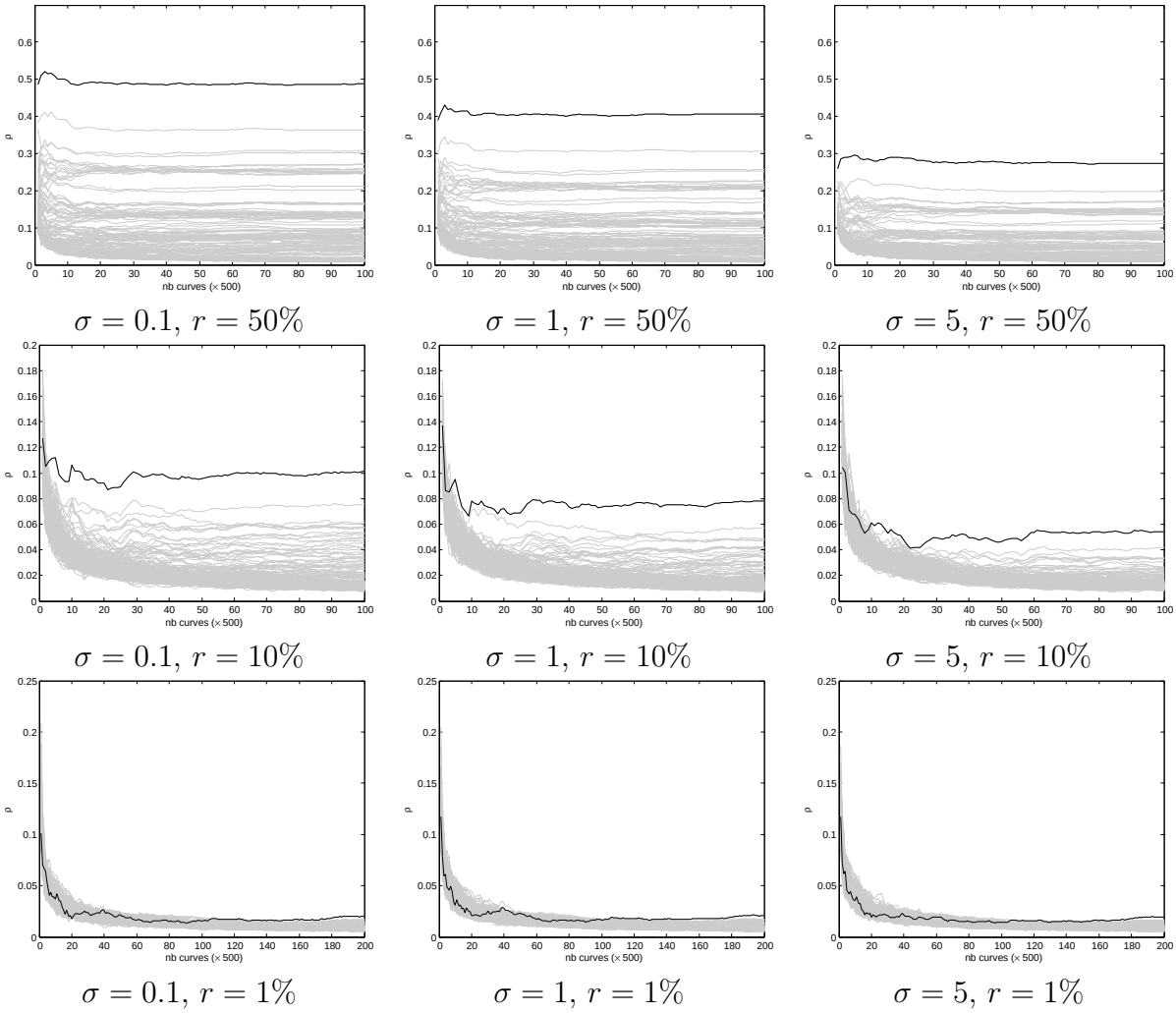


Figure 3.17 – Convergence of the correlation for the 256 possible values k_i for the secret (the correct one being depicted in black) depending on the number of side-channel measurements ($\times 500$) for different levels of noise σ and fault injection success rates r.

as the fault injection success rate decreases and to a fewer extent as the noise of the side-channel leakage increases.

With regards to the results obtained when $\sigma = 5$ and $r = 10\%$, which appear to be plausible values in practice, it took us 3.35 seconds to retrieve one byte of the secret value by performing the CPA on $15,000$ curves of 128 points each[5]. Assuming a genuine curve should be made of at least $5,000$ points, we can estimate the time required to practically perform the attack to about 1 minute 5 seconds per byte. That is to say, it takes about 2 hours 20 minutes to recover the complete secret value if we consider a 1024-bit RSA module.

For the sake of clarity, we restrained the experiments presented here to the case where the processor manipulates 8-bit words, and thus ε is an 8-bit error. The same experiments have been run for processor word-size up to 32 bits with success. Besides, about the same number of curves were necessary for the CPA to highlight the correct secret byte.

The next section shows how it is possible to considerably reduce the complexity of our attack thanks to the use of lattice techniques.

[5]The execution time given here and in Sec. 3.4.5 have been obtained on a 32-bit CPU @3.2GHz

3.4.4 Reducing the Attack Complexity Using Coppersmith's Methods

This section aims at improving the attack complexity using Coppersmith's methods. It is in line with the problem of factorizing N knowing half part of prime p (or q), that was solved in [56]. With respect to our case, we highlight that if the CA presented in Sec. 3.4.2 provides about half of the secret $\varepsilon qi_q \bmod N$, then the other half part can be straightforwardly computed by solving a well-designed modular polynomial equation that we elaborate in the sequel. Besides, we deal with two cases (ε known and unknown), depending on the fault model that is considered.

3.4.5 Bringing Up the Original Problem to Solving a Modular Equation

Suppose we are given the t least significant bits (LSB) of the secret $\varepsilon qi_q \bmod N$. The latter value can be rewritten as follows:

$$\varepsilon qi_q \equiv 2^t x_0 + k \bmod N \ , \tag{3.45}$$

where t and k are known values, and x_0 is the $\lceil \log_2(N) - t \rceil$-bit unknown integer that is to be recovered.

Lemma 6. *The unknown secret part x_0 is solution of the polynomial $P_\varepsilon(x)$:*

$$P_\varepsilon(x) = x^2 + c\ (2^{t+1}k - 2^t\varepsilon)\ x + c\ (k^2 - k\varepsilon) \equiv 0 \bmod N \ , \tag{3.46}$$

where $c = (2^{2t})^{-1} \bmod N$, k, t, N are known, and ε is the induced fault.

Proof. The Bézout identity applied to our context yields that primes p and q interrelate with integers $i_p = p^{-1} \bmod q$ and $i_q = q^{-1} \bmod p$ by the following relation:

$$pi_p + qi_q \equiv 1 \bmod N \ . \tag{3.47}$$

Multiplying (3.47) by ε leads to the relation $\varepsilon pi_p + \varepsilon qi_q \equiv \varepsilon \bmod N$, or equivalently to $\varepsilon pi_p \equiv \varepsilon - \varepsilon qi_q \bmod N$. Therefore, replacing εqi_q using (3.45) allows us to deduce an equivalence for εpi_p:

$$\varepsilon pi_p \equiv \varepsilon - 2^t x_0 - k \bmod N \ . \tag{3.48}$$

As $N = pq$, we then multiply (3.45) by (3.48), to get the relation:

$$\varepsilon qi_q \cdot \varepsilon pi_p \equiv (2^t x_0 + k) \cdot (\varepsilon - 2^t x_0 - k) \equiv 0 \bmod N \ . \tag{3.49}$$

Eventually, developing the right-hand side of (3.49), and multiplying by $c = (2^{2t})^{-1} \bmod N$ leads to the obtention of the monic polynomial $P_\varepsilon(x)$.

□

The initial problem of retrieving the unknown part of $\varepsilon qi_q \bmod N$ is thereby altered in solving the modular polynomial equation (3.46). In the sequel, we deal with two possible cases regarding ε, whether it is known to the adversary or not.

In the first case we assume that the fault ε is known to the adversary. This case corresponds to the *bit-flip* and *stuck-at* fault models (Sec. 2.1.2) since the message is known to the attacker and the fault location can be chosen. In both cases, since the

fault ε is known, the problem is reduced to solving a univariate modular polynomial equation, *cf.* Eq. 3.46. This problem is known to be hard. However, when the integer solution x is small, Coppersmith showed [55] that it can be retrieved using the well-known LLL algorithm. Accordingly, we induce the following proposition:

Proposition 6. *Given $N = pq$ and the low order $1/2\ \log_2(N)$ bits of $\varepsilon qi_q \bmod N$ and assuming ε is known, one can recover in time polynomial in $(\log_2(N), d)$ the factorization of N.*

Proof. From Coppersmith's Theorem [56], we know that, given a monic polynomial $P(x)$ of degree d, modulo an integer N of unknown factorization, and an upper bound X on the desired solution x_0, one can find in polynomial time all integers x_0 such that

$$P(x_0) \equiv 0 \bmod N \qquad \text{and} \qquad |X| < N^{1/d} \ . \tag{3.50}$$

In our case we have $d = 2$, and since x_0 is a $\lceil \log_2(N) - t \rceil$-bit integer, we know that $|x_0| < X = 2^{\lceil \log_2(N) - t \rceil}$. Thus, the condition in (3.50) becomes $2^{\lceil \log_2(N) - t \rceil} < N^{1/2}$, *i.e.*

$$t > \frac{1}{2} \log_2(N) \ . \tag{3.51}$$

Therefore, knowing at least half part of the secret $\varepsilon qi_q \bmod N$ allows to recover the whole secret. As previously done, computing $gcd(\varepsilon qi_q \bmod N, N)$ provides the factorization of N. □

Note that the method is deterministic, and as will be seen further (Table 3.3), it is reasonably fast.

Now let us assume for the second case that the fault ε is known to the adversary. This case is met in the *unknown constant* fault model (Sec. 2.1.2). In such a case, one can consider the polynomial $P_\varepsilon(x)$ as a bivariate modular polynomial equation with unknown values x and ε. This specific scheme has also been studied by Coppersmith and includes an additional difficulty of algebraic dependency of vectors which induces the heuristic characteristic of the method [55]. As depicted in Sec. 3.4.5, in our experiments nearly 100% of the tests verified the favorable property of independency. Accordingly, in this vast majority of cases, the following proposition holds:

Proposition 7. *Under an hypothesis of independency (see discussion above), given $N = pq$ and the low order $1/2\ \log_2(N) + s$ bits of $\varepsilon qi_q \bmod N$, where s denotes the bit-size of ε, and assuming ε is unknown, one can recover in time polynomial in $(\log_2(N), d)$ the factorization of N.*

Proof. Coppersmith's Theorem for the bivariate modular case [55] notifies that given a polynomial $P(x, \varepsilon)$ of total degree d, modulo an integer N of unknown factorization, and upper bounds X and E on the desired solutions x_0, ε_0, it may be possible (heuristic) to find in polynomial time all integer couples (x_0, ε_0) such that

$$P(x_0, \varepsilon_0) \equiv 0 \bmod N \qquad \text{and} \qquad |X \cdot E| < N^{1/d} \ . \tag{3.52}$$

In our case, we have $d = 2$ and $E = 2^s$. The integer x_0 is $\lceil \log_2(N) - t \rceil$-bit long, therefore we have $X = 2^{\lceil \log_2(N) - t \rceil}$. Thus, the condition in (3.52) becomes $2^{\lceil \log_2(N) - t \rceil} \cdot 2^s < N^{1/2}$, *i.e.*

$$t > \frac{1}{2} \log_2(N) + s \ . \tag{3.53}$$

This means that knowing s more bits of the secret $\varepsilon qi_q \bmod N$ than before, would allow the recovering of the whole secret. □

Table 3.3 – Size t required (in bytes) for the method to work and timings (Magma V2.17-1), as a function of the lattice dimension in Case 1 (ε known, being an 8-bit integer).

Size t required (bytes)	86	72	70	69	68	67	**66**	65	64
Dimension of the lattice	3	9	11	15	17	23	**37**	73	Theoretical
Time for LLL (seconds)	< 0.01	0.03	0.07	0.29	0.52	2.63	**34.25**	2587.7	bound

Table 3.4 – Size t required (in bytes) for the method to work and timings (Magma V2.17-1), as a function of the lattice dimension in Case 2 (ε unknown, being a 32-bit integer).

Size t required (bytes)	86	78	76	74	73	**72**	71	70	69
Dimension of the lattice	5	12	22	35	51	**70**	117	201	Theoretical
Time for LLL (seconds)	< 0.01	0.02	0.16	1.17	5.88	**30.22**	605.9	12071.1	bound

Remark 5. *The bound of success in Proposition 7 can actually be slightly improved using results of [30]. Indeed, Coppersmith's bound applies to polynomials whose monomials shape is rectangular, while in our case the monomial ε^2 does not appear in $P(x, \varepsilon)$ which corresponds to what they called an extended rectangle in [30]. For the sake of simplicity, we only mentioned Coppersmith's bound since practical results are similar.*

Results From Our Implementation

We have implemented this lattice-based improvement using Magma Software [37], with N a 1024-bit integer *i.e.* 128 bytes long, in the cases where ε is an 8-bit known value (for Case 1) and a 32-bit unknown value (for Case 2). We chose Howgrave-Graham's method [100] for the univariate case, and its generalization by Jochemsz *et al.* [104] for the bivariate case since both have the same bound of success as Coppersmith's method (sometimes even better for [104]) and they are easier to implement. As we know, the theoretical bound given in Coppersmith's method is only asymptotic [56]. Thus, we report in Table 3.3 (for Case 1) and in Table 3.4 (for Case 2) the size t (in bytes) of the secret $\varepsilon q i_q \bmod N$ that is known to the attacker before applying Coppersmith's method, the lattice dimension used to solve (3.46) and finally the timings of our attack.

As depicted in Table 3.3, and combining these results with the experiments of Sec. 3.4.3, the best trade-off is to perform a CPA on the 66 first bytes, taking $66 \times 1m05s = 1h11m30s$, and to retrieve the 62 remaining bytes using lattices in $34.25s$, bringing the total time up to 1 hour 12 minutes, instead of the previous 2 hours 20 minutes.
In order to illustrate Case 2, we have chosen to rather show our results for ε being a 32-bit value, since when ε is 8-bit long, we obtained slightly better results by considering the 255 possible values of the variable ε together with their corresponding polynomials $P_\varepsilon(x)$, and by running the method on each of the polynomials until finding the solution x_0 that allows to factorize N. This indeed leads to a best trade-off of 70 bytes required

from the CPA and the 58 remaining bytes computed with lattices by performing 255 times the LLL algorithm in the worst case, for a total of $68 \times 1m05s + 255 \times 0.52s$, *i.e.* 1 hour 16 minutes instead of 2 hours 20 minutes. Besides, this exhaustive search can be performed in parallel and it also has the advantage to be deterministic.
However, when ε is 32-bit long, an exhaustive search becomes impractical and, as depicted in Table 3.4, the best trade-off would be to perform a CPA on 72 bytes and to compute the 56 remaining bytes with lattices (even if heuristic, it worked in nearly 100% of the tests in practice), resulting in a total of $72 \times 1m05s + 30.22s$, *i.e.* 1 hour 18 minutes instead of the previous 2 hours 20 minutes.

3.4.6 Countermeasures

In this section, we describe different countermeasures to protect an implementation against the CA presented in Sec. 3.4.2.

Blind Before Splitting

Our first proposition consists in avoiding the possibility to inject the same fault during several signature computations. To do so, we deport the blinding of the input message m before executing the two exponentiations modulo p and q:

$$m' = m + k_0 N \bmod k_1 N \ , \tag{3.54}$$

with k_0 and k_1 two n-bit random values generated at each algorithm execution (n being typically 64). Hence $S'_p = m'^{d_p} \bmod k_2 p$ and $S'_q = m'^{d_q} \bmod k_3 q$.

This countermeasure prevents an attacker from injecting always the same error during the signature computation. Indeed if the fault is injected on m at the very beginning of one exponentiation, then the corresponding error cannot be fixed due to the blinding injected by Eq. 3.54.

Moreover, if the fault is injected when the message m is manipulated in Eq. 3.54, then the error ε impacts the computation of both S'_p and S'_q, leading to non-exploitable faulty outputs.

Such a countermeasure induces a small overhead in terms of memory space since m' must be kept in memory during the first exponentiation but the execution time remains the same.

Verification Blinding

Our second countermeasure aims at annihilating the second hypothesis of our attack: a predictive variable is manipulated in plain during the verification. To do so, we inject a $\lceil \log_2(N) \rceil$-bit random r before performing the final reduction with N, *cf.* Eq. 3.55. Therefore, each and every variable manipulated during the verification is blinded.

$$((\widetilde{S}^e + r - m) \bmod k_1 N) \bmod N \stackrel{?}{=} r \ . \tag{3.55}$$

One may note that the final comparison should be performed securely with regards to the attack described in [117] since information on $\varepsilon q i_q$ could leak if such a comparison was performed through a substraction.

The cost of such a countermeasure is negligible since it mainly consists in generating a $\lceil \log_2(N) \rceil$-bit random variable.

3.4.7 Conclusion

With this work we have introduced a new combined attack on CRT-RSA. Even if a secure implementation does not return the faulty signature when the computation is disturbed, we show how to combine FA with SCA during the verification process to obtain information on the faulty signature. Such information allows us to factorize the public modulus and thus to recover the whole private key. We also show that Coppersmith's methods to solve univariate and bivariate modular polynomial equations can be used to significantly reduce the complexity of our new attack. Finally, we provide simulations to confirm the efficiency of our method and we present two countermeasures which have a very small penalty on the performance of the algorithm. Our main objective was to prove that stacking several countermeasures does not provide global security despite addressing each and every attack separately. Therefore, the main consequence of this work is that fault injection countermeasures must also be designed to resist SCA and vice versa.

3.5 Conclusion

In this chapter we have investigated the problem of passive physical attacks, also denoted as side-channel attacks. While fault attacks Chap. 2 are very invasive and can be detected, side-channel attacks are less intrusive, can be concealed and can provide similar results as faults. They have been one of the main concerns of the recent advances in cryptographic implementations both on attacks and countermeasures. In Sec. 3.1 we have thus briefly recalled the framework used in research to analyze side-channel attacks. Namely we have recalled the various kind of side-channel vectors, together with the various theoretical models that emerged in the literature to study the subject. We have afterwards recalled the main strategies to break implementations for both symmetric and asymmetric algorithms, together with some of the countermeasures that have been found to thwart them. One of the countermeasures for symmetric ciphers that have been widely studied is the Rivain-Prouff [146] secure multiplication countermeasure. In Sec. 3.2 we introduce our recent work which has been submitted to CHES 2016. By exploiting the repeated manipulation of the variables used in the secure multiplication we show that it is possible to break with an horizontal attack in the Gaussian leakage model [14]. We afterwards present in Sec. 3.2 the results of our work of security analysis of a recent publication which exploits the properties of linear codes to secure symmetric implementations from both side-channels and fault attacks. We also provide some suggestion to patch the problem that we have found [9]. Finally, in Sec. 3.4 we present an attack that have been published at PKC 2013 against CRT-RSA secure implementations. We have used an attack combining a fault and a side-channel to mount a Bellcore attack against an CRT-RSA implementation protected against faults and side-channels. We show that this countermeasure is not sufficient against an attacker that employs both of them at the same time. In our work we also suggest some possible efficient countermeasures against our new attack [10].

Chapter 4

Conclusions and Open Problems

This thesis focuses on the security of cryptographic schemes in real world implementations. It has been shown that far from the classical model of black box security, modern cryptographic implementations need to withstand attacks where the enemy knows the system, but also she is given partial access to its sensitive internal state. Two main expressions exists of such a model, namely fault attacks and side-channel attacks. While the former deals with attackers that can tamper with the algorithm internals, the latter concerns physical observables that depends on the sensitive internal states. Our work considers both these aspects to evaluate the security of embedded cryptosystems implementations.

4.1 Fault Attacks

In the second chapter we have analyzed the security of various countermeasures to thwart fault attacks. Similarly to what happens for block ciphers, it is still not clear how to provide a sound model for analyzing cryptographic primitives against faults; thus it is often let to the community the role of validating the security of new countermeasures by continuous challenge. Our work mainly focused on the security of infective countermeasures. We have suggested attacks against several infective countermeasures, both asymmetric and symmetric ones. We have demonstrated that almost none of the infective countermeasures suggested in the literature provide enough confidence to be viable. We thus think that a sound model for theoretical analysis of fault attacks is becoming more and more urging in order to provide sound proofs of security for countermeasures and improved design techniques for new ciphers.

We have also analyzed the soundness of fault countermeasures for elliptic curves when special input points are chosen by the attacker. One of the main results of this work is a concrete reason to avoid cryptosystems where users can choose the input point and retrieve the result of the scalar multiplication on an elliptic curve.

4.2 Side-Channel Attacks

The third chapter of this thesis deals with side-channel attacks on cryptographic implementations. We have worked on three different aspects of side-channel security. First of all we have analyzed the security of a recently suggested countermeasure for the AES cryptosystem, based on the use of dual codes. We observe that serious biases in the countermeasure are detectable with far less effort than what was suggested in the orig-

inal work. Thus we have suggested some cheap variants of the countermeasure which may avoid our attack.

Afterwards, theoretical investigations led us to show practical examples of symmetric masking schemes in which increasing the number of shares used to thwart high-order side-channel attacks may indeed reduce the overall security when the noise distribution affecting each observation is constant, which is what we typically observe in practice. Despite mostly theoretical, this works tackle a very important issue of practical implementations that seems to be often underestimated by designers.

Finally we investigated the security of state-of-the-art RSA implementations, we show that the common countermeasure consisting in verifying the signature with the public exponent may indeed open the way to attacks that combine fault and side-channel analysis. Our work allows to put into perspective the common development of countermeasures, which are often directed against faults or side channel attacks, and fails to thwart combinations of the two.

4.3 Open Problems

During our research we have found some interesting problems which may require further investigations.

Concerning infective countermeasures we remark that despite a number of cryptanalytic attacks, this paradigm still seems a promising subject which may solve problems related to classical countermeasures. However, all suggested infective algorithms seem to be adaptations of classical countermeasures. Thus it would be of interest to investigate if there exists infective algorithms that are intrinsically different from classical ones, and to what extent they do really provide better countermeasures.

As already mentioned earlier in this manuscript, another line of research that should be investigated is the development of a formal model for the study of the security of cryptographic implementations in the presence of faults. While this problem is not new to the community, only limited advances have been made, in particular concerning automatic formal analysis tools. Where side-channel analysis can count on countermeasures proven under some particular model, fault countermeasures still often relies on the intuition of cryptographers to withstand attacks.

Finally, as we have observed, common points on elliptic curves provide a useful resource to attackers. However there seems to be no investigation on the use of such particular points to build efficient countermeasures, which may be possible due to their interesting properties. Furthermore we suggest that such particular points may be useful to provide efficient implementation tricks for elliptic curve algorithms.

Personal Bibliography

[1] Guillaume Barbu and Alberto Battistello. Analysis of a Code-based Countermeasure against Side-Channel and Fault Attacks, *Unpublished*.

[2] Guillaume Barbu, Alberto Battistello, Guillaume Dabosville, Christophe Giraud, Guénaël Renault, Soline Renner, and Rina Zeitoun. Combined Attack on CRT-RSA - Why Public Verification Must Not Be Public? In *Public-Key Cryptography–PKC 2013*, pages 198–215. Springer, 2013.

[3] Alberto Battistello. Common Points on Elliptic Curves: The Achilles' Heel of Fault Attack Countermeasures. In *Constructive Side-Channel Analysis and Secure Design*, pages 69–81. Springer, 2014.

[4] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme. In *Cryptographic Hardware and Embedded Systems-CHES 2016*. Springer, 2016.

[5] Alberto Battistello and Christophe Giraud. Fault analysis of infective AES computations. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, pages 101–107. IEEE, 2013.

[6] Alberto Battistello and Christophe Giraud. Lost in Translation: Fault Analysis of Infective Security Proofs. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2015 Workshop on*. IEEE, 2015.

[7] Alberto Battistello and Christophe Giraud. A Note on the Security of CHES 2014 Symmetric Infective Countermeasure. In Elisabeth Oswald and François-Xavier Standaert, editors, *Third International Workshop on Constructive Side-Channel Analysis and Secure Design – COSADE 2016*, LNCS, pages 101–107. IEEE, Springer, 2016.

Bibliography

[1] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 601–610. ACM, 2001. 21

[2] F. Amiel, B. Feix, L. Marcel, and K. Villegas. Passive and Active Combined Attacks – Combining Fault Attacks and Side Channel Analysis –. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 92–99. IEEE Computer Society, 2007. 116

[3] A. Antipa, D. Brown, A. Menezes, R. Struik, and S. Vanstone. Validation of elliptic curve public keys. In *Public Key Cryptography—PKC 2003*, pages 211–223. Springer, 2003. 44, 45

[4] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2003. 42, 43, 47, 48

[5] S. Azzi, M. Christofi, D. Vigilant, and B. Barras. Using Linear Codes as a Fault Countermeasure for Nonlinear Operations: Application to AES and Formal Verification. 2015. 106

[6] J. Balasch, S. Faust, and B. Gierlichs. Inner Product Masking Revisited. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 486–510. Springer, 2015. 78, 86

[7] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. Cryptology ePrint Archive, Report 2004/100, 2004. http://eprint.iacr.org/2004/100. 40

[8] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006. 39

[9] G. Barbu and A. Battistello. Analysis of a Code-based Countermeasure against Side-Channel and Fault Attacks, *Unpublished*. I, II, 75, 125

[10] G. Barbu, A. Battistello, G. Dabosville, C. Giraud, G. Renault, S. Renner, and R. Zeitoun. Combined Attack on CRT-RSA - Why Public Verification Must Not

Be Public? In *Public-Key Cryptography–PKC 2013*, pages 198–215. Springer, 2013. II, 125

[11] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, and B. Grégoire. Compositional Verification of Higher-Order Masking: Application to a Verifying Masking Compiler. Cryptology ePrint Archive, Report 2015/506, 2015. http://eprint.iacr.org/. 87, 150

[12] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, 2011. 83

[13] A. Battistello. Common Points on Elliptic Curves: The Achilles' Heel of Fault Attack Countermeasures. In *Constructive Side-Channel Analysis and Secure Design*, pages 69–81. Springer, 2014. I, II, 38

[14] A. Battistello, J.-S. Coron, E. Prouff, and R. Zeitoun. Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme. In *Cryptographic Hardware and Embedded Systems-CHES 2016*. Springer, 2016. I, II, 75, 125

[15] A. Battistello and C. Giraud. Fault analysis of infective AES computations. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, pages 101–107. IEEE, 2013. I, II, 38, 58

[16] A. Battistello and C. Giraud. Lost in Translation: Fault Analysis of Infective Security Proofs. In N. Homma and V. Lomné, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2015 Workshop on*. IEEE, IEEE, 2015. I, II, 38

[17] A. Battistello and C. Giraud. A Note on the Security of CHES 2014 Symmetric Infective Countermeasure. In E. Oswald and F.-X. Standaert, editors, *Third International Workshop on Constructive Side-Channel Analysis and Secure Design – COSADE 2016*, LNCS, pages 101–107. IEEE, Springer, 2016. I, II, 38

[18] A. Bauer, É. Jaulmes, E. Prouff, and J. Wild. Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations. In E. Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013. 89

[19] S. Belaïd, J.-S. Coron, P.-A. Fouque, B. Gérard, J.-G. Kammerer, and E. Prouff. Improved side-channel analysis of finite-field multiplication. In *Cryptographic Hardware and Embedded Systems–CHES 2015*, pages 395–415. Springer, 2015. 76, 80, 115

[20] S. Belaïd, P.-A. Fouque, and B. Gérard. Side-channel analysis of multiplications in GF (2128). In *Advances in Cryptology–ASIACRYPT 2014*, pages 306–325. Springer, 2014. 76, 80, 115

[21] Bellcore. New Threat Model Breaks Crypto Codes. Press Release, Sept. 1996. 42, 44

[22] 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. 44, 48

[23] I. Biehl, B. Meyer, and V. Müller. Differential fault attacks on elliptic curve cryptosystems. In *Advances in Cryptology—CRYPTO 2000*, pages 131–146. Springer, 2000. 44, 45, 68

[24] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991. 28, 46

[25] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology—CRYPTO'97*, pages 513–525. Springer, 1997. 45

[26] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir. Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. volume 6110, pages 299–319. Springer, 2010. 28

[27] A. Biryukov and D. Khovratovich. Related-key Cryptanalysis of the Full AES-192 and AES-256. 2009. http://eprint.iacr.org/2009/317. 28

[28] A. Biryukov, D. Khovratovich, and I. Nikolič. Distinguisher and Related-key attack on the Full AES-256. In S. Halevi, editor, *Advances in Cryptology – CRYPTO '09*, volume 5677 of *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009. 28

[29] G. Blakely. Safeguarding cryptographic keys. In *National Comp. Conf.*, volume 48, pages 313–317, New York, June 1979. AFIPS Press. 85

[30] J. Blomer and A. May. A Tool Kit for Finding Small Roots of Bivariate Polynomials over the Integers. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 251–267. Springer, 2005. 123

[31] 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 2003*, pages 311–320. ACM Press, 2003. 44

[32] J. Blömer, M. Otto, and J.-P. Seifert. Sign change fault attacks on elliptic curve cryptosystems. In *Fault Diagnosis and Tolerance in Cryptography*, pages 36–52. Springer, 2006. 44, 45, 67, 68, 71

[33] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full AES. volume 7073, pages 344–371. Springer, 2011. 28

[34] 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 *LNCS*, pages 37–51. Springer, 1997. 42, 45, 117

[35] D. Boneh et al. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999. 29, 31

[36] J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. *International Journal of Applied Cryptography*, 2(3):212–228, 2012. 69

[37] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993). 123

[38] S. Briais, J.-M. Cioranesco, J.-L. Danger, S. Guilley, D. Naccache, and T. Porteboeuf. Random active shield. In G. Bertoni and B. Gierlichs, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2012*, pages 103–113. IEEE, IEEE Computer Society, 2012. 85

[39] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 16–29. Springer, 2004. 82

[40] J. Bringer, C. Carlet, H. Chabanne, S. Guilley, and H. Maghrebi. Orthogonal Direct Sum Masking: A Smartcard Friendly Computation Paradigm in a Code, with Builtin Protection against Side-Channel and Fault Attacks. In D. Naccache and D. Sauveron, editors, *Information Security Theory and Practice International Workshop – WISTP 2014*, volume 8501 of *LNCS*, pages 40–56. Springer, 2014. 106, 108, 109, 111, 112, 145

[41] J. Bringer, H. Chabanne, and T. H. Le. Protecting AES against side-channel analysis using wire-tap codes. *Journal of Cryptographic Engineering*, 2(2):129–141, 2012. 106

[42] C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-Order Masking Schemes for S-Boxes. In A. Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012. 78, 86

[43] M. Carreira-Perpinan. Mode-finding for mixtures of Gaussian distributions Carreira-Perpinan. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 22(11):1318–1323, November 2000. 88

[44] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. 78, 85, 89, 91, 92

[45] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In B. S. K. Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002. 83

[46] D. Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In J. Simon, editor, *Proceedings of the 20th*

Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pages 11–19. ACM, 1988. 85

[47] Chevallier-Mames, Benoît and Ciet, Mathieu and Joye, Marc. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *Computers, IEEE Transactions on*, 53(6):760–768, 2004. 81

[48] H. Choukri and M. Tunstall. Round Reduction Using Faults. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2005*, LNCS. Springer, 2005. 60, 64

[49] M. Ciet and M. Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, codes and cryptography*, 36(1):33–43, 2005. 44, 45, 68, 71

[50] 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 2005*, LNCS, pages 124–132. Springer, 2005. 44, 48

[51] J.-M. Cioranesco, J.-L. Danger, T. Graba, S. Guilley, Y. Mathieu, D. Naccache, and X. T. Ngo. Cryptographically secure shields. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 25–31. IEEE, 2014. 85

[52] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Horizontal Correlation Analysis on Exponentiation. In M. Soriano, S. Qing, and J. López, editors, *Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*, volume 6476 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2010. 86, 89

[53] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Square Always Exponentiation. In D. J. Bernstein and S. Chatterjee, editors, *INDOCRYPT 2011 - 12th International Conference on Cryptology in India*, volume 7107 of *LNCS*, pages 40–57. Springer, 2011. 116

[54] D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996. 21

[55] D. Coppersmith. Finding a small root of a univariate modular equation. volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1996. 21, 122

[56] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997. 121, 122, 123

[57] J. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems*, pages 292–302. Springer, 1999. 71, 81

[58] J. Coron. Higher Order Masking of Look-Up Tables. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual*

International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014. 78, 83, 85, 86

[59] J. Coron, A. Roy, and S. Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. *J. Cryptographic Engineering*, 5(2):73–83, 2015. 78, 86

[60] J.-S. Coron, C. Giraud, N. Morin, G. Piret, and D. Vigilant. Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm. In L. Breveglieri, M. Joye, I. Koren, D. Naccache, and I. Verbauwhede, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2010*, pages 89–96. IEEE Computer Society, 2010. 42, 47, 50

[61] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. Higher-order side channel security and mask refreshing. In *Fast Software Encryption*, pages 410–424. Springer, 2013. 85

[62] J.-S. Coron and A. Tchulkine. A new algorithm for switching from arithmetic to boolean masking. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 89–97. Springer, 2003. 85

[63] G. Dabosville, J. Doget, and E. Prouff. A new second-order side channel attack based on linear regression. *Computers, IEEE Transactions on*, 62(8):1629–1640, 2013. 83

[64] J. Daemen and V. Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013. 29

[65] B. Debraize. Efficient and provably secure methods for switching from arithmetic to boolean masking. In *Cryptographic Hardware and Embedded Systems–CHES 2012*, pages 107–121. Springer, 2012. 85

[66] W. Diffie and M. E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976. 24, 29

[67] J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert. Univariate side channel attacks and leakage modeling. *Journal of Cryptographic Engineering*, 1(2):123–144, 2011. 83

[68] 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 *LNCS*, pages 68–83. Springer, 2009. 117

[69] A. Duc, S. Dziembowski, and S. Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 423–440, 2014. 79, 86, 100

[70] A. Duc, S. Faust, and F. Standaert. Making Masking Security Proofs Concrete - Or How to Evaluate the Security of Any Leaking Device. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 401–429, 2015. 85, 86, 105

[71] V. Dupaquis and A. Venelli. Redundant Modular Reduction Algorithms. In E. Prouff, editor, *Smart Card Research and Advanced Applications, 10th International Conference – CARDIS 2011*, LNCS, pages 102–114. Springer, 2011. 116

[72] J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, A. Tria, and T. Vaschalde. Fault Round Modification Analysis of the Advanced Encryption Standard. In *IEEE International Symposium on Hardware-Oriented Security and Trust – HOST 2012*, pages 28–39. IEEE, 2012. 60

[73] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1985. 31

[74] J. Fan, B. Gierlichs, and F. Vercauteren. To infinity and beyond: combined attack on ECC using points of low order. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 143–159. Springer, 2011. 67, 68, 71, 116

[75] J.-C. Faugere, C. Goyet, and G. Renault. Attacking (EC) DSA given only an implicit hint. In *Selected Areas in Cryptography*, pages 252–274. Springer, 2012. 67

[76] S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 135–156, 2010. 87, 98

[77] B. Feix and A. Venelli. Defeating with Fault Injection a Combined Attack Resistant Exponentiation. In W. Schindler and S. Huss, editors, *Third International Workshop on Constructive Side-Channel Analysis and Secure Design – COSADE 2013*, LNCS. Springer, 2013. 44

[78] J. Ferrigno and M. Hlavác. When AES blinks: introducing optical side channel. *Information Security, IET*, 2(3):94–98, 2008. 77

[79] P. FIPS. 186-2. digital signature standard (dss). *National Institute of Standards and Technology (NIST)*, 2000. 31

[80] FIPS PUB 186-4. *Digital Signature Standard*. National Institute of Standards and Technology, July 2013. 67, 69

[81] FIPS PUB 197. *Advanced Encryption Standard*. National Institute of Standards and Technology, Nov. 2001. 26, 59

[82] P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. Fault attack on elliptic curve Montgomery ladder implementation. In *Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on*, pages 92–98. IEEE, 2008. 68

[83] J. Friedman. Tempest: A signal problem. *NSA Cryptologic Spectrum*, 1972. 35, 76

[84] C. Gauss and A. Clarke. *Disquisitiones Arithmeticae*. Springer, 1986. 11

[85] L. Genelle, E. Prouff, and M. Quisquater. Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011. 78, 86

[86] D. Genkin, A. Shamir, and E. Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *Advances in Cryptology–CRYPTO 2014*, pages 444–461. Springer, 2014. 76

[87] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems–CHES 2008*, pages 426–442. Springer, 2008. 82, 83

[88] B. Gierlichs, J.-M. Schmidt, and M. Tunstall. Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output. In A. Hevia and G. Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 305–321. Springer, 2012. 46, 55, 57, 58, 59, 73

[89] B. Gierlichs, J.-M. Schmidt, and M. Tunstall. Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output. Cryptology ePrint Archive, Report 2012/678, 2012. http://eprint.iacr.org/. 55, 58

[90] C. Giraud. Dfa on aes. In *Advanced Encryption Standard–AES*, pages 27–41. Springer, 2004. 45

[91] C. Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *Computers, IEEE Transactions on*, 55(9):1116–1120, 2006. 42, 81

[92] C. Glowacz, V. Grosso, R. Poussier, J. Schüth, and F. Standaert. Simpler and More Efficient Rank Estimation for Side-Channel Security Assessment. In G. Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2015. 93, 97

[93] L. Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In *Public key cryptography—PKC 2003*, pages 199–211. Springer, 2003. 67, 80

[94] 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 *LNCS*, pages 158–172. SV, 1999. 85

[95] V. Grosso, E. Prouff, and F. Standaert. Efficient Masked S-Boxes Processing - A Step Forward -. In D. Pointcheval and D. Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, volume 8469 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2014. 78, 83, 86

[96] S. Guilley, A. Heuser, and O. Rioul. A Key to Success - Success Exponents for Side-Channel Distinguishers. In A. Biryukov and V. Goyal, editors, *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, volume 9462 of *Lecture Notes in Computer Science*, pages 270–290. Springer, 2015. 91, 92

[97] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009. 78

[98] S. Halevi and H. Lin. After-the-fact leakage in public-key encryption. In *Theory of Cryptography*, pages 107–124. Springer, 2011. 78

[99] G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers*. Oxford University Press, 1979. 15

[100] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *6th IMA International Conference*, volume 1355, pages 131 – 142. Springer, 1997. 123

[101] N. Howgrave-Graham and N. Smart. Lattice attacks on Digital Signature Schemes. In *Designs, Codes and Cryptography*, volume 23, pages 283–290, 2001. 32, 34

[102] L. H. Ing. The History of The Chinese Remainder Theorem. *Mathematical Medley*, 30, 2003. 13

[103] Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 463–481, 2003. 78, 81, 85, 86

[104] E. Jochemsz and A. May. A Strategy for Finding Roots of Multivariate Polynomials with New Applications in Attacking RSA Variants. In *ASIACRYPT'06*, pages 267–282, 2006. 123

[105] M. Joye and M. Tunstall. *Fault Analysis in Cryptography*. Springer, 2012. 40

[106] M. Joye and S.-M. Yen. The Montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 291–302. Springer, 2002. 81

[107] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, et al. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology–CRYPTO 2010*, pages 333–350. Springer, 2010. 31

[108] D. Knuth. *The Art of Computer Programming 2: Seminumerical Algorithms*. MA: Addison-Wesley, 1968. 19

[109] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. 31

[110] 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 *LNCS*, pages 104–113. Springer, 1996. 35, 76

[111] P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc., 1998. 35, 76, 77, 79, 80, 81

[112] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. volume 6110, pages 388–397, 2010. 35, 76, 77, 80

[113] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011. 80

[114] A. Lenstra. Memo on RSA Signature Generation in the Presence of Faults. Manuscript, 1996. 42

[115] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. 21

[116] C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology—CRYPTO'97*, pages 249–263. Springer, 1997. 44, 67

[117] V. Lomné, T. Roche, and A. Thillard. On the Need of Randomness in Fault Attack Countermeasures – Application to AES. In G. Bertoni and B. Gierlichs, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2012*, pages 85–94. IEEE Computer Society, 2012. 46, 52, 53, 54, 73, 124

[118] S. F. Marcin Andrychowicz, Stefan Dziembowski. Circuit compilers with O(1/log n) leakage rate. In M. Fischlin and J.-S. Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615. Springer, 2016. 87, 101

[119] D. P. Martin, J. F. O'Connell, E. Oswald, and M. Stam. Counting Keys in Parallel After a Side Channel Attack. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 313–337. Springer, 2015. 93, 97

[120] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *LNCS*, pages 386–397. Springer, 1993. 28, 46

[121] A. May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In *Advances in Cryptology–CRYPTO 2004*, pages 213–219. Springer, 2004. 31

[122] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996. 15, 22

[123] T. Messerges. Using second-order power analysis to attack DPA resistant software. In *Cryptographic Hardware and Embedded Systems—CHES 2000*, pages 27–78. Springer, 2000. 83, 84

[124] S. Micali and L. Reyzin. Physically Observable Cryptography (Extended Abstract). In M. Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004. 77, 81, 88

[125] D. Micciancio. Lattice-based cryptography. In *Encyclopedia of Cryptography and Security*, pages 713–715. Springer, 2011. 19

[126] D. Micciancio. The geometry of lattice cryptography. In *Foundations of security analysis and design VI*, pages 185–210. Springer, 2011. 19

[127] V. Miller. Use of Elliptic Curves in Cryptography. In H. Wiliams, editor, *Advances in Cryptology – CRYPTO '85*, volume 218 of *LNCS*, pages 417–426. Springer, 1985. 31

[128] A. Moradi. Wire-Tap Codes as Side-Channel Countermeasure–an FPGA-based experiment–. 106

[129] A. Moradi. Wire-Tap Codes as Side-Channel Countermeasure. In *Progress in Cryptology–INDOCRYPT 2014*, pages 341–359. Springer, 2014. 83

[130] D. Mukhopadhyay. An Improved Fault Based Attack of the Advanced Encryption Standard. In B. Preneel, editor, *AFRICACRYPT 2009*, volume 5580 of *LNCS*, pages 421–434. Springer, 2009. 46, 53, 54

[131] O. Neiße and J. Pulkus. Switching blindings with a view towards IDEA. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 230–239. Springer, 2004. 85

[132] P. Nguyen and I. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. In *Designs, Codes and Cryptography*, 2003. 32, 34

[133] P. Q. Nguyen. *La géométrie des nombres en cryptologie*. PhD thesis, Université de Paris 07, 1999. 21

[134] P. Q. Nguyen and J. Stern. The Two Faces of Lattices in Cryptology. In *Revised Papers from the International Conference on Cryptography and Lattices*, CaLC '01, pages 146–180, London, UK, UK, 2001. Springer-Verlag. 19, 21

[135] Y. Oren, M. Renauld, F. Standaert, and A. Wool. Algebraic Side-Channel Attacks Beyond the Hamming Weight Leakage Model. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2012. 86

[136] T. PARI-Group. Pari/gp , version `2.5.3`, Bordeaux. 2013. 69

[137] 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 *LNCS*, pages 77–88. Springer, 2003. 46

[138] J. M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975. 31

[139] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978. 67

[140] E. Prouff and M. Rivain. Masking Against Side-Channel Attacks: a Formal Security Proof. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013 - 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *LNCS*, pages 142–159. SV, 2013. 79, 85, 86

[141] E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *Computers, IEEE Transactions on*, 58(6):799–811, 2009. 84

[142] E. Prouff and T. Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems, 13th International Workshop – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011. 78, 83, 86

[143] P. Rauzy and S. Guilley. Countermeasures against High-Order Fault-Injection Attacks on CRT-RSA. In A. Tria and D. Choi, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2014*, pages 68–82. IEEE, 2014. 47, 48, 49, 50, 51, 52

[144] P. Rauzy, M. Moreau, S. Guilley, and Z. Najm. Using modular extension to provably protect ecc against fault attacks. Cryptology ePrint Archive, Report 2015/882, 2015. http://eprint.iacr.org/. 72, 73

[145] M. Rivain. Securing rsa against fault analysis by double addition chain exponentiation. In *Topics in Cryptology–CT-RSA 2009*, pages 459–480. Springer, 2009. 42

[146] M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 413–427, 2010. 78, 85, 86, 87, 106, 125

[147] 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. 29

[148] 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 *LNCS*, pages 413–426. Springer, 2007. 116

[149] T. Roche, V. Lomné, and K. Khalfallah. Combined Fault and Side-Channel Attack on Protected Implementations of AES. In E. Prouff, editor, *Smart Card Research and Advanced Applications, 10th International Conference – CARDIS 2011*, LNCS, pages 152–169. Springer, 2011. 116

[150] S. Ross. *A First Course in Probability*. Pearson Prentice Hall, 2010. 16

[151] W. Schindler, K. Lemke, and C. Paar. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 30–46. Springer, 2005. 83

[152] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, and J.-P. Seifert. Simple photonic emission analysis of AES. volume 7428 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2012. 77

[153] J.-M. Schmidt, M. Tunstall, R. Avanzi, I. Kizvhatov, T. Kasper, and D. Oswald. Combined implementation attack resistant exponentiation. In M. Abdalla and P. Barreto, editors, *LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 305–322. Springer, 2010. 44

[154] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2):201–224, 1987. 21

[155] R. Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995. 69

[156] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979. 85

[157] A. Shamir. How to check modular exponentiation. Eurocrypt'97 rump session, 1997. 42, 48

[158] D. Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math*, volume 20, pages 415–440, 1971. 31

[159] C. E. Shannon. A Mathematical Theory of Communication. *Bell system technical journal*, 27:379–423, 623–656, 1948. 22, 26

[160] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell system technical journal*, 28:656–715, 1949. 22, 26

[161] F. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010. 89, 92

[162] S. H. Standard. FIPS 180-1. *Secure Hash Standard, NIST, US Dept. of Commerce, Washington DC April*, 9:21, 1995. 32

[163] Standards for Efficient Cryptography Group (SECG). *SEC 2 Ver 2.0 : Recommended Elliptic Curve Domain Parameters*. Certicom Research, Jan. 2010. 67

[164] D. R. Stinson. *Cryptography: theory and practice*. CRC press, 2005. 22

[165] H. Tupsamudre, S. Bisht, and D. Mukhopadhyay. Destroying Fault Invariant with Randomization – A countermeasure for AES Against Differential Fault Attacks. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *LNCS*, pages 93–111. Springer, 2014. 46, 58, 59, 60, 63, 73

[166] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58:345–363, 1938. 18

[167] A. M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 2(1):161–228, 1939. 18

[168] W. Van Eck. Electromagnetic radiation from video display units: an eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985. 35

[169] N. Veyrat-Charvillon, B. Gérard, and F. Standaert. Soft Analytical Side-Channel Attacks. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014. 86

[170] N. Veyrat-Charvillon and F.-X. Standaert. Mutual information analysis: how, when and why? In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 429–443. Springer, 2009. 83

[171] 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 *LNCS*, pages 130–145. Springer, 2008. 42

[172] 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. 44

[173] M. J. Wiener. Cryptanalysis of short RSA secret exponents. *Information Theory, IEEE Transactions on*, 36(3):553–558, 1990. 31

[174] A. Wiles. Modular elliptic curves and Fermat's last theorem. *Annals of Mathematics*, pages 443–551, 1995. 15

[175] M. Witteman, J. van Woudenberg, and F. Menarini. Defeating RSA Multiply-Always and Message Blinding Countermeasures. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *LNCS*, pages 77–88. Springer, 2011. 116

[176] P. Wright and P. Greengrass. *Spycatcher: The candid autobiography of a senior intelligence officer*. Dell Publishing Company, 1987. 35

[177] P. Wright and P. Greengrass. *Spycatcher: The candid autobiography of a senior intelligence officer*. Dell Publishing Company, 1987. 76

[178] S.-M. Yen and D. Kim. Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2004*, pages 381–385. IEEE Computer Society, 2004. 44

[179] 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 *LNCS*, pages 53–61. Springer, 2006. 44

[180] S.-M. Yen, S.-J. Kim, S.-G. Lim, and S.-J. Moon. RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *LNCS*, pages 397–413. Springer, 2001. 44

[181] S.-M. Yen, S. Moon, and J.-C. Ha. Hardware fault attack on rsa with crt revisited. In *Information Security and Cryptology—ICISC 2002*, pages 374–388. Springer, 2002. 43

[182] Y. Zhou and D. Feng. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. *IACR Cryptology ePrint Archive*, 2005:388, 2005. 35

[183] J. F. Ziegler and W. Lanford. Effect of cosmic rays on computer memories. *Science*, 206(4420):776–788, 1979. 39

Personal Patents Bibliography

[1] Guillaume Barbu, Alberto Battistello, Christophe Giraud, and Soline Renner. Method for testing the security of an electronic device against an attack, and electronic device implementing countermeasures, 09 2012. I, II

[2] Alberto Battistello, Olivier Chamley, and Christophe Giraud. Procede et systeme pour la verification de la validitè d'une signature numerique de message, 11 2013. I, II

[3] Alberto Battistello, Guillaume Dabosville, Christophe Giraud, and Laurie Genelle. Géneration de message pour test de génération de clés cryptographiques, 09 2015. I, II

[4] Alberto Battistello and Christophe Giraud. Generation of cryptographic keys, 10 2016. I, II

[5] Alberto Battistello, Christophe Giraud, Guillaume Dabosville, and Laurie Genelle. Integrity Verification of Key Pairs, 06 2015. I, II

[6] Alberto Battistello, Christophe Giraud, Julien Matha, and Nicolas Morin. Contremesure à la préparation des cartes pour détecter des attaques lumière, 08 2012. I, II

Appendix A

Annexes

A.1 The [16,8,5]-code and Related Matrices

As exposed in Appendix B of [40], the selected code was picked up from the Magma linear code database.

We give hereafter the generating and parity matrices G and H as well as the matrices $G^T(GG^T)^{-1}$ and $H^T(HH^T)^{-1}$ used for the projections from $\mathbb{F}_2^n$ to $\mathcal{C}$ and $\mathcal{D}$ respectively and finally the matrices L and L' corresponding to the standard and *masked* XTIME linear application.

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$G^T(GG^T)^{-1} = \begin{pmatrix} 0&0&1&1&1&0&0&1\\ 0&0&1&0&0&1&0&1\\ 1&1&1&1&1&0&1&0\\ 1&0&1&0&1&1&0&0\\ 1&0&1&1&1&1&1&0\\ 0&1&0&1&1&1&1&1\\ 0&0&1&0&1&1&1&1\\ 1&1&0&0&0&1&1&0\\ 0&1&0&1&1&0&1&0\\ 1&1&0&0&0&1&0&1\\ 0&1&1&0&0&0&1&0\\ 0&0&1&1&0&0&0&1\\ 1&1&0&0&1&0&0&1\\ 1&0&0&0&1&1&0&0\\ 1&0&0&1&0&1&1&1\\ 0&1&1&1&0&0&1&0 \end{pmatrix}, H^T(HH^T)^{-1} = \begin{pmatrix} 0&1&0&0&1&1&1&0\\ 1&1&1&0&1&0&0&1\\ 0&0&1&1&0&0&0&1\\ 1&0&0&1&0&0&1&1\\ 1&0&0&0&1&1&0&0\\ 0&1&0&0&0&1&1&0\\ 1&0&1&0&0&0&1&1\\ 0&1&0&1&1&0&1&0\\ 1&1&1&0&0&0&1&1\\ 1&0&1&1&0&1&0&0\\ 1&1&0&1&1&0&1&0\\ 0&1&1&0&1&1&0&1\\ 0&0&1&1&1&1&0&1\\ 0&1&0&1&1&0&1&1\\ 1&0&1&0&0&1&1&0\\ 1&0&0&1&1&1&0&1 \end{pmatrix}$$

$$L = \begin{pmatrix} 0&0&0&1&1&0&1&1\\ 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0 \end{pmatrix}$$

Recalling $L' = G^T(GG^T)^{-1}LG \oplus H^T(HH^T)^{-1}H$,

$$L' = \begin{pmatrix} 1&1&0&0&1&0&1&1&1&1&0&1&0&0&1&0\\ 0&0&1&0&1&1&1&1&1&1&1&0&1&1&0&0\\ 0&0&1&1&0&1&0&1&0&1&0&0&0&0&1&1\\ 1&1&1&1&1&1&1&1&0&1&1&1&0&0&1&0\\ 1&1&0&1&0&0&0&1&0&1&0&1&0&0&1&1\\ 1&1&1&0&0&1&0&1&1&1&0&0&1&0&1&0\\ 0&1&1&1&0&0&1&1&0&0&1&1&0&0&1&0\\ 0&1&0&1&0&0&0&0&1&1&1&1&1&1&1&0\\ 1&1&1&0&1&1&1&0&0&0&1&0&0&1&0&1\\ 0&1&0&1&0&1&0&0&1&0&0&1&1&0&1&1\\ 1&0&1&0&0&1&1&0&1&1&1&1&1&0&0&1\\ 0&1&0&1&0&0&1&1&1&0&0&1&0&1&1&1\\ 0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&1\\ 1&0&0&0&1&1&1&1&1&1&1&1&0&1&0&1\\ 1&0&1&0&0&0&1&0&0&0&1&0&1&0&0&1\\ 1&0&0&1&0&1&1&0&0&1&1&1&0&0&1&0 \end{pmatrix}$$

A.2 CHES 2014 AES Infective Attacks Success Probabilities

We describe here the success probabilities of the different attacks described in Sec. 2.5.

A.2.1 Probability of Success of Attack 1

The success of Attack 1 depends on the chances for the attacker to fault the increment of i in the loop corresponding to the last redundant round execution. Let us denote by e_1 the event of faulting the last redundant round during the q-th loop. The probability $\mathcal{P}(e_1)$ is thus the probability of having a bit-string $rstr$ that contains 20 "1" on the first $q-1$ positions, one bit set on the q-th position and a last sub-string with only one bit set on the last $t-q$ positions. The corresponding number of such sub-strings being equal to $\binom{q-1}{20}$, $\binom{1}{1}$ and $\binom{t-q}{1}$ respectively, this leads us to $\binom{q-1}{20}\binom{t-q}{1}$ exploitable $rstr$ strings.

By dividing this value by the number of possible $rstr$ strings, we obtain the probability $\mathcal{P}(e_1)$:

$$\mathcal{P}(e_1) = \frac{\binom{q-1}{20}\binom{t-q}{1}}{\binom{t}{22}} \quad . \tag{A.1}$$

As described in Sec. A.2.5, we then compute by using Eq. A.14 the probability to obtain at least one useful faulty ciphertext by repeating the fault injection r times.

A.2.2 Probability of Success of Attack 2

Let us evaluate the probability that the event e_2 of obtaining a useful faulty ciphertext by setting to zero the variable λ at Step 5 of Alg. 6 happens. The probability $\mathcal{P}(e_2)$ corresponds to the probability of obtaining a string $rstr$ that has 21 bits set on the first $q-1$ positions, a "1" on the q-th position and only "0"'s on the last $t-q$ positions. As we have done in Sec. A.2.1, we compute this probability as the number of such strings divided by the total number of possible $rstr$ strings. As there is only one possibility that the last $t-(q-1)$ bits of $rstr$ are exactly "1 0 $\cdots$ 0", we thus obtain:

$$\mathcal{P}(e_2) = \frac{\binom{q-1}{21}}{\binom{t}{22}} \quad , \tag{A.2}$$

As described in Sec. A.2.5, we then compute by using Eq. A.14 the probability to obtain at least one useful faulty ciphertext by repeating the fault injection r times.

A.2.3 Probability of Success of Attack 3

Let us denote by e_3 the event that a random byte error disturbs the string $rstr$ such that it contains only 21 or 20 "1". To evaluate the probability $\mathcal{P}(e_3)$ that the event e_3 occurs, let us assume for the sake of simplicity that the attacker disturbs the least significant byte B of $rstr$ which corresponds to a random byte fault model. By firstly evaluating the case 21, we observe that the probability that a bit-string has exactly 21 bits set on the first $t-8$ positions and the remaining "1" in one of the last 8 positions is:

$$\mathcal{P}(HW(B)=1) = \frac{\binom{t-8}{21}\binom{8}{1}}{\binom{t}{22}} \quad , \tag{A.3}$$

where we denote by $HW(B)$ the Hamming weight of the byte B. Equation (A.3) corresponds to the probability that the last byte of $rstr$ has an Hamming weight equal to 1. By summing the corresponding probabilities for all the Hamming weights between

1 and 8 we obtain the probability that the last byte of $rstr$ has an Hamming weight greater than zero:

$$\mathcal{P}(HW(B) > 0) = \sum_{i=1}^{8} \frac{\binom{t-8}{22-i}\binom{8}{i}}{\binom{t}{22}} \quad . \tag{A.4}$$

Now, let us compute the probability of injecting a random error on a byte of Hamming weight i such that the byte contains only $i-1$ "1" after the disturbance. We thus count for each possible value of B how many 8-bit values e exist such that $HW(B \oplus e) = HW(B) - 1$. This corresponds to the number of possible errors setting to "0" j bits "1" while setting to "1" $j-1$ bits "0". Afterwards we divide the result by the number of possible values for the error e:

$$\begin{aligned} \mathcal{P}(HW(B \oplus e) &= HW(B) - 1|B) \\ &= \frac{\sum_{j=1}^{HW(B)} \binom{HW(B)}{j}\binom{8-HW(B)}{j-1}}{255} \quad . \end{aligned} \tag{A.5}$$

This corresponds to the probability that $HW(B \oplus e) = HW(B) - 1$ by injecting a random error e on a random 8-bit value B.

By combining the two probabilities above, we obtain the probability that $rstr$ contains 21 "1" after a random error injection on the last byte of $rstr$:

$$\mathcal{P}(HW(B \oplus e) = 21) = \sum_{i=1}^{8} \frac{\binom{t-8}{22-i}\binom{8}{i}}{\binom{t}{22}} \sum_{j=1}^{i} \frac{\binom{i}{j}\binom{8-i}{j-1}}{255} \quad . \tag{A.6}$$

For the case where $rstr$ contains only 20 "1", we use the same reasoning and we obtain:

$$\mathcal{P}(HW(B \oplus e) = 20) = \sum_{i=2}^{8} \frac{\binom{t-8}{22-i}\binom{8}{i}}{\binom{t}{22}} \sum_{j=2}^{i} \frac{\binom{i}{j}\binom{8-i}{j-2}}{255} \quad . \tag{A.7}$$

Thus the total probability of disturbing the generation of one byte of $rstr$ such that it contains a total of 21 or 20 "1" is:

$$\mathcal{P}(e_3) = \sum_{i=1}^{8} \frac{\binom{t-8}{22-i}\binom{8}{i}}{\binom{t}{22}} \sum_{j=1}^{i} \frac{\binom{i}{j}\binom{8-i}{j-1}}{255} + \sum_{i=2}^{8} \frac{\binom{t-8}{22-i}\binom{8}{i}}{\binom{t}{22}} \sum_{j=2}^{i} \frac{\binom{i}{j}\binom{8-i}{j-2}}{255} \quad . \tag{A.8}$$

As described in Sec. A.2.5, we then compute by using Eq. A.14 the probability to obtain at least one useful faulty ciphertext by repeating the fault injection r times.

A.2.4 Probability of Success of Attack 4

In the following we denote by e_4 the event that the error e is injected after a cipher round and is such that $q \oplus e > t$. In order to evaluate the probability $\mathcal{P}(e_4)$ we need to compute:

- the probability that the error e leads to $q \oplus e > t$,
- the probability that the attacker disturbs the algorithm after a cipher round and not after a redundant or dummy round.

For the first probability, without loss of generality, we assume that q is coded over one byte which should be the case in practice. We thus obtain that the probability of injecting an 8-bit error e such that $q \oplus e > t$ depends only on t and is given by:

$$\mathcal{P}(q \oplus e > t) = \frac{2^8 - t}{2^8} \ . \tag{A.9}$$

In order to evaluate the second probability we remark that it is equivalent to the probability that the string $rstr$ contains two or three "1" in the first q positions. We recall that $rstr$ is a string with 22 "1" at most. Thus the number of possible strings $rstr$ with only two "1" in the first q positions is:

$$\binom{q}{2}\binom{t-q}{20} \ . \tag{A.10}$$

Summing Eq. A.10 to the number of possible strings $rstr$ with only three "1" in the first q positions we obtain the number of favorable cases for the attacker:

$$\binom{q}{2}\binom{t-q}{20} + \binom{q}{3}\binom{t-q}{22-3} \ . \tag{A.11}$$

By dividing by the total number of possible $rstr$ strings we thus obtain the probability that the algorithm has executed only one cipher round after q rounds:

$$\mathcal{P}(HW(rstr[1,\ldots,q]) \in [2,3]) = \frac{\binom{q}{2}\binom{t-q}{20} + \binom{q}{3}\binom{t-q}{19}}{\binom{t}{22}} \ , \tag{A.12}$$

where $rstr[1,\ldots,q]$ denotes the sub-string of $rstr$ between the first and the q-th position. By combining the two probabilities we obtain:

$$\mathcal{P}(e_4) = \frac{2^8 - t}{2^8} \sum_{i=2}^{3} \frac{\binom{q}{i}\binom{t-q}{22-i}}{\binom{t}{22}} \ , \tag{A.13}$$

which corresponds to the probability that the algorithm returns an exploitable faulty ciphertext by injecting a random error after q rounds.

As described in Sec. A.2.5, we then compute by using Eq. A.14 the probability to obtain at least one useful faulty ciphertext by repeating the fault injection r times.

A.2.5 Attack Repetition Probability

For each attack, we denote by $\mathcal{P}(e_i)$ the probability that event e_i occurs. By assuming that $\mathcal{P}(e_i)$ is independent for each execution we can compute the probability of getting at least one useful faulty ciphertext by repeating the fault injection r times as:

$$\mathcal{P}_r = 1 - (1 - \mathcal{P}(e_i))^r \ . \tag{A.14}$$

A.3 Mutual Information Approximation

In this section, we develop the mutual information between the Hamming weight of uniform random variable X defined over $\mathbb{F}_{2^k}$ and a noisy observation L of this Hamming weight, the noise being modeled by a Gaussian random variable with 0 mean and

standard deviation σ. By definition, we have $L = \mathrm{HW}(X) + B$, with $B \sim \mathcal{N}(0, \sigma^2)$. By definition of the entropy, we have:

$$\mathbb{H}(X|L) = \mathbb{H}(X, L) - \mathbb{H}(L) \ . \tag{A.15}$$

Given X, the distribution of L is Gaussian with standard deviation σ. The differential entropy corresponding to a Gaussian distribution with standard deviation σ is $\ln(\sigma\sqrt{2\pi e})$, whereas the entropy of a uniform random variable over $\mathbb{F}_{2^k}$ is k. We therefore get:

$$\mathbb{H}(X, L) = \mathbb{H}(X) + \mathbb{H}(B) = k + \ln(\sigma\sqrt{2\pi e}) \tag{A.16}$$

The distribution of L is the sum of two distributions. We can model the distribution of $\mathrm{HW}(X)$ as Gaussian with standard deviation $\sigma_k = \sqrt{k}/2$; this is true for large k. The sum of two independent and normally distributed random variables has a normal distribution. Moreover, its mean and variance are simply obtained by summing those of the added variables. In our context, this gives:

$$\mathbb{H}(L) \simeq \ln\left(\sqrt{(\sigma_k^2 + \sigma^2)2\pi e}\right) \simeq \ln\left(\sqrt{(k/4 + \sigma^2)2\pi e}\right) \tag{A.17}$$

Combining (A.15)-(A.17) leads to:

$$\begin{aligned}\mathbb{H}(X \mid L) &\simeq k + \ln(\sigma\sqrt{2\pi e}) - \ln\left(\sqrt{(k/4 + \sigma^2)2\pi e}\right) \simeq k + \frac{1}{2}\ln\left(\frac{\sigma^2}{k/4 + \sigma^2}\right) \\ &\simeq k + \frac{1}{2}\ln\left(\frac{1}{1 + k/(4\sigma^2)}\right) \simeq k - \frac{k}{8\sigma^2} \ ,\end{aligned}$$

after approximating $\ln(1 - U)$ by $-U$, and $k/(k + 4\sigma^2)$ by $k/4\sigma^2$ (which is true when $\sigma^2 \gg k$). Eventually, the amount of information $\mathrm{I}(X; L) \doteq \mathbb{H}(X) - \mathbb{H}(X \mid L)$ given by the noisy Hamming weight leakage can be approximated for $\sigma^2 \gg k$ by:

$$\mathrm{I}(X; L) \simeq \frac{k}{8\sigma^2}$$

A.4 Proof of Lemma 2

Our proof is essentially the same as in [11]. We construct two sets I and J corresponding to the input shares of $x^\star$ and $y^\star$ respectively. We denote by M_{ij} the result of the subroutine $\mathsf{MatMult}((x_1, \ldots, x_n), (y_1, \ldots, y_n))$. From the definition of MatMult and RefreshMasks, it is easy to see that each M_{ij} can be perfectly simulated from x_i and y_j; more generally any internal variable within MatMult can be perfectly simulated from x_i and/or y_j for some i and j; for this it suffices to simulate the randoms in RefreshMasks exactly as they are generated in RefreshMasks.

We divide the internal probes in 4 groups. The four groups are processed separately and sequentially, that is we start with all probes in Group 1, and finish with all probes in Group 4.

- Group 1: If M_{ii} is probed, add i to I and J.
- Group 2: If $r_{i,j}$ or $c_{i,j}$ is probed (for any $i \neq j$), add i to I and J.

Note that after the processing of Group 1 and 2 probes, we have $I = J$; we denote by U the common value of I and J after the processing of Group 1 and 2 probes.

- Group 3: If $M_{ij} \oplus r_{i,j}$ is probed: if $i \in U$ or $j \in U$, add $\{i, j\}$ to both I and J.

- Group 4: If M_{ij} is probed (for any $i \neq j$), then add i to I and j to J. If some probe in MatMult requires the knowledge of x_i and/or y_j, add i to I and/or j to J.

We have $|I| \leq t_1$ and $|J| \leq t_1$, since for every probe we add at most one index in I and J. The simulation of probed variables in groups 1 and 4 is straightforward. Note that for $i < j$, the variable r_{ij} is used in all partial sums c_{ik} for $k \geq j$; moreover r_{ij} is used in $r_{ij} \oplus M_{ij}$, which is used in r_{ji}, which is used in all partial sums c_{jk} for $k \geq i$. Therefore if $i \notin U$, then r_{ij} is not probed and does not enter in the computation of any probed c_{ik}; symmetrically if $j \notin U$, then r_{ji} is not probed and does not enter in the computation of any probed c_{jk}.

For any pair $i < j$, we can now distinguish 4 cases:

- Case 1: $\{i, j\} \in U$. In that case, we can perfectly simulate all variables r_{ij}, M_{ij}, $M_{ij} \oplus r_{ij}$, M_{ji} and r_{ji}. In particular, we let $r_{ij} \leftarrow \mathbb{F}_{2^k}$, as in the real circuit.

- Case 2: $i \in U$ and $j \notin U$. In that case we simulate $r_{ij} \leftarrow \mathbb{F}_{2^k}$, as in the real circuit. If $M_{ij} \oplus r_{i,j}$ is probed (Group 3), we can also simulate it since $i \in U$ and $j \in J$ by definition of the processing of Group 3 variables.

- Case 3: $i \notin U$ and $j \in U$. In that case r_{ij} has not been probed, nor any variable c_{ik}, since otherwise $i \in U$. Therefore r_{ij} is not used in the computation of any probed variable (except r_{ji}, and possibly $M_{ij} \oplus r_{i,j}$). Therefore we can simulate $r_{ji} \leftarrow \mathbb{F}_{2^k}$; moreover if $M_{ij} \oplus r_{ij}$ is probed, we can perfectly simulate it using $M_{ij} \oplus r_{ij} = M_{ji} \oplus r_{ji}$, since $j \in U$ and $i \in J$ by definition of the processing of Group 3 variables.

- Case 4: $i \notin U$ and $j \notin U$. If $M_{ij} \oplus r_{i,j}$ is probed, since r_{ij} is not probed and does not enter into the computation of any other probed variable, we can perfectly simulate such probe with a random value.

From cases 1, 2 and 3, we obtain that for any $i \neq j$, we can perfectly simulate any variable r_{ij} such that $i \in U$. This implies that we can also perfectly simulate all partial sums c_{ik} when $i \in U$, including the output variables c_i. Finally, all probed variables are perfectly simulated.

We now consider the simulation of the output variables c_i. We must show how to simulate c_i for all $i \in \mathcal{O}$, where $\mathcal{O}$ is an arbitrary subset of $[1, n]$ such that $t_1 + |\mathcal{O}| < n$. For $i \in U$, such variables are already perfectly simulated, as explained above. We now consider the output variables c_i with $i \notin U$. We construct a subset of indices V as follows: for any probed Group 3 variable $M_{ij} \oplus r_{ij}$ such that $i \notin U$ and $j \notin U$ (this corresponds to Case 4), we put j in V if $i \in \mathcal{O}$, otherwise we put i in V. Since we have only considered Group 3 probes, we must have $|U| + |V| \leq t_1$, which implies $|U| + |V| + |\mathcal{O}| < n$. Therefore there exists an index $j^\star \in [1, n]$ such that $j^\star \notin U \cup V \cup \mathcal{O}$. For any $i \in \mathcal{O}$, we can write:

$$c_i = M_{ii} \oplus \bigoplus_{j \neq i} r_{ij} = r_{i,j^\star} \oplus \left(M_{ii} \oplus \bigoplus_{j \neq i, j^\star} r_{ij} \right)$$

We claim that neither $r_{i,j^\star}$ nor $r_{j^\star,i}$ do enter into the computation of any probed variable or other $c_{i'}$ for $i' \in \mathcal{O}$. Namely $i \notin U$ so neither $r_{i,j^\star}$ nor any partial sum c_{ik} was probed; similarly $j^\star \notin U$ so neither $r_{j^\star,i}$ nor any partial sum $c_{j^\star,k}$ was probed, and the output $c_{j^\star}$ does not have to be simulated since by definition $j^\star \notin \mathcal{O}$. Finally if $i < j^\star$ then $M_{i,j^\star} \oplus r_{i,j^\star}$ was not probed since otherwise $j^\star \in V$ (since $i \in \mathcal{O}$); similarly if $j^\star < i$ then $M_{j^\star,i} \oplus r_{j^\star,i}$ was not probed since otherwise we would have $j^\star \in V$ since $j^\star \notin \mathcal{O}$. Therefore, since neither $r_{i,j^\star}$ nor $r_{j^\star,i}$ are used elsewhere, we can perfectly simulate c_i by generating a random value. This proves the Lemma.

A.5 Generalization of Mask Refreshing for Arbitrary n

In this section we describe the generalization of our new mask refreshing algorithm from Section 3.2.9 to arbitrary n. We note that to ensure the t-SNI property such generalization is not completely straightforward; for example if at lines 16 and 17 of Algorithm 18 below we would replace $\lfloor n/2 \rfloor$ by $\lceil n/2 \rceil$, then for $n = 3$ we would have $d_3 = a_3$ and the algorithm could not be t-SNI.

Algorithm 18: RefreshMasks

Require: $a_1, \ldots, a_n$
Ensure: $d_1, \ldots, d_n$ such that $\bigoplus_{i=1}^{n} d_i = \bigoplus_{i=1}^{n} a_i$

1: **if** $n = 1$ **then**
2: **return** (a_1)
3: **end if**
4: **if** $n = 2$ **then**
5: $r \leftarrow^{\$} \{0,1\}^k$
6: **return** $(a_1 \oplus r, a_2 \oplus r)$
7: **end if**
8: **for** $i = 1$ **to** $\lfloor n/2 \rfloor$ **do**
9: $r_i \leftarrow^{\$} \{0,1\}^k$
10: $b_i \leftarrow a_i \oplus r_i$
11: $b_{\lfloor n/2 \rfloor + i} \leftarrow a_{\lfloor n/2 \rfloor + i} \oplus r_i$ $\{b_i \oplus b_{\lfloor n/2 \rfloor + i} = a_i \oplus a_{\lfloor n/2 \rfloor + i}\}$
12: **end for**
13: **if** $n \bmod 2 = 1$ **then**
14: $b_n \leftarrow a_n$
15: **end if**
16: $(c_1, \ldots, c_{\lfloor n/2 \rfloor}) \leftarrow \mathsf{RefreshMasks}(b_1, \ldots, b_{\lfloor n/2 \rfloor})$
17: $(c_{\lfloor n/2 \rfloor + 1}, \ldots, c_n) \leftarrow \mathsf{RefreshMasks}(b_{\lfloor n/2 \rfloor + 1}, \ldots, b_n)$
18: **for** $i = 1$ **to** $\lfloor n/2 \rfloor$ **do**
19: $r_i \leftarrow^{\$} \{0,1\}^k$
20: $d_i \leftarrow c_i \oplus r_i$
21: $d_{\lfloor n/2 \rfloor + i} \leftarrow c_{\lfloor n/2 \rfloor + i} \oplus r_i$ $\{d_i \oplus d_{\lfloor n/2 \rfloor + i} = c_i \oplus c_{\lfloor n/2 \rfloor + i}\}$
22: **end for**
23: **if** $n \bmod 2 = 1$ **then**
24: $d_n \leftarrow c_n$
25: **end if**
26: **return** $(d_1, \ldots, d_n)$

The following Lemma shows that our new RefreshMasks algorithm is still t-SNI for

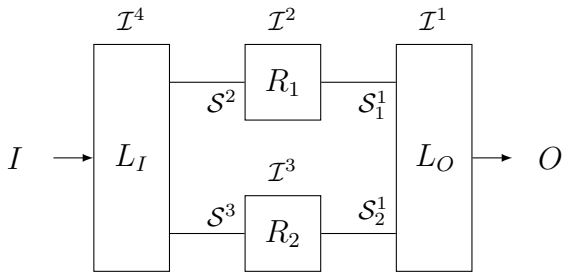


Figure A.1 – RefreshMasks as composition of gadgets

arbitrary n.

Lemma 7 (t-SNI of RefreshMasks)**.** *Let $(a_i)_{1\le i\le n}$ be the input shares of the* RefreshMasks *operation, and let $(d_i)_{1\le i\le n}$ be the output shares. For any set of t intermediate variables and any subset $|O| \le t_O$ of output shares such that $t + t_O < n$, there exists a subset I of indices with $|I| \le t$, such that the distribution of those t intermediate variables as well as the output shares $d_{|O}$ can be perfectly simulated from $a_{|I}$.*

Proof. We show how to adapt the proof of Lemma 3 to arbitrary n. The case $n = 2$ is the same as previously. We now consider the algorithm with $n \ge 3$ shares.

As previously we start with the L_O gadget. Note that for even n the L_O gadget is the same as in the proof of Lemma 3, but for odd n there is a special wire corresponding to $d_n = c_n$ We show that for the input indices $\mathcal{S}_1^1$ and $\mathcal{S}_2^1$ required to simulate L_O, we always have:

$$|\mathcal{S}_1^1| \le t_1 + \lfloor t_O/2 \rfloor, \quad |\mathcal{S}_2^1| \le t_2 + \lceil t_O/2 \rceil \tag{A.18}$$

for some positive integers t_1, t_2, with $t_1 + t_2 \le |\mathcal{I}^1|$; this is an adaptation of (3.21) for arbitrary n. As previously we assume that within $\mathcal{I}^1$ only the intermediate variables r_i, c_i and $c_{\lfloor n/2 \rfloor + i}$ are probed, and not the output variables $c_i \oplus r_i$ and $c_{\lfloor n/2 \rfloor + i} \oplus r_i$, since such output variables can be equivalently obtained from $\mathcal{O}$; similarly if $n \bmod 2 = 1$, we assume that $\mathcal{I}^1$ does not contain the variable $d_n = c_n$.

We first consider the L_O gadget without the wire $d_n = c_n$ when $n \bmod 2 = 1$, which we denote by L'_O; we denote by $\mathcal{O}'$ the corresponding set of output variables that must be simulated, and $t'_O = |\mathcal{O}'|$. For $n = 0 \bmod 2$ we have $L'_O = L_O$ and $t'_O = t_O$. We can then apply Lemma 4 as previously for all $1 \le i \le \lfloor n/2 \rfloor$ on each set of intermediate variables $\{c_i, c_{\lfloor n/2 \rfloor + i}, r_i\}$ and output variables $\{c_i \oplus r_i, c_{\lfloor n/2 \rfloor + i} \oplus r_i\}$; summing the inequalities, we obtain that the gadget L'_O can be perfectly simulated from two sets of input indices $\mathcal{S}_1^1 \subset \{1, \dots, \lfloor n/2 \rfloor\}$ and $\mathcal{S'}_2^1 \subset \{\lfloor n/2 \rfloor + 1, \dots, 2\lfloor n/2 \rfloor\}$, such that $|\mathcal{S}_1^1| \le t_1 + t'_O/2$ and $|\mathcal{S'}_2^1| \le t_2 + t'_O/2$ for some positive integers t_1, t_2, with $t_1 + t_2 \le |\mathcal{I}^1|$. Since $|\mathcal{S}_1^1|$ and $|\mathcal{S'}_2^1|$ are integers, we can use the bounds:

$$|\mathcal{S}_1^1| \le t_1 + \lfloor t'_O/2 \rfloor, \quad |\mathcal{S'}_2^1| \le t_2 + \lfloor t'_O/2 \rfloor \tag{A.19}$$

We now consider the full gadget L_O. When $n = 0 \bmod 2$, we have $L'_O = L_O$, so we can take $\mathcal{S}_2^1 = \mathcal{S'}_2^1$; then with $t'_O = t_O$, from (A.19) we obtain the bounds in (A.18). When $n \bmod 2 = 1$, we must consider the additional wire $d_n = c_n$. We distinguish two cases. If $n \notin \mathcal{O}$, then we have $t'_O = t_O$, and we can take $\mathcal{S}_2^1 = \mathcal{S'}_2^1$; therefore we obtain again (A.18). Finally if $n \in \mathcal{O}$, we have $t'_O = t_O - 1$, and we can take $\mathcal{S}_2^1 = \mathcal{S'}_2^1 \cup \{n\}$. This gives from (A.19):

$$|\mathcal{S}_2^1| \le |\mathcal{S'}_2^1| + 1 \le t_2 + \lfloor (t_O - 1)/2 \rfloor + 1 \le t_2 + \lceil t_O/2 \rceil$$

which gives again (A.18).

We now consider the R_1 and R_2 gadgets. For R_1 the t-SNI condition is:

$$|\mathcal{I}^2| + |\mathcal{S}_1^1| < \lfloor n/2 \rfloor \tag{A.20}$$

and when such condition is satisfied we have that the probed intermediate variables in $\mathcal{I}^2$ and the output variables in $\mathcal{S}_1^1$ can be simulated from a subset of input indices $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |\mathcal{I}^2|$. Similarly for gadget R_2 the t-SNI condition is:

$$|\mathcal{I}^3| + |\mathcal{S}_2^1| < \lceil n/2 \rceil \tag{A.21}$$

Note that when $n = 3$ we can still apply Condition (A.20) on R_1; in that cases it requires $|\mathcal{I}^2| = |\mathcal{S}_1^1| = 0$, and then $|\mathcal{S}^2| = 0$.

Consider now the following two inequalities:

$$|\mathcal{I}^2| + t_1 + \lfloor t_O/2 \rfloor < \lfloor n/2 \rfloor \tag{A.22}$$
$$|\mathcal{I}^3| + t_2 + \lceil t_O/2 \rceil < \lceil n/2 \rceil \tag{A.23}$$

As previously, we have using (A.18) that Inequality (A.22) implies Condition (A.20) for gadget R_1, and similarly Inequality (A.23) implies Condition (A.21) for R_2. And as previously, at least one of the two inequalities (A.22) or (A.23) must be satisfied, since otherwise, using $\lfloor x/2 \rfloor + \lceil x/2 \rceil = x$ for any $x \in \mathbb{Z}$, we obtain using $t_1 + t_2 \leq |\mathcal{I}^1|$:

$$n \leq |\mathcal{I}^2| + t_1 + \lfloor t_O/2 \rfloor + |\mathcal{I}^3| + t_2 + \lceil t_O/2 \rceil \leq |\mathcal{I}^2| + |\mathcal{I}^3| + |\mathcal{I}^1| + t_O \leq t + t_O$$

which contradicts the bound $t + t_O < n$. This again shows that the two gadgets R_1 and R_2 cannot be both saturated.

If both inequalities (A.22) and (A.23) are satisfied, then both gadgets R_1 and R_2 are non-saturated, and by recursively applying Lemma 3 on both gadgets, we get $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ and $|\mathcal{S}^3| \leq |\mathcal{I}^3|$. One can therefore let $I = \mathcal{S}^2 \cup \mathcal{S}^3$ and simulate the L_I gadget as in the real circuit; we then have as required:

$$|I| \leq |\mathcal{S}^2| + |\mathcal{S}^3| \leq |\mathcal{I}^2| + |\mathcal{I}^3| \leq t$$

Assume now that (A.22) is satisfied and (A.23) is not. Then R_1 is non-saturated and we can apply Lemma 3 on R_1, which gives as previously $|\mathcal{S}^2| \leq |\mathcal{I}^2|$. For R_2 we take $\mathcal{S}^3 = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$, which means that all inputs of R_2 must be simulated. We now consider the L_I gadget. By applying Lemma 5 for all $1 \leq i \leq \lfloor n/2 \rfloor$ on each set of intermediate variables $\{a_i, a_{\lfloor n/2 \rfloor + i}, r_i\}$ and output variable $a_i \oplus r_i$, where all output variables $a_{\lfloor n/2 \rfloor + i} \oplus r_i$ must be simulated, and by summing the inequalities, we construct $I \subset \{1, \ldots, n\}$ such that:

$$|I| \leq 2 \cdot |\mathcal{S}^2| + |\mathcal{I}^4| + (n \bmod 2) \leq 2 \cdot |\mathcal{I}^2| + |\mathcal{I}^4| + (n \bmod 2) \tag{A.24}$$

where the additional term $(n \bmod 2)$ comes from the wire $b_n = a_n$ when $n \bmod 2 = 1$; namely we must have $n \in I$ since $n \in \mathcal{S}^3$.

It remains to show that $|I| \leq t$. Since by assumption (A.22) is satisfied and (A.23) is not, we have:

$$\begin{aligned} |\mathcal{I}^2| + t_1 + \lfloor t_O/2 \rfloor &< \lfloor n/2 \rfloor \\ |\mathcal{I}^3| + t_2 + \lceil t_O/2 \rceil &\geq \lceil n/2 \rceil = \lfloor n/2 \rfloor + (n \bmod 2) \end{aligned}$$

which gives:

$$|\mathcal{I}^2| + t_1 + \lfloor t_O/2 \rfloor + (n \bmod 2) < |\mathcal{I}^3| + t_2 + \lceil t_O/2 \rceil$$

which implies:

$$|\mathcal{I}^2| + (n \bmod 2) < |\mathcal{I}^3| + t_2 + (t_O \bmod 2)$$

and therefore:

$$|\mathcal{I}^2| + (n \bmod 2) \leq |\mathcal{I}^3| + t_2 \leq |\mathcal{I}^3| + |\mathcal{I}^1|$$

Then from (A.24) we obtain:

$$|I| \leq |\mathcal{I}^2| + |\mathcal{I}^3| + |\mathcal{I}^1| + |\mathcal{I}^4| \leq t$$

as required.

Assume conversely that (A.23) is satisfied and (A.22) is not. Then R_2 is non-saturated and we can apply Lemma 3, which gives as previously $|\mathcal{S}^3| \leq |\mathcal{I}^3|$. For R_1 we take $\mathcal{S}^2 = \{1, \ldots, \lfloor n/2 \rfloor\}$, which means that all inputs of R_1 must be simulated. We now consider the L_I gadget. By applying Lemma 5 for all $1 \leq i \leq \lfloor n/2 \rfloor$ on each set of intermediate variables $\{a_i, a_{\lfloor n/2 \rfloor + i}, r_i\}$ and output variable $a_{\lfloor n/2 \rfloor + i} \oplus r_i$, where all output variables $a_i \oplus r_i$ must be simulated, and by summing the inequalities, we construct $I \subset \{1, \ldots, n\}$ such that:

$$|I| \leq 2 \cdot |\mathcal{S}^3| + |\mathcal{I}^4| \leq 2 \cdot |\mathcal{I}^3| + |\mathcal{I}^4| \tag{A.25}$$

Namely when $n \bmod 2 = 1$, if $n \in \mathcal{S}^3$ we can put n in I, and therefore the bound (A.25) still applies.

Since by assumption (A.23) is satisfied and (A.22) is not, we have:

$$\begin{aligned} |\mathcal{I}^2| + t_1 + \lfloor t_O/2 \rfloor &\geq \lfloor n/2 \rfloor \\ |\mathcal{I}^3| + t_2 + \lceil t_O/2 \rceil &< \lceil n/2 \rceil = \lfloor n/2 \rfloor + (n \bmod 2) \end{aligned}$$

which gives:

$$|\mathcal{I}^3| + t_2 + \lceil t_O/2 \rceil < |\mathcal{I}^2| + t_1 + \lfloor t_O/2 \rfloor + (n \bmod 2)$$

which implies:

$$|\mathcal{I}^3| < |\mathcal{I}^2| + t_1 + (n \bmod 2)$$

and therefore:

$$|\mathcal{I}^3| \leq |\mathcal{I}^2| + t_1 \leq |\mathcal{I}^2| + |\mathcal{I}^1|$$

Then from (A.25) we obtain as previously:

$$|I| \leq 2 \cdot |\mathcal{I}^3| + |\mathcal{I}^4| \leq |\mathcal{I}^3| + |\mathcal{I}^2| + |\mathcal{I}^1| + |\mathcal{I}^4| \leq t$$

as required. This terminates the proof of Lemma 7. □ □