Security of cryptosystems against power-analysis attacks

Sonia Belaïd

THALES

Thales Communications & Security
Laboratoire Chiffre

ENS

École Normale Supérieure
Équipe Crypto

École doctorale Sciences Mathématiques de Paris Centre – ED 386
Spécialité : Informatique

Thèse de doctorat

Security of Cryptosystems Against Power-Analysis Attacks

Spécialité : Informatique

présentée et soutenue publiquement le 22 octobre 2015 par

Sonia Belaïd

pour obtenir le grade de

Docteur de l'École normale supérieure

devant le jury composé de

Directeurs de thèse **:**	
Michel Abdalla	**(CNRS et École normale supérieure)**
Pierre-Alain Fouque	**(Université de Rennes 1 et Institut universitaire de France)**
Rapporteurs **:**	
Louis Goubin	**(Université de Versailles Saint-Quentin en Yvelines)**
Elisabeth Oswald	**(University of Bristol, Royaume-Uni)**
Examinateurs **:**	
Gilles Barthe	**(IMDEA Software, Espagne)**
Pascal Paillier	**(CryptoExperts)**
Emmanuel Prouff	**(ANSSI)**
François-Xavier Standaert	**(Université catholique de Louvain-la-Neuve, Belgique)**
Invités **:**	
Éric Garrido	**(Thales Communications & Security)**
Mehdi Tibouchi	**(NTT Secure Platform Lab, Japon)**

Remerciements

Je présente en premier lieu mes sincères remerciements à mes deux directeurs de thèse Michel Abdalla et Pierre-Alain Fouque. Je les remercie sincèrement de m'avoir donné l'opportunité de réaliser cette thèse à l'ENS en parallèle de mon travail chez Thales et de m'avoir guidée tout au long de ces trois ans. Je les remercie pour leurs idées précieuses, leurs enseignements, leurs encouragements, et la liberté qu'ils m'ont octroyée dans mon travail de recherche.

Je tiens ensuite à remercier Éric Garrido qui m'a accueillie au sein du laboratoire Crypto de Thales Communications & Security. Il m'a accordé sa confiance en me confiant un poste d'ingénieur mais a également fait tout son possible pour m'accorder le temps nécessaire à mon projet de thèse. Grâce à lui, j'ai pu combiner recherche et ingénierie dans des conditions idéales. Je suis également très heureuse de sa participation dans mon jury.

Je remercie David Pointcheval, qui m'a accueillie dans l'équipe Crypto de l'ENS. Je lui exprime toute ma gratitude pour m'avoir permis de réaliser cette thèse en me finançant pendant ces trois ans. Je le remercie également sincèrement pour sa disponibilité et sa gentillesse, son bureau ayant toujours été ouvert pour des discussions aussi bien techniques qu'administratives.

Je souhaite remercier sincèrement Emmanuel Prouff. Il m'a donné goût à la recherche en cryptographie et en particulier l'envie de réaliser une thèse. Je le remercie pour ses précieux conseils au fil des années et pour nos discussions enrichissantes autour des attaques par canaux auxiliaires qui m'ont énormément appris et qui m'apprennent encore beaucoup aujourd'hui. Je suis ravie qu'il ait accepté de participer à mon jury.

I would like to sincerely thank Elisabeth Oswald and Louis Goubin who reported my thesis. I am very grateful for the time they dedicated to this review and for their precious remarks to improve my thesis. I guess that the task may be painful and I sincerely thank them for their patience.

Je tiens à exprimer mes sincères remerciements à Elisabeth Oswald et Louis Goubin qui ont accepté de rapporter mon manuscrit de thèse. Je les remercie pour le temps qu'ils ont accepté de consacrer à la lecture de ce manuscrit et pour leurs précieuses remarques qui m'ont permis de l'améliorer. Je devine la difficulté de rapporter une thèse et les remercie sincèrement pour leur patience.

Je remercie également Gilles Barthe pour m'avoir accueillie à Madrid, m'avoir fait découvrir le monde très enrichissant des méthodes formelles et avoir accepté de participer à mon jury de thèse. Je remercie Pascal Paillier pour nos précieux échanges lors de conférences, du projet ANR Prince et des très sympathiques soirées annuelles CryptoExperts. Je suis également ravie de sa participation dans mon jury. J'aimerais également remercier sincèrement François-Xavier Standaert. Nos échanges ont débuté avant ma thèse et il m'a donné de précieux conseils pour ma recherche. Il m'a accueillie à plusieurs reprises à Louvain-la-Neuve et a accepté de travailler avec moi sur deux articles qui m'ont beaucoup appris. Il a toujours fait preuve d'une grande disponibilité et je suis très heureuse qu'il fasse partie de mon jury de thèse. Enfin, je remercie

Mehdi Tibouchi pour accepter de faire le déplacement, je suis ravie de sa participation dans mon jury.

Mes remerciements se dirigent ensuite vers mes co-auteurs pour nos précieuses collaborations qui m'ont énormément appris, tant au niveau technique qu'au niveau rédactionnel : Michel Abdalla, Gilles Barthe, Luk Bettale, Jean-Sébastien Coron, Fabrizio De Santis, Emmanuelle Dottax, François Dupressoir, Pierre-Alain Fouque, Laurie Genelle, Benoît Gérard, Benjamin Grégoire, Vincent Grosso, Johann Heyszl, Jean-Gabriel Kammerer, Stefan Mangard, Marcel Medwed, David Pointcheval, Emmanuel Prouff, Franck Rondepierre, Sylvain Ruhault, Jörn-Marc Schmidt, François-Xavier Standaert, Pierre-Yves Strub, Stefan Tillich et Damien Vergnaud.

J'aimerais aussi adresser des remerciements particuliers à toute l'équipe d'IMDEA et Benjamin Grégoire pour leur disponibilité sans égale sur Skype semaines et week-ends mais également jours et nuits à l'approche des fameuses deadlines.

J'aimerais bien sûr également remercier chaleureusement mes collègues et responsables chez Thales. Je remercie évidemment Éric Garrido, François Larbey et Didier Le Maitre pour leur accueil au laboratoire. Je remercie vivement Guillaume Fumaroli pour m'avoir fait confiance et accueillie pour la première fois en stage. Il m'a beaucoup appris et m'a permis de réaliser des travaux très intéressants. Je remercie Sylvain Lachartre pour m'avoir permis de réitérer l'expérience avec un second stage puis d'intégrer le laboratoire de façon permanente. Je le remercie également pour son soutien et son aide tout au long de ces 5 ans. J'aimerais également remercier Émeline Hufschmitt pour son aide, son soutien et ses très bons conseils. Je la remercie de faire du café du matin un moment privilégié et de toutes les discussions partagées. Je remercie David Lefranc pour son humour et sa bonne humeur. Je le remercie également pour son aide et sa gentillesse et comme Émeline, pour les précieux cafés du matin. Un grand merci également à Renaud pour avoir accepté de relire une partie de ma thèse, mais aussi pour son humour de moins en moins répréhensible et ses ballades en scooter qui me font régulièrement oublier l'horreur de la ligne 13. Merci à Alexandre pour sa gentillesse et son humour, et pour partager avec moi le poids de la génération Y, pour laquelle DEA ne veut pas dire grand chose. Je remercie chaleureusement Philippe Painchault qui m'a toujours aidée à concilier travail et thèse et qui m'a toujours accompagnée lorsque j'en avais besoin. Je tiens à remercier Olivier Orcière pour être un collègue de bureau réellement prévenant et agréable et pour être aussi créatif dans ses jeux de mots. Enfin, je remercie notre dernier venu Olivier Bernard, qui s'est très vite intégré dans notre équipe grâce à sa constante bonne humeur. Je n'oublie pas les thésards : Julia Chaulet, Thomas Prest (qui se retrouvera dans la catégorie ENS) et Thomas Ricosset et les anciens : Christopher Goyet, Aurore Guillevic et Ange Martinelli. Merci en particulier à Aurore pour son aide tout au long de mes stages et de mes débuts à l'ENS et chez Thales. Enfin, j'aimerais remercier chaleureusement les cryptologues de la DGA-MI avec qui j'ai eu la chance d'échanger sur mon sujet de thèse.

Dans le même esprit, j'aimerais adresser de chaleureux remerciements à mes collègues de l'ENS. Je remercie tous les thésards et post-doc qui ont partagé l'open space avec moi, notamment Adrian, Alain, Anca, Aurore, Fabrice, Florian, Geoffroy, Houda, Itai, Léo, Liz, Mario, Pierre-Alain, Pierrick, Rafael, Raphael, Romain, Sylvain, Tancrède, Thierry, Thomas, Thomas et tous ceux qui passent nous voir quand ils en ont l'occasion. Je remercie également tous les permanents et chercheurs associés : David Pointcheval, Michel Abdalla, Vadim Lyubashevsky, David Naccache, Damien Vergnaud, Hoeteck Wee, Céline Chevalier et Duong-Hieu Phan.

Je remercie pour leur aide indispensable Lydie Pezant, Nathalie Gaudechoux, Joëlle Isnard, Michelle Angely, Lise-Marie Bivard, Valérie Mongiat et Jacques Beigbeder. Grâce à eux, j'ai pu réaliser des missions, recevoir un financement et travailler dans les meilleures conditions possibles.

Je voudrais aussi remercier chaleureusement mes anciens collègues chez Oberthur : Philippe Andouard, Guillaume Barbu, Alberto Battistello, Luk Bettale, Guillaume Dabosville, Paul Dischamp, Julien Doget, Emmanuelle Dottax, Christophe Giraud, Nicolas Morin, Robert Naciri, Gilles Piret, Emmanuel Prouff, Soline Renner, Franck Rondepierre, Michele Sartori, Yannick Sierra et Rina Zeitoun. En particulier, j'aimerais remercier Christophe pour ses précieux conseils techniques et son écoute. Il m'a beaucoup appris et c'est un réel plaisir de le revoir au détour de conférences, quand cela est possible. Je souhaite remercier également Rina, qui m'a très gentiment accueillie chez Oberthur, qui m'a toujours écoutée et soutenue et avec qui je continue à partager des moments très agréables malgré nos chemins différents. J'espère que nos départs respectifs de Jussieu et de l'ENS ne changeront rien à cette amitié.

J'aimerais aussi remercier plusieurs cryptologues rencontrés ou re-rencontrés lors de conférences ou de projets. Je remercie Tancrède Lepoint pour tous ces bons moments passés depuis notre entrée à Poincaré. Je remercie Matthieu Rivain pour nos précieuses discussions sur les attaques par canaux auxiliaires, mais aussi pour tous les moments en conférences/vacances. Je remercie également Thomas Roche et Victor Lomné qui sont également de très belles rencontres de cette thèse et que je prends plaisir à revoir à chaque conférence. Un grand merci aussi à Christina Boura que j'ai rencontrée à sa propre soutenance et que j'ai recroisée avec grand plaisir à Versailles et en conférence. Je la remercie, tout comme Valentin Suder, pour leur aide et soutien. Je remercie également Frédéric De Portzamparc pour son humour, sa gentillesse et sa compagnie lors des conférences. Je remercie Jean-Christophe Zapalowicz pour tous les bons moments passés dans le cadre de cette thèse, mais comme Matthieu et Renaud aussi dans des contextes moins professionnels. Je remercie enfin Guénaël Renault pour avoir co-organisé, avec Emmanuel et Matthieu, une édition de CHES tout à fait mémorable.

Dans un esprit plus personnel, j'aimerais remercier mes amis qui m'ont soutenue mais ont surtout su me distraire pendant ces trois ans. Je remercie les nancéien(ne)s que je suis ravie de revoir à chaque séjour à la maison. En particulier, je remercie Liloo de profiter de ses missions (puissent elles être plus nombreuses) pour venir nous voir régulièrement mais surtout d'être une amie exceptionnelle. Je remercie également les grenoblois (qui ne le sont pas vraiment) mais qui depuis l'Ensimag sont devenus des amis très précieux. En particulier, je remercie Clara qui a toujours été très présente même depuis l'autre bout du monde. Je la remercie de m'avoir fait troquer la crypto pour la plongée aux Philippines. Je remercie aussi les palois pour m'avoir acceptée dans le clan très fermé des béarnais mais qui s'étend tout de même aux basques. Enfin, je remercie les bretons que je suis ravie d'avoir rencontrée en arrivant à Paris. En particulier, Florence a joué un rôle majeur dans cette thèse en venant jusqu'à la rue d'Ulm pour partager des sushis.

J'adresse de profonds remerciements à ma famille qui a fait preuve d'un soutien sans faille durant ses trois années, mais également durant toutes celles qui les ont précédées. Je remercie sincèrement mes parents qui m'ont inspirée au point de réaliser un doctorat dans la même discipline qu'eux. Ils m'ont toujours guidée, encouragée et aidée. Je les remercie de m'avoir offert de si beaux exemples et de m'avoir permis d'arriver jusqu'ici. Je remercie également ma sœur Nadia qui, même depuis un autre monde (le droit), a toujours fait preuve d'intérêt et m'a encouragée dans mon travail. Enfin, je remercie mes cousines et ma belle-famille qui me font le grand plaisir de venir assister à ma soutenance.

Enfin, pour conclure ces remerciements, j'aimerais exprimer toute ma gratitude à Clément, qui m'a encouragée tout au long de ces trois ans, et ce malgré sa propre thèse. Je le remercie d'avoir supporté les week-ends et soirées studieux, de m'avoir expliqué et ré-expliqué les probabilités, de m'avoir accompagnée dans les (nombreux) moments de stress et surtout du bonheur qu'il m'apporte en partageant ma vie.

Résumé

Les attaques par canaux auxiliaires sont les attaques les plus efficaces contre les systèmes cryptographiques. Contrairement aux attaques classiques qui n'exploitent que les entrées et sorties des algorithmes, elles utilisent également les fuites physiques du composant sous-jacent. Dans cette thèse, nous nous intéressons aux attaques par canaux auxiliaires qui exploitent la consommation de courant des composants pour retrouver les clefs secrètes : les attaques par analyse de courant.

La majorité des attaques par analyse de courant existantes repose sur l'observation de variables dépendant uniquement de quelques bits de secret avec la stratégie diviser-pour-régner. Dans cette thèse, nous exhibons de nouvelles attaques, sur des multiplications, qui exploitent l'observation de variables intermédiaires largement dépendantes de grands secrets.

En parallèle, nous nous intéressons aux deux contre-mesures algorithmiques les plus répandues contre ces attaques : les fonctions intrinsèquement résistantes aux fuites physiques et les schémas de masquage. Dans un premier temps, nous définissons un schéma de chiffrement résistant aux fuites physiques. Dans un second temps, nous construisons, à l'aide des méthodes formelles, un outil permettant de vérifier automatiquement la sécurité d'implémentations masquées. Nous exhibons également de nouvelles propriétés de sécurité qui nous permettent de générer une implémentation masquée à partir d'une implémentation non protégée. Finalement, nous présentons une étude de comparaison entre ces deux contre-mesures dans le but d'aider les experts industriels à déterminer la meilleure protection à intégrer dans leurs produits.

mots-clés : attaques par canaux auxiliaires, attaques par analyse de courant, cryptographie résistante aux fuites physiques, masquage aux ordres supérieurs.

Abstract

Side-channel attacks are the most efficient attacks against cryptosystems. While the classical black-box attacks only exploit the inputs and outputs of cryptographic algorithms, side-channel attacks also use the physical leakage released by the underlying device during algorithms executions. In this thesis, we focus on one kind of side-channel attacks which exploits the power consumption of the underlying device to recover the algorithms secret keys : power-analysis attacks.

Most of the existing power-analysis attacks rely on the observations of variables which only depend on a few secret bits using a divide-and-conquer strategy. In this thesis, we exhibit new kinds of attacks which exploit the observation of intermediate variables highly dependent on huge secrets.

We also study two commonly used algorithmic countermeasures against side-channel attacks : leakage-resilient primitives and masking schemes. On the one hand, we define a leakage-resilient encryption scheme based on a regular update of the secret key and we prove its security. On the other hand, we build, using formal methods, a tool to automatically verify the security of masked algorithms. We also exhibit new security and compositional properties which can be used to generate masked algorithms at any security order from their unprotected versions. Finally, we propose a comparison between these two countermeasures in order to help industrial experts to determine the best protection to integrate in their products.

keywords : side-channel attacks, power-analysis attacks, leakage-resilient cryptography, higher-order masking.

Contents

Part I

Introduction on Side-Channel Analysis

Chapter 1

Introduction

Cryptology (science of secret) refers to the protection of communications from evil-minded third parties called *adversaries* or *attackers*. It gathers two fields: *cryptography* and *cryptanalysis*.

Cryptography tackles the problem of protecting sensitive information (*e.g.,* before their transmission on an insecure channel) and is implemented by means of *cryptographic algorithms*. Among other properties, cryptography is used to ensure *confidentiality* (secret data), *authentication* (data origins) and *integrity*. To guarantee confidentiality for instance, sensitive messages, referred as *plaintexts*, are turned into incomprehensible messages, referred as *ciphertexts*, for any unauthorized person. In this case, the algorithm turning plaintexts into ciphertexts is called an *encryption* algorithm and the inverse algorithm to recover the plaintexts from the ciphertexts is called a *decryption* algorithm. These algorithms take also as input a secret value called a *key*. This key is mandatory to recover the plaintext from the ciphertext. In practice, we distinguish between *asymmetric* cryptography with *pairs of public and private keys* and *symmetric* cryptography with *secret keys*. The former can be compared to a mail box system in which everybody can send a message and only the addressee can open (here decrypt) it. The latter is quite different since it requires the prior exchange of a common secret key between two people willing to discuss.

On the contrary, cryptanalysis refers to attacks, when an adversary tries to recover the secret information manipulated in a cryptographic algorithm. This information can either be a key or some key-dependent data that the attacker can use instead of the key itself. Following Kerckhoffs' principles claimed in 1883, we generally assume that the attacker has a perfect knowledge of the cryptographic algorithm. In practice, we often assume that the attacker also has access to the algorithm's inputs and outputs (*e.g.,* plaintexts and ciphertexts for the encryption).

Contents

1.1 Asymmetric Cryptography

Asymmetric cryptography relies on the use of key pairs, each dedicated to a party. The first key of a party is public and can be used by anyone who wants to securely communicate with the party (*e.g.,* to cipher a message for the party in the context of encryption). The second key is private and can only be used by the party itself (*e.g.,* to decipher a message it receives). The term asymmetric directly comes from these differences between senders and receivers which do not exist in symmetric cryptography.

The two keys of a pair are strongly related to allow such exchanges. In particular, theoretically, the private key can be recovered from the public key. However, in practice, asymmetric cryptography relies on difficult mathematical problems according to which it is computationally hard to recover the private key from the public one. Nowadays, two main mathematical problems used to build asymmetric algorithms are the difficulty to factorize the product of two large prime numbers (RSA system [RSA78]) and the discrete logarithm problem (*e.g.,* El Gamal system [Gam85]).

The main possibilities offered by asymmetric cryptography can be gathered in two categories: encryption of messages to ensure confidentiality and electronic signatures to guarantee authenticity.

1.1.1 Encryption Algorithms

We describe here the procedure of encryption and decryption between two parties. As usually done in cryptology, we refer to these two parties as Alice and Bob. In order to send a message to Bob, Alice first needs to get Bob's public key k_B. The latter can be obtained from the Internet or by asking to Bob but it must be done in a secure way to prevent an attacker from exchanging Bob's key with his own and thus from reading Alice's confidential messages. Once Alice securely obtains Bob's public key, she can use it to cipher the message m she wants to address with the asymmetric encryption algorithm ENC_{k_B} associated to this public key (*e.g.,* RSA encryption algorithm). The corresponding ciphertext c is thus transmitted over a potentially insecure channel but it now looks unintelligible for anyone observing the communication. In particular, cryptographers usually refer to an attacker who observes the communication as Eve. At this point, only Bob can decipher the ciphertext c using his private key (denoted by k_B^{-1}) and the decryption algorithm $\mathsf{DEC}_{k_B^{-1}}$ corresponding to the encryption algorithm chosen by Alice. The procedure is illustrated in Figure 1.1.

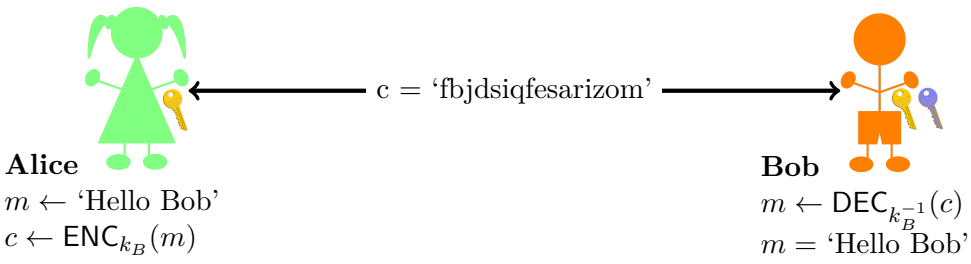


Figure 1.1: Alice sends a ciphertext to Bob using asymmetric encryption

1.1.2 Digital Signatures

Electronic signatures aim to provide the same advantages as handwritten signatures, namely authenticating the author of a message. In order to electronically sign a message m, Alice encrypts it with her private key k_A^{-1} and obtains a signature $\sigma = \mathsf{ENC}_{k_A^{-1}}(m)$. She then sends the plain message m together with its signature σ. When Bob receives this pair, he can use Alice's public key k_A to decrypt the signature σ and verify that it corresponds to m. The procedure is illustrated in Figure 1.2.

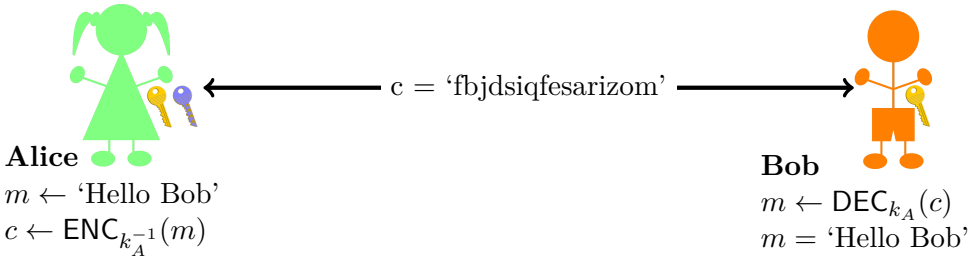


Figure 1.2: Alice sends a message and its signature to Bob using asymmetric cryptography

1.2 Symmetric Cryptography

Symmetric cryptography refers to the protection of the communications between two parties who share the same secret key. For efficiency reasons, symmetric algorithms are generally preferred to asymmetric ones. However, asymmetric algorithms are very useful to generate secret keys that will be shared between only two parties.

To ensure the security of communications with symmetric cryptography, we can use algorithms from different families depending on the required properties. Two widely used families of algorithms are encryption algorithms which provide confidentiality for the manipulated data and *message authentication codes* (MAC for short) which guarantee integrity and authentication of messages.

1.2.1 Encryption Algorithms

To ensure the confidentiality of the messages they exchange, Alice and Bob first need to establish a shared secret key k. Then, when Alice sends a message m to Bob, she first encrypts it using an encryption algorithm ENC_k using the key k. The obtained ciphertext c looks like a random message and only Bob who shares the same key k with Alice is able to decrypt it using a decryption algorithm DEC_k such that $\mathsf{DEC}_k(c) = m$. This procedure is illustrated by Figure 1.3.

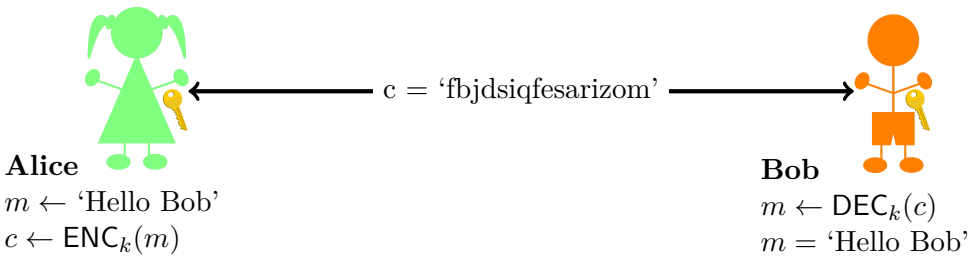


Figure 1.3: Alice sends a ciphertext to Bob using symmetric cryptography

Among the encryption algorithms, the most widely deployed are the *block cipher algorithms* combined with *modes of operation*. The mode of operation usually splits the plaintext into fixed-size blocks. Then, each part is ciphered by the block cipher according to some chaining defined by the mode. The goal is to ensure that two encryptions of the same block of plaintext return two different blocks of ciphertext. Without this mode, the attacker would get information on the global message (*e.g.,* exploiting the characters frequencies).

The most commonly used block cipher algorithm is the AES (Advanced Encryption Standard). It was initially proposed by Joan Daemen and Vincent Rijmen under the name of Rijndael and was selected by the NIST (National Institute of Standards and Technology) in 2000 [AES01] to become the new encryption standard.

1.2.2 Message Authentication Codes

A message authentication code or MAC is the data sent together with a message in order to verify its integrity and its origins. If Alice wants to send a message to Bob and to provide him with these latter guarantees, she can use the secret key that they share with the message to generate a MAC. She can then send both the MAC and the message to Bob. From the message and the secret key he shares with Alice, Bob can verify the MAC and compare the result with the data he received. If both MACs are equal, Bob has the guarantee that the message was not altered and that Alice is actually the one who sent it.

The main difference between MACs and signatures come from the use of the same or different keys. In particular, signatures can be performed only by a single person which owns the private key. This single person cannot deny having signed the document: we call this property the *non-repudiation*. However, in the context of MACs and symmetric cryptography, the secret key is shared and the property of non-repudiation cannot be achieved.

1.3 Attacks on Cryptographic Devices

Usually, we assume that the only secret information that should be protected in cryptography are the secret keys. Without the latter, an attacker should not be able to counteract the cryptographic protections (*e.g.,* she should not be able to usurp someone's identity or to break confidentiality). Therefore, secret keys are normally securely stored in cryptographic devices (*e.g.,* smart cards) which implement the corresponding algorithms.

Once implemented, these algorithms have to thwart two kinds of attacks: *black-box* attacks and *physical attacks* or *side-channel attacks*. Black-box attacks use only the knowledge of the algorithm, as well as the knowledge of some pairs of plaintexts and ciphertexts to recover the key. Most of current cryptographic algorithms are either assumed or proved secure against these attacks. *Physical attacks*, on the other hand, are much more powerful and target the algorithms' implementation. Corresponding attackers can either alter the underlying device to generate abnormal ciphertexts containing key-dependent information or they can observe the device's emanations to observe secret data. These physical attacks can be categorized in many different ways, depending on the adversary's capabilities, the available equipments or their complexity. In the following, we follow the classification proposed in [MOP07] and discuss active versus passive attacks and invasive versus non-invasive attacks.

1.3.1 Active versus Passive Attacks

In an active attack, the targeted device is tampered to reveal information on the secret. The goal of the attacker is to generate abnormal behaviors in order to obtain information which

should be hidden in a normal use. For instance, Ishai, Prabhakaran, Sahai, and Wagner propose in [IPSW06] protections against adversaries with such capabilities. Namely, they show how to transform a circuit into a functionally equivalent one able to detect such tampering and to erase sensitive data in this case.

Conversely, a passive attack only makes the device work as it normally would. The attacker tries to recover the secret information by observing the physical information generated when the targeted algorithm is normally executed by the device.

1.3.2 Invasive versus Non-Invasive Attacks

Invasive attacks, deeply studied in [Sko05], require a permanent action on the device by the attacker. More precisely, the invasive attacker is able to depackage the chip to access its underlying components. Then, she can establish direct contacts (*e.g.,* probes) with the chip to recover the secret key. Active invasive attacks alter the behavior of the device by sending signals for instance. Passive invasive attacks only exploit the direct connection with the device.

Semi-invasive attacks, introduced in [SA03], also lead to a permanent alteration of the device. However, in these cases, the attacker is restricted to the depackaging of the device but she cannot establish a direct contact with it. Nevertheless, an active semi-invasive attacker can still inject faults in the device using rays for instance.

Conversely, a non-invasive attacker can only observe the component externally, that is without modifying it in any way. She is restricted to observing its activity from the outside, using an oscilloscope, a timer, or any tool which does not alter the device. An active non-invasive attacker can also inject faults in the device but cannot depackage it. The most widely deployed attacks remain the passive non-invasive attacks since there are perhaps the less expensive to mount. Among these, the most important ones are the timing attacks [Koc96], the electromagnetic attacks [QS01, GMO01] and the power-analysis attacks [KJJ99]. Nevertheless, there also exist more exotic passive non-invasive attacks. For instance, Shamir and Tromer showed in 2004 how to mount side-channel attacks from variations in acoustic emissions and extended their method with Genkin in [GST13]. Ferrigno and Hlavác [FH08], followed by Schlösser et al. [SNK+12], focused on the optical emissions. Other examples are the exploitation of the PC keyboards sounds, as shown by Asonov and Agrawal [AA04] and improved by Zhuang, Zhou, and Tyga [ZZT09], the temperature (*e.g.,* [BKMN09]) and the leakage of photons [Sko09]. In this thesis, we will only focus on the widely deployed power-analysis attacks.

1.4 Motivation of this Thesis

In this thesis, we want to tackle the large problem of securing cryptographic algorithms against power-analysis attacks, which are perhaps the most studied type of side-channel attacks. This popularity may be explained both by their strength, since they allow a complete recovery of the secret keys in many black-box secure algorithms and by their cost since they do not require an expensive equipment.

In order to obtain secure algorithms against such powerful attacks, it is mandatory to jointly concentrate on new attacks and on improving the protections to which we will refer as *countermeasures*. We noticed that most of the existing attacks are based on the divide-and-conquer strategy. Thus, one of this thesis' goal is to investigate new kinds of attacks based on different strategies so that we can identify unknown security breaches and protect against them. In parallel, we aim to build new constructions that are secure against side-channel attacks without prohibitive complexities to match the device constraints. We also make an effort to automate

the verification of countermeasures in order to guarantee their security and avoid the breaches that are sometimes hidden by erroneous pen-and-paper proofs.

1.5 Outline of This Thesis

The following starts with an introduction on power-analysis attacks. It gathers the necessary background and the related state of the art. Then, our contributions are organized in two parts: new power-analysis attacks on multiplications between large data (*i.e.,* 128 bits) in Part II and the presentation and the evaluation of algorithmic countermeasures in Part III.

Chapter 2

Power Analysis

Most of widely used cryptosystems are secure in the *black-box model* when the adversary is limited to the knowledge of the algorithm and the observation of the inputs and outputs. However, once implemented and executed on a physical device (*e.g.,* a smart card), these cryptosystems may be subject to much more powerful attacks which also exploit the physical emanations of the device (*e.g.,* execution time, temperature, power consumption, electromagnetic radiations). These emanations, which generally depend on the data manipulated in the algorithms, are referred as *physical leakage* and these new attacks are gathered around the term Side-Channel Analysis (SCA for short). They will be developed later in this chapter.

Contents

2.1 Brief History

One of the first side-channel attacks was discovered in the forties. According to the NSA (National Security Agency), a researcher of Bell's laboratories noted, on an oscilloscope plugged in an encryption computer, consumption peaks related to the manipulated data. This discovery led the American government to launch in the fifties the program TEMPEST to study the compromising emanations. Thanks to it, Wim Van Eck published, in 1985, the first paper [Eck85] to explain how to decode the electromagnetic emanations used in video technologies.

The first side-channel attack against a cryptographic implementation was finally published in 1996 by Kocher [Koc96]. The latter shows how to recover secret keys used in well-deployed public-key algorithms such as Diffie-Hellman and RSA (named from its authors Rivest, Shamir,

and Adleman) using only a few executions. The difference between the computations processed for each secret key bit value translates into distinguishable execution times, making the entire secret key easily readable by an attacker. Three years later, Kocher, Jaffe, and Jun published new attacks exploiting, this time, the power consumption [KJJ99].

These first publications motivated the research in this area. Since then, several side-channel attacks and countermeasures are published each year.

2.2 Power-Analysis Attacks

Since the work of [KJJ99], the cryptographic community has shown a strong interest in power-analysis attacks. In this section, we aim to explain how they work. From leakage acquisitions or leakage simulations (*e.g.,* to evaluate the security of an implementation), we show how to recover the secret key based on two main methods: Simple Power Analysis and Differential Power Analysis.

2.2.1 Acquisitions

To mount a power-analysis attack on a device executing a cryptographic algorithm, the attacker Eve needs to plug both a computer to exchange data with the device and an oscilloscope to collect the consumption measures. Figure 2.1 illustrates the procedure when the device is a smart card.

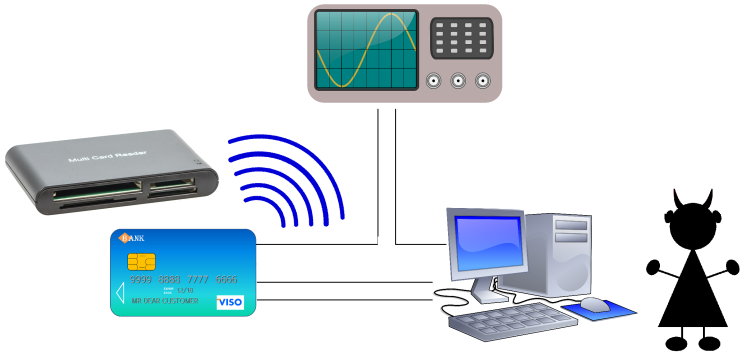


Figure 2.1: Power analysis acquisition procedure on a smart card

The set of consumption measurements collected during the execution of one cryptographic algorithm is generally referred to as its *consumption trace*. The collection of one or several consumption traces will be helpful for Eve to recover the key. An example of power consumption trace is given in Figure 2.2. It corresponds to the execution of a full AES-128 as implemented for the DPA Contest version 2 [VRG]. We can see the 3,253 successive and regular consumption measurements performed during the AES execution with ten similar patterns corresponding to the ten rounds of this algorithm.

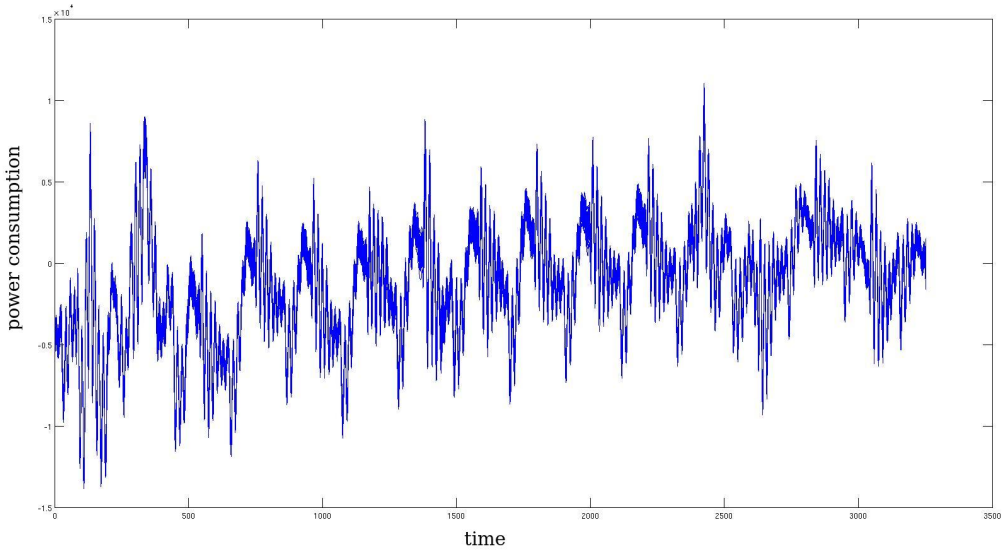


Figure 2.2: Consumption trace of a full AES-128 acquired for the DPA Contest version 2 [VRG]

2.2.2 Simulations

The leakage can vary from one component to another and is sometimes difficult to define for all situations (all operations, all memory storages, etc). Thus, designers generally simulate this leakage with the information they have on the targeted component. In the best case, they have access to the accurate transistors netlist of the component which includes all the transistors and their connections. However, this information generally belongs to the component's designer and is not fully transmitted. Nevertheless, what is important to prevent power-analysis attacks is to capture the relative difference between power consumptions and this can be done without such a precise model. Usually, the leakage is modeled from either the Hamming distance between two manipulated data (*i.e.,* number of bit's differences between them) or the Hamming weight of a manipulated data (*i.e.,* number of bits equal to one). The Hamming distance actually fits well the reality of embedded devices which generally leak the number of bit transitions that occur in a circuit. For instance, it is well adapted to model the power consumption of data buses in micro-controllers or registers in hardware implementations. In the case of registers, two variables v_1 and v_2 consecutively stored in the same register may leak the following value:

$$\mathcal{L}_{\mathsf{HD}}(v_1, v_2) = \mathsf{HD}(v_1, v_2) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2) \tag{2.1}$$

with ε an independent noise which follows a Gaussian law of mean zero and standard deviation σ. This noise reflects both the uncertainty on the leakage model and the difficulty to perform the measurements. Hence, its standard deviation is a useful information to evaluate the uncertainty on the leaking value, or equivalently its entropy. When the designer does not have enough knowledge to predict the transitions, he generally models the leakage with the noisy Hamming weight of the manipulated variables. Indeed, although the leakage rather depends on the transitions, the Hamming weight is not uncorrelated to the leakage in practice. In this case, the leakage of an intermediate variable v is modeled by:

$$\mathcal{L}_{\mathsf{HW}}(v) = \mathsf{HW}(v) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2). \tag{2.2}$$

As mentioned above, the level of noise is important to evaluate the security of an implementation. It is supposed to give information on the uncertainty the attacker might have concerning the leaking values. However, the standard deviation of the leakage does not reveal anything without information on the signal. Assume that the leakage is exactly as modeled by Equation (2.2) and that the manipulated variables are uniformly distributed in $\mathsf{GF}(2^{128})$. Since the Hamming weight of uniform variables follows a binomial law, the variance of the signal for 128-bit data is equal to 32, while it is equal to 2 for 8-bit ones. It is clear that a fixed value of σ cannot be similarly interpreted for both architectures. Therefore, we rather use a more accurate indication called *signal-to-noise ratio* (see [MOP07] for further details) which returns the ratio between the signal variance and the noise variance. If the leakage is modeled by Equation (2.2), the SNR is equal to $\frac{32}{\sigma_{128}^2}$ for 128-bit variables and to $\frac{2}{\sigma_8^2}$ for 8-bit ones. Comparing two implementations on two such architectures, we frequently assume that the SNR are equal and thus that $\sigma_{128} = 4\sigma_8$.

2.2.3 Simple Power Analysis

The easiest side-channel attacks to mount are gathered around the term Simple Power Analysis (SPA for short). They exploit the information observed on one or a very few consumption traces. With the many experiments performed so far, we know that the power consumption of a device generally depends on the state changes and thus on the manipulated data and the processed computations. Thus, we can recognize the power consumption trace corresponding to a classical AES-128 because it realizes ten times the same computations for its ten rounds, which is revealed on the trace by ten barely identical patterns. Furthermore, as shown by Kocher in 1996 [Koc96], if the computations depend on the secret key bit values, a consumption trace may *a priori* directly reveals the entire key value. A drawing of such a trace is given in Figure 2.3. The large (resp. small) peaks represent the power consumption generated during the operations which are performed when the current key bit is equal to one (resp. zero). Identifying the two kinds of peaks reveals the secret key.

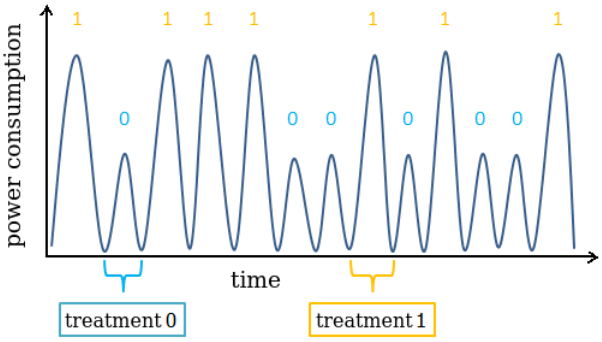


Figure 2.3: SPA on an algorithm in which computations depend on the values of the secret key bits : secret key = 1011100101001

The simple power analysis remains limited in practice. Indeed, removing the dependence between computations and key bits values is often easily achievable and sufficient to prevent it.

2.2.4 Differential Power Analysis

In 1999, Kocher, Jaffe, and Jun published a new power-analysis attack [KJJ99] that was much more powerful than SPA and that relied on a statistical power analysis. The basic idea is to

combine the information collected on several consumption traces acquired during the execution of the targeted algorithm with different inputs.

To perform a power-analysis attack, the attacker Eve generally focuses on a cryptographic algorithm in which the secret key is split in parts of a small number of bits. It is the case for instance in the block cipher algorithms DES (Data Encryption Standard) and AES. Then, Eve acquires a number N of consumption traces corresponding to the execution of the algorithm with N different plaintexts $\{m_i\}_{0 \leqslant i < N}$ and the same secret key. In parallel and with her knowledge of the algorithm, Eve tries to identify a one-bit intermediate variable which depends both on a small number b of key bits and on the known input (or the known output). She can thus compute the value of this intermediate variable (zero or one) for each possible key k^* (2^b possibilities for a b-bit key) and each time for the N plaintexts used for the acquisition. Once these computations are made, Eve can separate the plaintexts into two groups, for each key hypothesis, according to the computed value. Then, the idea is to identify on the consumption traces the time-related point corresponding to the processing of the targeted intermediate variable (*e.g.,* in AES, the attacker can identify the first pattern which should correspond to the first round and test the attack on all the corresponding measurements). Once this point in time is identified, Eve can compute the difference between the consumption means of the two groups for each key hypothesis. The key hypothesis which corresponds to the highest difference of means corresponds with a certain probability to the correct b-bit chunk of the key. Eve can finally repeat the experiment on each remaining part of the key. Kocher, Jaffe and Jun called this attack DPA for *Differential Power Analysis*.

After this major result, other works were published to complete and diversify this technique. In particular, the *prediction model*, which corresponds to the choice of an intermediate bit value to estimate the power consumption in the aforementioned DPA, can be different. For instance, other techniques use the exact value of a larger variable (*e.g.,* 8-bit variable) or its Hamming weight which is actually often well correlated to the power consumption observed when the variable is manipulated. Furthermore, the difference of means used by Kocher, Jaffe, and Jun is now called *distinguisher*. Several distinguishers have been identified so far like the Pearson correlation and the mutual information analysis which are both described hereafter. Although DPA used to refer to an attack with a specific prediction model and the difference of means as distinguisher, we now frequently gather all the statistical power-analysis attacks (independent of the prediction model or their distinguisher) around the term DPA. The generic principle of a DPA to attack an AES is illustrated in Figure 2.4 with $\mathcal{L}$ the leakage function, $v(k, m_i)$ the targeted variable which depends on the key k and each message m_i, $\mathcal{P}$ the prediction model, and $\mathcal{D}$ the distinguisher value for each key chunk hypothesis.

Description of Two Distinguishers

We now make a focus on two widely studied distinguishers: the practical Pearson correlation coefficient and the generic mutual information analysis.

Pearson Correlation Coefficient. The Pearson coefficient of correlation is probably the most widely used distinguisher for power-analysis attacks (see [LSP04, BCO04]). The corresponding DPA is specifically called *Correlation Power Analysis* (CPA for short). Once the leakage has been predicted for an intermediate variable v for each key hypothesis k^* and using a function of prediction $\mathcal{P}$, the attacker has at her disposal a vector of N elements $p = \{p_i\}_{0 \leqslant i < N} = \{\mathcal{P}(v(k^*, m_i))\}_{0 \leqslant i < N}$. She then computes the Pearson coefficient of correlation ρ of this vector p with the vector $\ell = \{\ell_i\}_{0 \leqslant i < N} = \{\mathcal{L}(v(k, m_i))\}_{0 \leqslant i < N}$ of the acquired

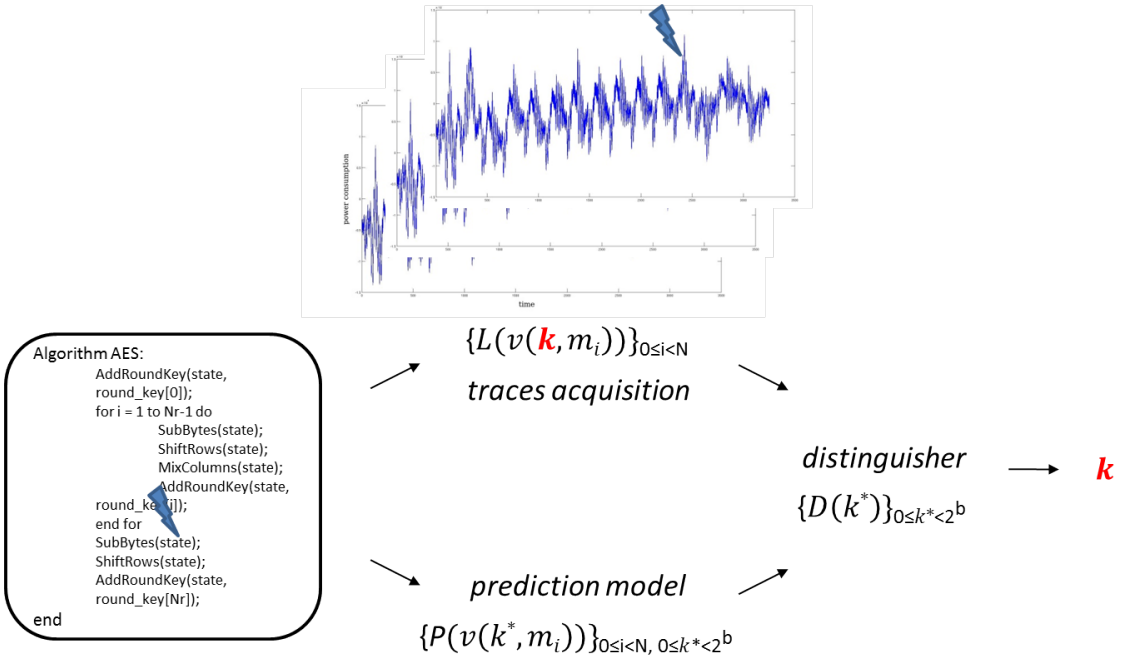


Figure 2.4: DPA principle

leakage values at the point of interest for the same messages:

$$\rho(p,\ell) = \frac{\mathrm{Cov}(p,l)}{\sqrt{\mathrm{Var}(p)\mathrm{Var}(\ell)}} = \frac{\sum_{i=0}^{N-1}(p_i - \mathbb{E}(p_i))(\ell_i - \mathbb{E}(\ell_i))}{\sqrt{\left(\sum_{i=0}^{N-1}(p_i - \mathbb{E}(p_i))^2\right)\left(\sum_{i=0}^{N-1}(\ell_i - \mathbb{E}(\ell_i))^2\right)}}. \tag{2.3}$$

A value close to 1 or -1 indicates that there exists a linear equation to approximate the relationship between the leakage and the predictions p. On the contrary, a correlation equal to zero reveals the absence of linear correlation between the data. In our side-channel analysis context, the key hypothesis k^* which maximizes the result of this correlation coefficient is assumed to be the correct key hypothesis.

To illustrate this procedure, we mount a CPA on the consumption traces acquired during the execution of an AES-128 in the context of the second version of the DPA Contest [VRG]. Knowing that there is only one round of AES per clock cycle, we can assume that the internal state is stored between each round. Since the storage in memory usually leaks more than the other operations and since we know the ciphertexts, we choose to target the whole last round which processes the different bytes separately. The operations on each byte of the 16-byte internal state $s = \{s_{ij}\}_{i,j\in\{0,3\}}$ during this last round are represented in Figure 2.5 with $k = \{k_{ij}\}_{i,j\in\{0,3\}}$ the 16-byte last round key and $c = \{c_{ij}\}_{i,j\in\{0,3\}}$ the 16-byte ciphertext.

In this last round, it is worth noticing that storing the ciphertext in memory does not overlap the same state's bytes of the previous round. As shown in Figure 2.5, the operation ShiftRows modifies the bytes order. But the leakage actually reveals the transition between the last and the final states, whatever the connections between their bytes. Thus, as prediction model, we choose the Hamming distance between each byte of the internal state before the last round and its update in the ciphertext. For each key hypothesis k^*, for each byte i and each ciphertext c,

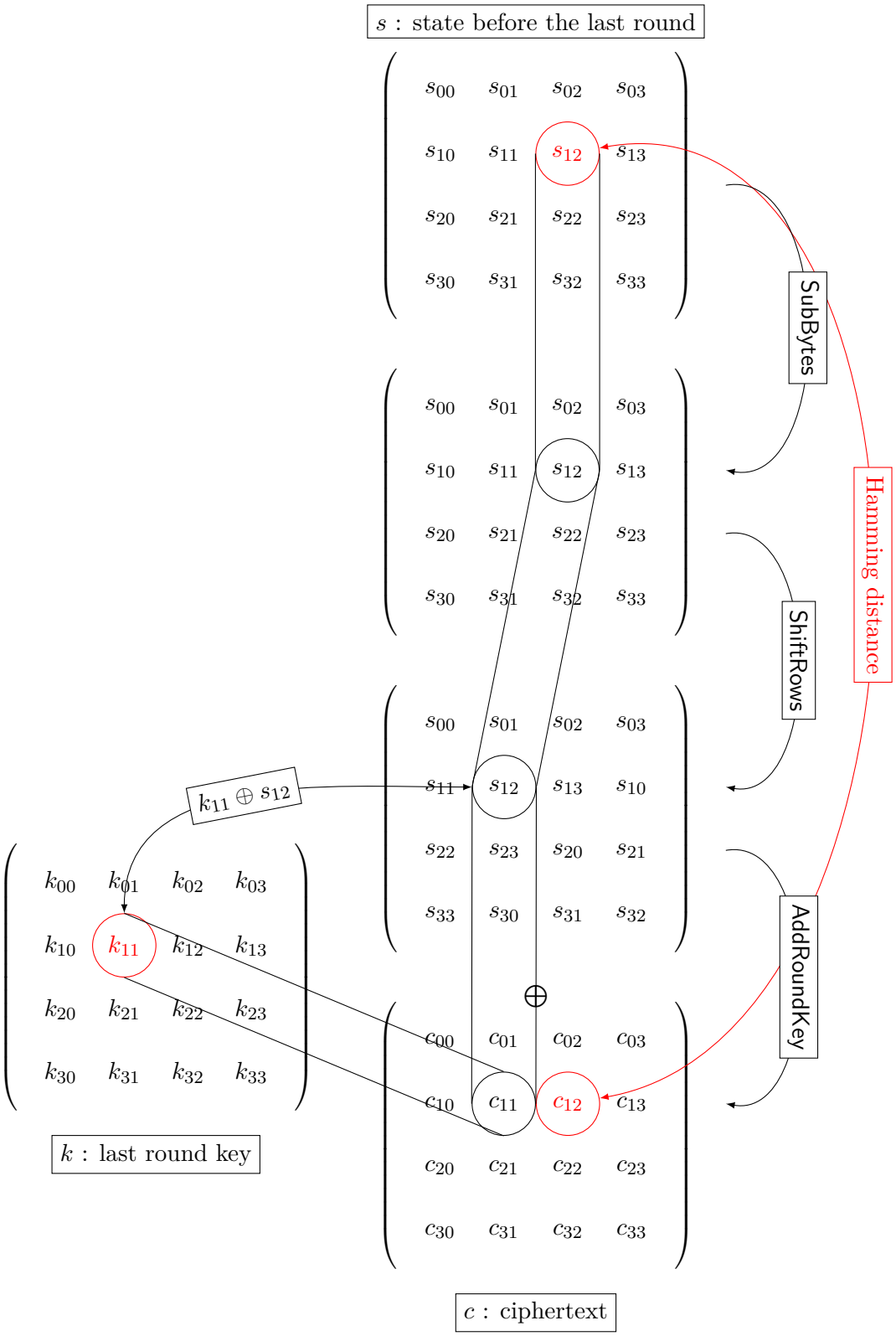


Figure 2.5: AES last round

we compute the following prediction:

$$\mathsf{HD}\left(c_j, \mathsf{InvSubBytes}(c_i \oplus k_i^*)\right)$$

where j represents the position of the byte i before the operation ShiftRows. In the example of Figure 2.5, the prediction for key k_{11}^* and ciphertext c is thus:

$$\mathsf{HD}\left(c_{12}, \mathsf{InvSubBytes}(c_{11} \oplus k_{11}^*)\right).$$

Figure 2.6 represents the correlations for each 256 key hypothesis k_{00}^* corresponding to the first byte of the last round key using 20,000 ciphertexts and consumption traces. The absolute correlation of the hypothesis corresponding to the correct key byte is drawn in red while the absolute correlations of the other 255 hypothesis are drawn in blue. We can see that the good hypothesis actually has a better correlation value in the last round.

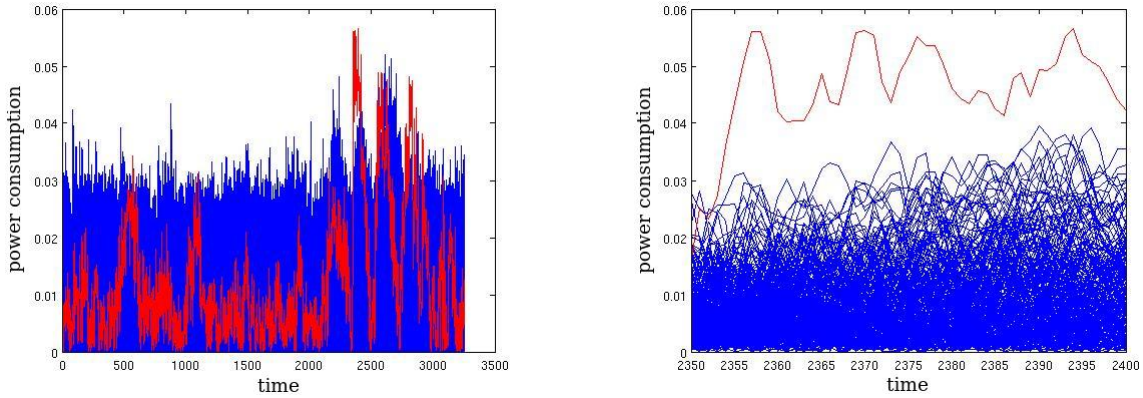


Figure 2.6: Correlations for the 256 last round first key byte hypotheses for the full AES (left) and with a zoom on the last round (right)

Mutual Information Analysis. At CHES 2008, a new distinguisher named Mutual Information Analysis (MIA for short) was introduced by Gierlichs, Batina, Tuyls, and Preneel [GBTP08]. It relies on the mathematical mutual information tool which measures the dependence between two random variables: in our case, the leakage $\mathcal{L}$ at some point of interest and the prediction $\mathcal{P}_{k^\star}$ for each key hypothesis $k^\star$. With continuous leakage and the discrete predictions, the mutual information is defined as:

$$\mathcal{I}(\mathcal{L}, \mathcal{P}) = \sum_{p \in \mathcal{P}} \int_{\mathbb{R}} \mathbb{P}[\mathcal{P} = p, \mathcal{L} = \ell] \log \frac{\mathbb{P}[\mathcal{P} = p, \mathcal{L} = \ell]}{\mathbb{P}[\mathcal{P} = p] \cdot \mathbb{P}[\mathcal{L} = \ell]} d\ell. \tag{2.4}$$

The mutual information between two variables is always greater or equal to zero. It is upper bounded by the lowest Shannon entropy H between both variables if one of them is a deterministic function of the other one and it is equal to zero when both random variables are independent:

$$0 \leqslant \mathcal{I}(\mathcal{L}, \mathcal{P}) \leqslant \min(H(\mathcal{L}), H(\mathcal{P})).$$

Concretely, the mutual information evaluates the level of dependence between two random variables. Ideally, we expect the mutual information to be equal to zero when it applies to the leakage and the predictions of a wrong key hypothesis. However, in practice, there is still some dependencies between the leakage and wrong predictions. Therefore, and as for the CPA, the adversary determines the correct key by selecting the highest distinguisher value.

In order to compute the mutual information between the leakage and the prediction, the adversary first needs to estimate the probability density function of the joint distribution $\mathbb{P}[\mathcal{P} = p, \mathcal{L} = \ell]$ from the available samples. This issue of density estimation is well studied in statistics but the best way to perform it in our context is still unclear [WO11a]. Nevertheless, several proposals have been made and two methods appeared to give reasonable results for power-analysis attacks: the use of histograms and the use of kernels. In both case, the attacker needs to fix some parameters. Among them, the number of bins for histograms and the bandwidth for kernels are perhaps the most important. On the one hand, having more bins or a smaller bandwidth improves the precision of the estimation and allows to capture more dependencies. On the other hand, it requires more consumption traces to be accurate, otherwise it can lead to wrong results. The choice of such parameters is discussed for instance in [Tur93, GBTP08, CTO+14].

At the end, the MIA distinguisher is very powerful since, contrary to the Pearson correlation coefficient, it captures any kind of dependencies and is not restricted to the linear ones. However, probably partly due to the estimation difficulties, it is less effective than CPA when the prediction model fits the reality of leakage such as explained for example in [SGV09,VS09,WO11b,BGP+11, WO11a]. Conversely, it better supports model inaccuracies even if it won't help for a totally wrong prediction model.

2.3 Modeling and Evaluating Cryptographic Implementations

In this section, we discuss the modeling of the leakage of cryptographic implementations. This modeling aims to help designers to define secure implementations and to help evaluators to properly analyze their security. We give a few techniques to describe the latter.

2.3.1 Modeling the Leakage

In order to reason about the security of cryptographic implementations, the community has defined several leakage models which aim to capture the information that is leaked to the adversary. Most of them can be separated into three categories:

1. the leakage is assumed to take the form of an arbitrary leakage function whose entropy or output size is bounded,

2. the leakage is assumed to take the form of a specific function, such as the exact values of a specific number of intermediate variables or the noisy values of all the intermediate variables,

3. the leakage is characterized after a profiling step and it takes a form which completely depends on the inherent considered device.

The first category benefits to model any kind of leakage as long as the information is limited, which seems to be reasonable in practice when the limitation is defined on a single invocation. However, such generic models come at the price of more complex constructions. Indeed, to thwart any kind of attacks which are only limited by the quantity of leaked information, the cryptographic implementations must often be more robust than necessary. Such models, referred to as leakage-resilient cryptography models, will be discussed in Section 2.4.2.

The second category generally lead to less complex cryptographic implementations but cannot thwart all possible power-analysis attacks in all contexts. For instance, the probing leakage

model assumes that the adversary gets back the exact values of a fixed number of intermediate variables. It comes from two observations. First, the storage of a variable in a register is known to generally leak a function of this variable. Nevertheless, in practice, it generally leaks the Hamming distance between the stored variable and the variable which is erased. Since the model only considers the new variable, some attacks might remain. Second, it is difficult to combine several variables when their leakage contains noise. Therefore it makes sense to bound the number of observations the attacker can make. Such models will be discussed in Section 2.4.1 since they particularly fit the masking countermeasure.

The last category gathers the most specific leakage models since they are defined from the characterization of the underlying device. However, such a definition requires the ability to control the device and in particular to inject the desirable data and keys on one of its instances. In practice, it is difficult for an attacker to have such capabilities. However, it is very convenient for evaluators who want to precisely assess the security of a cryptographic implementation. Since this last category is not specific to a countermeasure, we give further details hereafter while discussing evaluation methods.

2.3.2 Evaluating the Security

Whereas it is difficult to protect cryptographic implementations against all the existing attacks, it is also difficult to evaluate the global security of a given cryptographic implementation. Therefore, some cryptographers have investigated generic techniques, defined frameworks and designed tools to help the evaluators and the designers in their task.

An example of framework is the one proposed by Standaert, Malkin, and Yung at Eurocrypt 2009 [SMY09] to analyze side-channel key recovery attacks. It underlines the advantage of estimating leakage parameters through a profiling phase to perform a better evaluation and provides theoretical tools to deduce success rates of common side-channel attacks. In the following, we discuss these two aspects. We first explain the principle of template attacks which allows to properly analyze physical information leakages. Then, we present a tool, designed by Veyrat-Charvillon, Gérard, and Standaert in [VGS13] which allows the estimation of the framework's metrics. In particular, it provides a very useful method to estimate the rank of a large size key (*e.g.,* 128 bits) from the probabilities of small key chunks hypothesis.

Template Attacks. Template attacks, introduced by Chari, Rao, and Rohatgi [CRR03], are the most powerful power-analysis attacks in an information theoretic sense. They can be divided in two phases: a leakage characterization and then an attack with this collected information.

In practice, the leakage does not exactly (or not at all) verify the classical noisy Hamming weight model presented in Equation (2.2). However, it is usually valid to approximate the distribution of the power consumption involved by the manipulation of a single uniformly distributed variable by a normal distribution [MOP07]. Such a normal distribution is fully determined by a mean μ and a variance σ^2. At a larger scale, the distribution of power traces can be approximated by multivariate normal distributions, fully characterized by a mean vector $\boldsymbol{\mu}$ and a covariance matrix Σ. In template attacks, we assume that the attacker can manipulate the secret key to her choosing (as it is usually the case of an evaluator) and thus compute the pairs $(\boldsymbol{\mu}, \Sigma)_{x_i,k_i}$ for each pair of data and key (x_i, k_i).

Once the power traces have been characterized, the attacker can use this knowledge to recover the secret key. Given a consumption trace $\boldsymbol{\ell}$, the attacker can compute the probability density functions (pdf) of $\boldsymbol{\ell}$ and every multivariate normal distribution. The best probability should

design the correct data x_i and k_j[1]. Multiplying the resulting probabilities for the different plaintexts x_i improves the key selection.

Security Graphs. Using the aforementioned techniques to perform DPA, the attacker usually gets a list of probabilities for each 2^b key hypothesis (when the key chunk is of bit size b). For instance, using the Pearson correlation coefficient, we obtain a correlation value for each key hypothesis. These correlation values can easily be turned into probabilities by dividing each of them by their global sum. The highest probability indicates the most probable key chunk hypothesis and it is easy to evaluate the rank of the correct key chunk by identifying its position in the probabilities' list. However, an attacker is more interested in the rank of the n-bit key among the 2^n possible keys (*e.g.,* 2^{128} in the AES) than in the n/b key chunks' ranks. Therefore, Veyrat-Charvillon, Gérard, and Standaert built in [VGS13] an efficient algorithm to estimate such a rank from the n/b lists of probabilities. Using this algorithm, they provide a method to draw *security graphs* in order to illustrate the evolution of an attack with the desired metric (for instance, the number of measurements). An example of such a graph is given is Figure 2.7. It draws the rank evolution of the correct 128-bit secret key when simulating template attacks based on Equation (2.1) on an unprotected AES Sbox in software with randomly generated known inputs. The black (resp. white) line indicates the minimum (resp. maximum) rank (in logarithm scale) of the correct secret key and the red line indicates the mean rank of this key on several experiments.

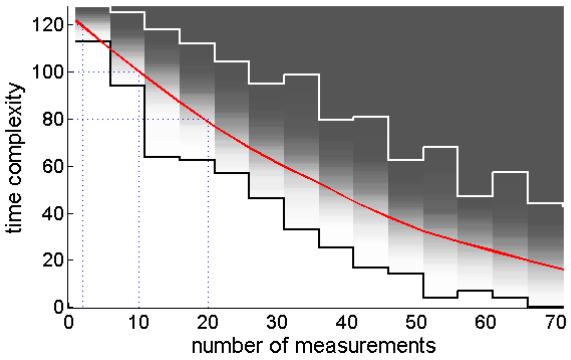


Figure 2.7: DPA-based security graphs for an unprotected AES-128 with known inputs

2.4 Countermeasures

In order to thwart DPA, the community has made a great effort to find dedicated and generic countermeasures. Some of them are actually completely independent from the executed algorithm. For instance, in the case of circuits, the wires transporting the information bits can be doubled to spread, in addition to the original signal, its complementary. As a consequence, the sum of the power consumption is always (in theory) constant and thus independent from the manipulated data. This countermeasure is referred as *dual-rail* and is used to protect cryptosystems such as explained in [TV03, PM05, CZ06]. Conversely, the designers can add some noise to noticeably remove the correlation between the power consumption and the manipulated data. However, the real impact of these countermeasures is very hard to quantify and is dependent on

[1]If the attacker knows the plaintext x_i, she can just compute the pdf of the power trace and the multivariate distributions with x_i.

the inherent device. Therefore, many works rather focus on algorithmic countermeasures, less dependent on the device. Among these countermeasures, *masking* and the use of *leakage-resilient primitives* are the most studied ones.

2.4.1 Masking

Masking is probably the most deployed side-channel countermeasure. Simultaneously introduced by Goubin and Patarin [GP99] and Chari *et al.* [CJRR99], it consists in randomizing the sensitive data (which depend both on a secret constant and on known variables). To do so, each sensitive variable x is divided into $t+1$ variables $\{x_i\}_{0 \leqslant i \leqslant t}$ referred to as *masks* or *shares*. Among these masks, t are generated uniformly at random and the last one is computed such that the combination of the $t+1$ masks, according to a chosen law $\star$, is equal to the initial variable x, *i.e.,* $x = x_0 \star ... \star x_t$. The sensitive variable is not used anymore and the computation is thus only made using the masks as independently as possible. Note that the choice of the masking operation $\star$ depends on the algorithm. For instance, in the AES, most operations are Boolean. Thus, the use of a *Boolean masking*, *i.e.,* $\star = \oplus$ (in $\mathsf{GF}(2^8)$), only multiplies the complexity of the linear operations by a factor $t+1$. However, to compute the Sbox, either we can conserve the Boolean masking (complexity in $O(t^2)$) or we can change for a *multiplicative masking*, *i.e.,* $\star = \times$ (in $\mathsf{GF}(2^8)$). However, in this last case, the conversion is expensive which limits the gain acquired during the multiplication. Another widely used example is the *additive masking* (modular addition) which can be used to protect hash functions SHA-1 and SHA-2 for instance.

If the inherent device leaks information on one manipulated variable at a time, a *t^{th}-order masking*, *i.e.,* with $t+1$ shares, generally resists to the *t^{th}-order attacks* which combine the information of at most t points in the consumption traces. Note that as soon as the adversary acquires the consumption traces, she is completely able to exploit every single power consumption value she gets (and not only t). The masking order rather reveals the difficulty to combine t points which jointly depends on the manipulated data (generally through a one-way function and a sufficient amount of noise). In particular, Chari, Jutla, Rao, and Rohatgi [CJRR99] have shown that the number of measurements required to mount a successful DPA increases exponentially with the number of shares in the very classical and generic context where each manipulated bit is leaking the sum of its value and a Gaussian noise $\mathcal{N}(\mu, \sigma^2)$. If an attacker is provided with the leakage of the $t+1$ shares of a single bit, then they have shown that the number of measurements m required by the adversary to distinguish between the two possible bit values with probability p is lower bounded as follows:

$$m \geqslant \sigma^{t+\delta},$$

with $\delta = 4\log(p)/\log(\sigma)$. Nevertheless, as mentioned above in the context of Boolean masking, the complexity of masked implementations also increases with the masking order.

As a trade-off between security and efficiency, nowadays, most masking scheme implementations are of first order (*i.e.,* with two shares) or second order (*i.e.,* with three shares). Mounting an attack against a first-order masking scheme is already a difficult task. Indeed, the attacker has to adapt the cryptanalysis techniques (prediction model, distinguisher) to the combination of two leakage values which jointly depend on the secret. In practice, either she can evaluate the mutual information between the two shares and the prediction or she can combine the leakage of both shares and use a classical distinguisher (*e.g.,* Pearson correlation coefficient) between this result and the prediction. For this purpose, two main functions have been proposed. The first one was given by Chari *et al.* in [CJRR99] and consists in multiplying the two leakage values which jointly depends on the secret. The second was introduced by Messerges in [Mes00] who

proposed to compute the absolute difference between the two related leakage values. In spite of the introduction of these two combining functions, the research is still open to determine their efficiency in different scenarios and eventually to exhibit more relevant functions.

As mentioned above, the complexity of attacking a masking scheme grows exponentially with its masking order [CJRR99]. Yet, evaluating the security of a higher-order masking implementation may be very costly for an evaluator, particularly if one needs to compute the corresponding success rates (which require several applications of the same attacks). Therefore, huge efforts have been made to approximate the success rates of first-order and higher-order attacks. Among the works on first-order attacks, Rivain proposed in [Riv09] a formula to evaluate the success rate of first-order attacks with additive distinguishers, *i.e.,* distinguishers for which the score of each key hypothesis can be completed with new leakage values (*e.g.,* Pearson correlation coefficient) considering the joint distribution of all the key scores. Recently, this work was extended by Lomné, Prouff, Rivain, Roche, and Thillard [LPR+14] to evaluate the success rates of higher-order attacks. Their methodology requires an estimation of the leakage parameters (see for instance [CRR03, APSQ06]) and the estimation of a score vector distribution using this leakage. The success rates can then be directly computed from the comparison between the score of the correct secret key and the score of each wrong hypothesis. In order to validate their estimation, the authors realized practical attacks and showed that experimental success rates appeared to be very close to their approximation.

Adversary Models

The first step towards formally reasoning about the security of masked algorithms is to define a leakage model that captures the information that is leaked to the adversary. For this purpose, Chari, Jutla, Rao, and Rohatgi [CJRR99] introduced the *noisy leakage model*. In this model, the adversary gets leaked values sampled according to a Gaussian distribution centered around the actual value of the wire. The authors showed that, under this model, the number of queries required to recover a sensitive bit was at least exponential in the masking order. The noisy leakage model was later extended by Prouff and Rivain [PR13] in several respects. First, the authors considered more general distributions to sample noisy leakage, rather than just Gaussian [CJRR99] or Bernoulli [FRR+10] leakage distributions. Moreover, they removed the limitation to one-bit observations, allowing the adversary to observe intermediate variables of any bit size. Finally, they also extended the notion of leakage to take computation, rather than data, into account, following the *only computation leaks information* principle introduced by Micali and Reyzin [MR04]. They also offered the first proof of security for a masked algorithm in this model, although it relies on leak-free components and a relatively weak adversary model.

In a different approach, Ishai, Sahai, and Wagner [ISW03] introduced in 2003 the *t-threshold probing model*, in which the adversary receives the exact value of *at most t internal variables* (of her choice) in the computation. Moreover, they also described a transformation that turns any boolean circuit secure in the *black-box model* into a functionally equivalent circuit secure in the t-threshold probing model.

In practice, the noisy leakage model is often thought of as more realistic, since experimental physical leakage is noisy [MOP07]. However, the t-threshold probing model is much more convenient to build security proofs since it considers exact values. Fortunately, the relationship between these two models was recently clarified by Duc, Dziembowski, and Faust [DDF14]. The authors advantageously recast the noisy leakage in the more classical statistical security model and showed that security in the extended noisy leakage model of [PR13] can be reduced to security in the t-threshold probing model of [ISW03], in which security proofs are much

more direct. In addition, the reduction did not rely on the existence of leak-free components. To proceed, Duc, Dziembowski, and Faust introduced a leakage model, similar to a model introduced by Ishai, Sahai, and Wagner [ISW03], called *random probing model*, in which the adversary obtains an intermediate value with probability ε. Using this new model, the authors proved the reduction in two steps. First, they showed that for any adversary in the δ-noisy leakage model on the set $\mathcal{X}^\ell$ of intermediate values, where δ represents the statistical distance between the distribution of the intermediate variables and the distribution of these variables given their leakage, there exists an adversary in the $\delta|\mathcal{X}|$-random probing model on $\mathcal{X}^\ell$. They also showed that this first simulation was perfect as long as $\delta < 1/|\mathcal{X}|$. Then, the authors demonstrated that, for any adversary in the $\delta|\mathcal{X}|$-random probing on $\mathcal{X}^\ell$ model, there exists an adversary in the $(2\delta|\mathcal{X}|\ell-1)$-threshold probing model of Ishai, Sahai, and Wagner. Thus, proving the security of a cryptosystem in the $(2\delta|\mathcal{X}|\ell-1)$-threshold probing model automatically ensures its security in the more realistic δ-noisy leakage model. Unfortunately, the second reduction is not tight. Consequently, proving the security of an implementation in the δ-noisy leakage model for ℓ intermediate values on $\mathcal{X}$ can be achieved by proving the security of an implementation in the $(2\delta|\mathcal{X}|\ell - 1)$-threshold probing model, that is when the attacker can observe $(2\delta|\mathcal{X}|\ell - 1)$ exact intermediate values. While this might still be reasonable for bits (*i.e.,* $|\mathcal{X}| = 2$), it becomes prohibitive when working on bytes (*i.e.,* $|\mathcal{X}| = 256$).

The reduction between these models led to prohibitive bounds on the success rates (or attacker advantages) according to the noise, the size of the definition set $\mathcal{X}$, and the number of measurements. Concretely, these large bounds result in prohibitive number of shares or amount of noise to achieve a reasonable security level. However, such theoretical bounds on the success rates are not always tight due to potential proof artifacts. In order to evaluate these biases, Duc, Faust, and Standaert proposed a comparison with concrete attacks [DFS15a]. In particular, they noticed from their experiments that the factor $|\mathcal{X}|$, as well as other constant factors, was mostly a proof artifact in the evaluation of the number of measurements to achieve a given success rate.

In order to improve the tightness of such bounds in theory, Dziembowski, Faust, and Skorski proposed a second reduction at Eurocrypt 2015 [DFS15b]. Instead of the t-threshold probing model, they showed a tight equivalence between the noisy leakage model and the newly defined *average probing model*, which is slightly different from the random probing model. Concretely, the security in the δ-noisy leakage model involves the security in the 2δ-average probing model while the security in the δ-average probing model involves the security in the 2δ-noisy leakage model. Although the bounds are much more advantageous, the authors once again require the use of leak-free gates to make security proofs of masking schemes. Figure 2.8 illustrates the relations and the properties of the three aforementioned models that we can use to make security proofs.

In this aforementioned models, only the values of fixed-size sets of intermediate variables are usually considered when determining the security order of an implementation. However, as shown by Balasch *et al.* in [BGG+14], this value-based leakage model does not fully capture some real-world scenarios in which additional physical leaks can occur, such as leakage due to transitions or glitches, where more than one intermediate variable is involved in each single observation. The transition-based model well fits the reality of software where it is common to observe the leakage of two variables during a register storage: the former variable and the new stored one. Thus, either the registers must be smartly used or the number of shares should be multiplied by two to avoid a security breach. With similar drawbacks, leakage on glitches is dedicated to hardware. Glitches are unwanted transitions at the output of logical gates resulting from differences in signals arrival time. The number of such transitions at a gate's output can

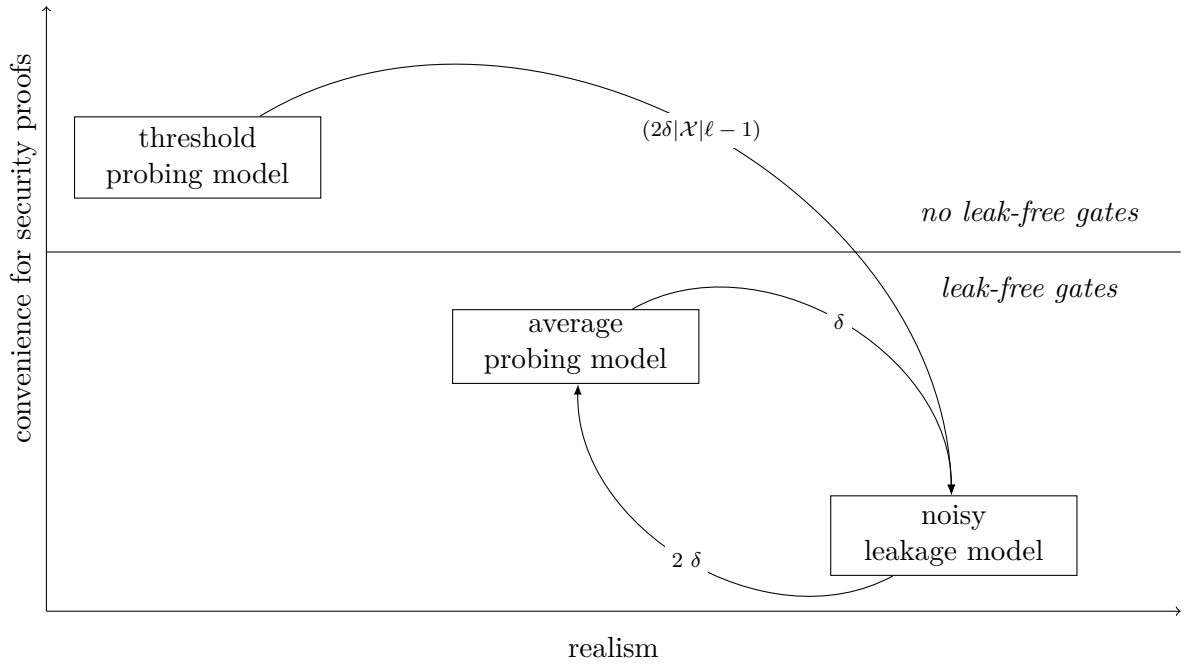


Figure 2.8: Realism and proof convenience of three leakage models

be detected by power analysis and may reveal information on the secret key. An attacker can thus mount a first-order attack in higher-order masking schemes. In this case, an algorithmic countermeasure is the use of so called *threshold implementations*. More than only masking the sensitive values, the number of shares is increased to ensure that the leaks due to glitches do not result in security flaws. Finally, a perfectly masked algorithm secure in the value-based model can succumb to first-order attacks in these finer-grained leakage models. Therefore, the t-threshold probing model is a relevant tool to evaluate the security of masked implementations but the number and the characteristics of the variables involved in each single observation should be well defined beforehand according to the inherent device.

Security Proofs in the t-Threshold Probing Model

In the t-threshold probing model, the adversary gets access to at most $t \times x$ intermediate variables where x represents the maximal number of intermediate variables leaking at a time (*e.g.,* $x = 1$ in the value-based model and $x = 2$ in the transition-based model). As far as we know, there is currently no generic proof for x equal to 2 or more since the leakage completely depends on the implementations (*e.g.,* the use of registers) and on the underlying device. Thus, we focus here on the value-based model, when $x = 1$. There are generally two steps to build the security proof of an higher-order masking scheme: (i) proving the security of each small masked algorithm (*e.g.,* multiplications, linear functions) and (ii) proving that their composition is still secure.

A first step is to prove the security of (small) masking functions in the value-based t-threshold probing model, Rivain and Prouff suggested in [RP10] to prove their *perfect simulation* property[2], that is, to show that any set of at most t intermediate variables is jointly independent from the secret. This property is sufficient and Rivain and Prouff used it to prove the security of their

[2]In the following, we will rename this property *simulatability*.

multiplication SecMult in [RP10]. Concretely, they showed that any set of t observations on the multiplication's intermediate variables can be simulated by at most t shares of each input. In 2014, Coron, Prouff, Rivain, and Roche followed the same approach to prove the security of their multiplication between two linearly dependent inputs in [CPRR14]. Although this method is perfectly valid, the complexity of the resulting security proofs make them difficult to verify and to completely trust. For instance, in the former proof, Rivain and Prouff missed a case which cannot be easily patched and in the latter, Coron *et al.* missed a group of intermediate variables which can fortunately be easily restored in their current proof. These examples illustrate the difficulty to cover all the cases. More generally, it is hard to verify the correctness of pen-and-paper reasonings. This demonstrates the need for automatic verification.

As a second step, we focus on composition issues. While the simulation approach is perfectly valid to prove the security of a masked algorithm in the t-threshold model (when $x = 1$), it faces limitations when the algorithm is too large. Indeed, the proof relies on the enumeration of all the possible fixed-size sets of intermediate variables. If there are too many variables with too many dependencies, the usual pen-and-paper proof quickly becomes unpractical. For instance, while the proof perfectly fits for the masked multiplication of Rivain and Prouff [RP10], it becomes too complicated to show the security of their whole AES Sbox. To thwart this issue, the community tried to find a method to securely compose secure small masked algorithms. However, either these methods do not come with a security proof or they face efficiency issues. For instance, the Sbox defined by Rivain and Prouff as the composition of secure blocks (in the sense of simulatability) failed to achieve the security against t^{th}-order attacks. Yet, the authors noticed the need of refreshing the masks that are reused in different masked algorithms. Thus, in their sequence of operations,

$$\begin{aligned}(z_0, z_1, \ldots, z_t) &\leftarrow (x_0, x_1, \ldots, x_t)^2 \\ (z_0, z_1, \ldots, z_t) &\leftarrow \mathsf{RefreshMasks}((z_0, z_1, \ldots, z_t)) \\ (y_0, y_1, \ldots, y_t) &\leftarrow \mathsf{SecMult}((z_0, z_1, \ldots, z_t), (x_0, x_1, \ldots, x_t)),\end{aligned}$$

even if both inputs of the multiplication SecMult depend on the shares of x, one of them has been refreshed using the algorithm RefreshMasks. Unfortunately, with the observations of one intermediate variable in RefreshMasks and $t/2$ intermediate variables in SecMult, Coron *et al.* [CPRR14] exhibited an attack. This example illustrates the complexity of securely composing blocks. In a different approach, Prouff and Rivain in [PR13], as well as Duc, Dziembowsky, and Faust in [DDF14], proposed to refresh the masks between each small masked function. While this method may be valid for an attacker which may combine t intermediate variables per function, it could be improved, in terms of efficiency, for an attacker able to combine t intermediate variables in the whole execution.

Formal Methods

Many tools designed to prove the security of masked algorithms in the t-threshold probing model have recently appeared (*e.g.,* [MOPT12, BRNI13, EWS14, EW14, EWTS14]). We describe here a few of them which represent the different main approaches.

Moss, Oswald, Page, and Tunstall initiated the work on automated verification (and generation) of masked programs. In [MOPT12], they developed an automated method to build a (value-based) first-order secure program from its insecure version. Their tool follows an information-flow type-based approach which categorizes the intermediate values in public values, secret values, and uniformly distributed values. While this method benefits from handling large programs extremely fast, it is dedicated to single values. It could be extended to verify

the security of t^{th}-order programs by concatenating sets of t values, but it would require a more elaborate type system and the verification would be completely unpractical already for programs of reasonable size.

In a slightly different context, Pettai and Laud [PL14] use a type-system to prove security of masked programs with a limited number of adversary observations imposed by their adversary model in the multi-party computation scenario. They do so by propagating information regarding linear dependencies on random variables throughout their arithmetic circuits and progressively replacing sub-circuits with random gates. Although their technique is limited to a few adversarial observations, they can deal with active adversaries which may be useful when considering active adversaries.

Bayrak, Regazzoni, Novo, and Ienne [BRNI13] also developed a type-based approach but their solution is based on an SMT-based method[3] for analyzing the sensitivity of sequences of operations. Informally, the notion of sensitivity characterizes whether a variable used to store an intermediate computation in the sequence of operations depends on a secret and is statistically independent from random variables. Their approach is specialized to first-order masking schemes, and suffers from some scalability issues: in particular, they only report analysis of a single round of AES.

Eldib, Wang, and Schaumont developed an alternative tool, SCSniffer [EWS14], that is able to analyze masked implementations of orders 1 and 2. Their approach is based on model counting [GSS09]: to prove that a set of probabilistic expressions is distributed independently from a set of secrets, model-counting-based tools count the number of valuations of the secrets that yield each possible value of the observed expressions and check that that number is indeed independent from the secret. This process in itself is inherently exponential in the size of the observed expressions, even when only one such observation is considered. To overcome this issue, SCSniffer implements an incremental approach for reducing the size of such expressions when they contain randomness that is syntactically independent from the rest of the program. This incremental approach is essential to analyzing some of their examples, but it is still insufficient for analyzing complete implementations. For instance, SCSniffer can only analyze one round of (MAC-)Keccak.

Finally, another approach based on program verification and equivalence of programs was introduced by Barthe, Grégoire, and Zanella-Béguelin [BGZB09] under the name pRHL (for probabilistic Relational Hoare Logic). To prove the independence of a set of observations from the secret, it shows that the joint distribution of this set coincides on any two executions (with different inputs). However, this method misses a pre-condition to directly deal with distributions of memories instead of memories themselves.

Compilers

In order to provide higher-order masked circuits secure in a given model, the cryptographic community has started to develop theoretical circuits compilers[4]. Such tools take a cryptographic scheme C in input together with a secret key k and returns a functionally equivalent circuit C' with a secret key k'. The resulting circuit C' instantiated with k' benefits from being additionally resistant to a certain class of side-channel attacks.

A pioneer work on the subject was provided in 2003 by Ishai, Sahai, and Wagner [ISW03] who designed a compiler (further referred to as ISW compiler) to transform any circuit made

[3]The SMT (Satisfiability Modulo Theories) problem is a decisional problem for logic formulas.

[4]The several examples we point out in this paragraph all refer to circuit compilers but note that the same transformation can be built for software implementations.

of N logical gates into an equivalent one of size $O(Nt^2)$ which is additionally secure in the $\lfloor (t-1)/2 \rfloor$-threshold probing model, where the attacker can read the exact values of at most $\lfloor (t-1)/2 \rfloor$ wires in the circuit. In 2010 [RP10], Rivain and Prouff observed that the proof was valid for any finite field and not only $\mathsf{GF}(2)$. In parallel, Faust *et al.* [FRR+10] extended the ISW compiler to two different and more realistic leakage models but requiring a leak-free hardware component. The first model assumes that the attacker receives at each cycle a leakage function of the circuit state which is in the AC^0 complexity class. The second model assumes that the attacker receives the exact value of each circuit bit with a fixed probability p (< 0.5). In 2012, Rothblum [Rot12] removed the need for the leak-free hardware component in the context of the first leakage model.

In order to fit better the reality of embedded devices, as mentioned above, Prouff and Rivain introduced the noisy leakage model in [PR13] and showed how to prove the security of a block cipher. Whereas this proof requires strong assumptions, Duc, Dziembowski, and Faust provided in [DDF14] a reduction from this realistic noisy leakage model to the convenient t-threshold probing model. From this result, they built a secure compiler that transforms any cryptographic scheme secure in the black-box model into one secure against the noisy leakage model by refreshing the masks between each operation. This method is sound if the attacker is able to observe t exact values in each operation but it is inefficient for an attacker which observes t exact values per algorithm invocation. While this compiler benefits from generating circuits secure in a realistic model with convenient proofs, it suffers from the large bound used to make the reduction between both leakage models. Therefore, Duc, Faust, and Standaert developed in [DFS15a] a new compiler secure in the average probing model which benefits from being equivalent to the noisy leakage model. Unfortunately, the authors need once again the use of a leak-free hardware component. In a different approach, Andrychowicz *et al.* [ADD+15] proposed a compiler which minimizes the overhead (transformed circuit of size $O(Nt)$ instead of $O(Nt^2)$) but in leakage models that are less realistic than the noisy leakage one.

2.4.2 Leakage-Resilient Cryptography

Two main limitations of masking are that it can be defeated by higher-order attacks [Mes00] and it generally induces strong performance overheads. Another line of work, introduced by Dziembowski and Pietrzak [DP08] under the name *leakage-resilient cryptography*, follows a complementary approach and tries to limit the useful information per invocation. For this purpose, the main assumption made by the authors is that the information leakage per iteration is limited in some sense. Of course, in practice, the attacker usually receives large quantity of information per iteration, but this limitation aims to represent the useful information among the whole leakage. In the following, we start with a description of the leakage-resilient cryptography model. Then, we recall a classical security model based on indistinguishability which can be adapted to prove the security of functions in presence of leakage. Afterwards, we describe the principle of fresh re-keying, introduced by Kocher [Koc03]. When applied in the context of symmetric cryptography, it is well deployed for the construction of leakage-resilient algorithms. Eventually, we describe two typical instances: a pseudo-random function and a pseudo-random number generator.

Leakage Function

In leakage-resilient cryptography, we also need to model the leakage and thus make further assumptions about it.

First of all, as it is the case in practice, we cannot give all the information to the attacker, thus we need to bound the useful leakage at some point. In the context of masking, the experiments

showed that combining variables that jointly depend on a sensitive value was increasingly hard with the number of variables. Thus and thanks to the reduction made by Duc, Dziembowsky, and Faust in [DDF14], a common assumption is that the attacker receives only a fixed number of intermediate variables exact values. For leakage-resilient primitives, there is no masking thus no recombination issue. Therefore, the usual assumption is to bound the overall leakage independently of the number of observed values. Concretely, the adversary receives a certain amount of bits (further denoted by λ), which can be interpreted as the mutual information between the leakage and the secrets. This bound can be applied either to each execution of the algorithm or to the whole protocol with several executions. In the following, we make the first assumption which fits better the reality of power analysis. We thus model the leakage using a function f for each invocation of the targeted function and whose output belongs to $\{0,1\}^\lambda$. As observed by Micali and Reyzin [MR04], such a leakage function realistically takes three input parameters: the current internal configuration C of the underlying device, a measurement parameter M revealing the choices of the attacker and a random string R to express the randomness part of the measurement. In the following and as justified in [SPY$^+$10], we will omit R whose impact is expressed by the bounded output and also M which does not help in side-channel analysis. In C, we will only keep the parameters which have an impact for an attacker, namely the secret key of size n and the inputs of size m:

$$f : \{0,1\}^n \times \{0,1\}^m \quad \rightarrow \quad \{0,1\}^\lambda. \tag{2.5}$$

Following the well-known assumption of Micali and Reyzin [MR04] according to which *only computation leaks information*, we assume that only the data manipulated at a time can leak information at this same time. The assumption does not hold for all the components in practice but generally fits well the reality of embedded devices. We thus consider that the input parameters of f are actually manipulated in the corresponding invocation.

Now that the leakage function's shape has been defined, we can discuss its complexity. Usually, we use to restrict in some way the set of leakage functions. Indeed, such a function f should represent the leakage of the device. If an AES is implemented and executed then the adversary can obviously observe $f(k,p) = [\mathsf{AES}(k,p)]_\lambda$ with the key k and the plaintext p and $[]_\lambda$ the truncation of the first λ bits. However, if the function being executed is an addition between the key and the plaintext, the previous leakage function does not make any sense. Concretely, the leakage function can target any intermediate result of the function being executed but it should represent the device leakage so it is not expected to make further computations on it. This restriction should avoid some unrealistic attacks which try to predict future executions rather than using the current leakage. However, the difficulty to define realistic functions shows the limitations of this leakage model.

Regarding its adaptivity, we can leave the attacker the possibility to modify the leakage function between two executions of the algorithm with the knowledge of its first observations. However, we think that each leakage function completely depends on the inherent device and the function being executed and thus cannot be modified between the different executions. This property is further referred to as *non-adaptive leakage-resilience* (naLR for short) of the leakage function. Another close relaxation is the notion of *non-adaptive function* introduced in [FPS12] for the pseudo-random functions (PRF for short) and denoted by na. It refers to a context where the adversary is expected to choose her input queries in advance, that is before seeing any leakage or output. Concerning the granularity, as suggested in [FPS12], we consider the global cryptographic algorithm as a combination of smaller blocks that leak independently. These blocks can be either different functions or several invocations of the same function, following the works [DP10, SPY$^+$10, FPS12]. Let us denote by τ_i the state before step i. In the following, we

assume that the adversary can choose a leakage function f_i for each delimited block i and gets $f_i(\tau_i)$ at the end of the step execution.

Security Models

A common expectation for cryptographic functions is a form of indistinguishability with truly random referred to as the *real-or-random* indistinguishability. For instance, a pseudo-random function is assumed to be secure if an attacker cannot distinguish with a significant advantage the output of the PRF for some input from the output of a uniformly random function for the same input. In a different level and as defined in [BDJR97], an encryption scheme is assumed to be secure if an attacker cannot distinguish with a significant advantage the encrypted query from the encryption of a random value.

However, when the attacker is given some leakage on the function, indistinguishability is much more difficult to achieve. It is not a decision on the outputs based on the knowledge of the inputs and the algorithm but it also involves information on intermediate values. With the leakage function described above, a strategic way to distinguish between the real output and the random one is to choose f as the first λ output bits. In this case, the attacker's advantage is significant even for a small λ since there is a few chances for the random output to share the same λ real one's bits. Thus, it is mandatory to define a different form of indistinguishability to prove the security of leaking functions.

An interesting notion of *physical indistinguishability* has been defined in [SPY$^+$10] for this purpose. The main idea is to give an adversary a certain number of queries with the corresponding leakage and to ask her to distinguish the next one. The notion we will use is slightly different. We give the adversary two oracles: a challenge oracle and a leakage oracle. She can submit a fixed number of distinct queries to her oracles which have to be divided between challenge queries and leakage queries. For each leakage query, she gets both the real output of her query and the leakage which is exactly the output of the leakage function f she chose. For each challenge query, she gets, in the case of an encryption scheme, either the real output or the encryption of a random input.

Note that a very recent work proposed by Pereira, Standaert, and Venkatesh [Ven15] proposes an alternative approach to model the security in presence of leakage. To avoid constraining the leakage in challenge queries, they rather aim to prove the security of leakage-resilient schemes by reducing the security of several queries to the security of a single one.

Construction of Leakage-Resilient Primitives via Re-keying

The main method to build leakage-resilient symmetric primitives gets use of the fresh re-keying technique. Contrary to masking, it does not modify the algorithm but the protocol. In DPA attacks, the attacker exploits the reuse of one single secret key processed with different known messages. With this information, she can set up a statistical treatment to recover the key. The idea of the fresh re-keying is to modify the secret key between each execution (or a few executions) of the algorithm. To do so, a function referred to as *re-keying function* and denoted by $\mathcal{R}$ is called between each execution to generate a new key $k^\star$ referred to as a *session key* generally from the master key k and a public varying nonce r. With this new function, the cryptographic algorithm denoted by $\mathcal{A}$ is just expected to be protected against Simple Power Analysis which is much easier and much less expensive. The security against DPA attacks is moved to the re-keying function which, with weaker mathematical expectations, is easier to protect. The protocol is illustrated in Figure 2.9.

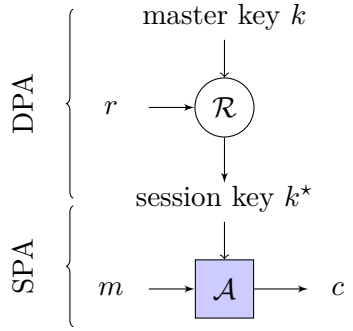


Figure 2.9: Fresh re-keying protocol

We describe hereafter two examples of leakage-resilient primitives: a pseudo-random generator (PRG) and a pseudo-random function (PRF). The former is used in several leakage-resilient constructions which require uniformly random inputs without security loss. The latter is an interesting example of leakage-resilient primitive which exploits the parallelism of hardware operations to improve its security against power-analysis attacks.

Leakage-Resilient Pseudo-Random Generator of Yu and Standaert. Good examples of leakage-resilient functions are the leakage-resilient PRGs. Several such algorithms have already been published [PSP+08, Pie09, YS13]. They perfectly illustrate the aforementioned models and can be used to build other leakage-resilient algorithms which require pseudo-randomness.

We describe here an efficient example which was designed by Yu and Standaert in [YS13] and which benefits from using a little randomness only. It is illustrated in Figure 2.10 and works as follows. Let $\mathbf{G} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and $\mathbf{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{2n}$ be two PRFs, with n the key size. Let $k_0 \in \{0,1\}^n$ be the initial *secret* key and s be the *public* seed, both randomly chosen. The PRF $\mathbf{G}$ is processed in counter mode (input incremented by 1 at each invocation) with a public initial counter to generate the public values $\{p_0, p_1, \ldots\}$. The second PRF $\mathbf{F}$ takes as inputs, for its i^{th} invocation, the public value p_i and the secret key k_i. It returns both a new secret key k_{i+1} and an output x_i.

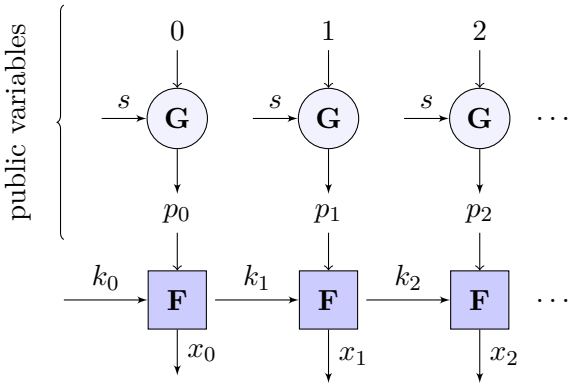


Figure 2.10: Pseudo-Random Number Generator from [YS13]

Yu and Standaert, in addition to introduce this generator, proved that it is leakage-resilient in the model previously defined with only n secret bits and n public bits, both randomly generated.

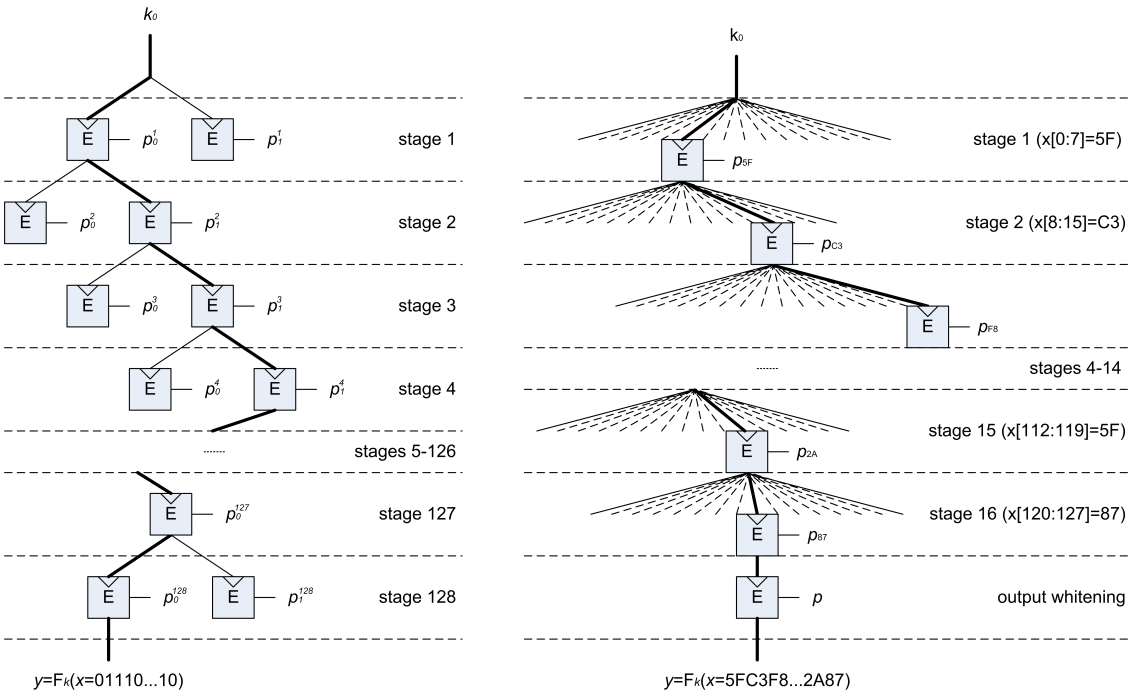


Figure 2.11: Leakage-resilient PRF GGM (left) and MSJ (right) from [MSJ12].

Leakage-Resilient Pseudo-Random Function of Medwed, Standaert and Joux. Most of the first proposed leakage-resilient constructions appeared to be completely prohibitive in terms of performances. In order to thwart this issue, Medwed, Standaert, and Joux introduced a new PRF [MSJ12] which maintains a good security level while making an important step forward to the practice. To achieve their goals, they started with the leakage-resilient construction of GGM [GGM84] (for its authors Goldreich, Goldwasser, and Micali). The latter uses a binary tree structure, as represented on the left of Figure 2.11 and generates 128-bit outputs from 128 calls to an intermediate (unprotected) PRF E. Each bit of the input determines the path in the binary tree. This construction benefits from limiting each key use to the encryption of two different plaintexts whereas the number of measurements is unbounded. In order to improve its efficiency, Medwed, Standaert, and Joux suggested to increase the number of plaintexts by stage to 256 instead of 2. Furthermore, they proposed to use the same set of 256 plaintexts in each stage. As a result, 128-bit outputs can be generated with only 16 PRF calls but each secret key can be used with 256 different plaintexts. The new construction, that we further refer as MSJ, is illustrated on the right of Figure 2.11.

Unfortunately, the increase in the number of encryptions with the same secret key significantly reduces the side-channel resistance of the PRF. Therefore, the authors proposed to exploit the parallelism of hardware implementations to redress the latter's security. Basically, in the hardware implementation of AES for instance, the parallel execution of the Sboxes adds an advantageous algorithmic noise to the leakage. However, this additional noise is not sufficient to significantly raise the security level. That is why, the authors presented another and new idea. Since they only need 256 plaintexts to build their PRF, they can use the plaintexts $p_j = j||j||...||j$, with $j \in [0.255]$. When mounting a DPA on such a construction, the attacker cannot target a particular Sbox since she cannot distinguish them by their inputs. As empirically evaluated by Medwed, Standaert and Joux, this improvement is reflected in an higher security level.

Chapter 3

Contributions of this Thesis

In this thesis, we study the security of symmetric cryptographic algorithms against power-analysis attacks. In particular, we develop new attacks and we propose several countermeasures. The former is presented in Part II while the latter is discussed in Part III.

Regarding side-channel attacks, we present new vulnerabilities in cryptographic algorithms. In particular, we focus on multiplications involving a known variable and a large secret (*e.g.,* 128 bits). We describe two new attacks when the attacker is limited to the observation of their outputs. Since these outputs are highly dependent on a large secret, such attacks were not thought possible until then.

With respect to protections against power-analysis attacks, we focus on two widely used algorithmic countermeasures: leakage-resilient primitives and masking schemes. Regarding the former, we propose a new leakage-resilient and efficient encryption scheme based on fresh re-keying. Concerning the latter, we build a practical tool able to formally verify their security in the t-threshold probing model (*e.g.,* up to order 5 for multiplications) and to deliver the corresponding security proof. Until then, such tools did not exceed verification above the second order. In order to go further in program size and masking order, we also tackle composition issues. So far, either the composition of secure functions was assumed to be secure or the masks were very frequently refreshed. However, the latter is not tight for security in the t-threshold probing model. Thus, we establish new theorems and build a second tool able to generate a secure algorithm at any order in the value-based t-threshold probing model from its non protected version. Finally, in order to identify the best protection between masking and leakage-resilient primitives in a real implementation, we define a framework to compare their impact according to the practical constraints and the targeted security level.

Contents

3.1 Power-Analysis Attacks on Multiplications

Part II of this thesis is dedicated to power-analysis attacks on cryptographic algorithms. In particular, we concentrate on multiplications between a known variable and a large secret (*e.g.,* 128 bits), when the attacker is limited to the observation of their outputs. We mainly focus on the Galois field multiplication in $\mathsf{GF}(2^{128})$ used, for instance, in the authenticated encryption algorithm AES-GCM. But we also extend our attacks to the multiplication involved in the fresh re-keying algorithm proposed by Medwed *et al.* in [MSGR10].

Most of power-analysis attacks rely on a divide-and-conquer strategy. However, each bit of such multiplications outputs depend on all the input secret key bits, making this classical strategy inapplicable. Furthermore, the physical leakage corresponding to these multiplications is usually modeled by the noisy Hamming weight of their outputs, which is a non-linear function of the key bits. Since the complexity of solving systems of such higher degree noisy equations generally exceeds an attacker power, we cannot directly exploit such relations. However, we notice that the less significant bit (or parity bit) of the outputs Hamming weight was actually a linear function in the key bits. Thus, from the leakage of the multiplications with different inputs, we can form a linear system of equations. Since the collected Hamming weights are noisy, the second member of our linear system contains a number of errors which depends on the noise level. Using decoding or LPN (Learning Parity with Noise) algorithms, the key can still be recovered, at least for low levels of noise. *This work is described in Part II, Chapter 3 and was published in the proceedings of Asiacrypt 2014 [BFG14].*

In order to improve this key recovery for higher levels of noise, we then present a new attack which relies, this time, on the most significant bits of the Hamming weight values of the multiplication's outputs. Indeed, filtering very high or very low such Hamming weight values isolates multiplication's outputs with a large quantity of ones or a large quantity of zeros. Thus, fixing all the bits of a multiplication's output whose Hamming weight is high (resp. low) to one (resp. zero) gives a linear system with a few errors. In particular, the errors probability is much less sensitive to the noise (even though it also depends on the filtering level). As done for the initial attack, we can solve the linear system using LPN algorithms. But for this work, we show how to improve these algorithms to reduce the number of required queries (in our case, consumption traces). *This second work is described in Part II, Chapter 4 and was published in the proceedings of CHES 2015 [BCF+15a].*

3.2 Countermeasures Based on Leakage-Resilient Cryptography Model and Masking

The study of countermeasures against power-analysis attacks and the construction of new instantiations cover Part III of this thesis. It is organized around the two main algorithmic countermeasures: leakage-resilient primitives and masking schemes.

3.2.1 Leakage-Resilient Encryption Scheme

In this thesis, a particular attention is paid to the security proofs of countermeasures and an effort is made to keep them competitive in terms of performances. Regarding the countermeasures based on leakage-resilient cryptography, we build an efficient and proven leakage-resilient secure encryption scheme based on the AES block cipher.

In this purpose, our first step is to prove that the combination of a leakage-resilient PRF (to derive new keys) and a block cipher was leakage-resilient secure. We then suggest to instantiate

the block cipher with the AES and the PRF with the proven secure construction of Faust, Pietrzak and Schipper [FPS12]. In order to improve the efficiency of the whole scheme and the synchronization between two parties, we modify the PRF in order to use more nodes of the binary tree. Doing so, the fresh re-keying function does not satisfy the PRF properties anymore. However, despite the weaknesses of the re-keying function, we manage to prove that the whole scheme remained secure in the leakage-resilient cryptography model.

This work is described in Part III, Chapter 2 and was published in the proceeding of CHES 2013 [ABF13].

3.2.2 Security of Masked Implementations

After the study of leakage-resilient primitives, we focus on masking which is currently the most widely used countermeasures against side-channel attacks. During the last few years the community has made a significant effort to propose higher-order masking schemes. Unfortunately, some of them were finally broken, in spite of their pen-and-paper security proofs. At the end, these security proofs were revealed incomplete or erroneous because they are generally hard to verify. In this thesis, we try to solve two aspects of these issues. First, with the help of researchers with expertise in formal methods, we build a formal tool able to verify the security of masked implementations in the t-threshold probing model. For a given order t, the tool either confirms the security of the algorithm in this model or exhibits a potential attack path. Then, we build a compiler able to generate secure implementations at any order.

Verification of Higher-Order Masking Schemes

The main goal of this work is to verify the security of implementations claimed secure at a given order in the t-threshold probing model. In practice, several proven masking schemes appeared to be insecure a few years after their publication. Either their proof was erroneous or their implementation induced security flaws. In both cases, current formal verification methods are generally limited in higher orders or program sizes and may fail to detect such issues.

To realize this verifier, we proceed in two steps. First, we set up rules to verify that a set of t variables is independent from the secrets. Then, we optimize the verification of all the possible sets of t variables in a program. Indeed, the enumeration of all these sets makes impossible the evaluation of a classical block cipher implemented at the first order. We thus suggest to verify the security of much larger sets of variables ($> t$), what can be linearly done with their growth. As a result, we manage to verify larger algorithms at higher orders.

This work is described in Part III, Chapter 3 and was published in the proceedings of Eurocrypt 2015 [BBD+15b].

Automatic Generation of Higher-Order Masking Schemes

The previous tool makes possible the verification of the composition of secure functions at a fixed order. However, it does not explain why some compositions are secure whereas other come with security breaches. Therefore, we then focus on this issue. By describing the algorithms as graphs, we observe that the sensitive points correspond to cycles, that is when the masks of a same variable are reused in different operations. The solution is then to refresh these masks before the reuse as suggested in [RP10]. However, to break the dependencies between the several reuses, we notice that the refreshing algorithm should satisfy specific properties. In particular, if t_i observations are made on the intermediate variables of the refreshing algorithm and t_s are made on its outputs ($t_i + t_s \leqslant t$), they should all be simulated with only t_i input masks (instead

of t for classical functions). With this requirement, we obtain sufficient independence despite the different reuses.

Based on these analyses, we build a compiler which takes as input a C implementation and a typing of the underlying functions and returns a functionally equivalent C implementation secure in the t-threshold probing model. We reuse the higher-order algorithms proposed in the literature for basic operations (*e.g.,* multiplication, linear functions, refreshing) and we verify their proof using formal methods. These functions and our new compositions theorems make possible the construction of proofs and the generation of compositions with a few number of refreshing calls.

This work is described in Part III, Chapter 4.

3.2.3 Masking, Leakage-Resilient Primitives or Both?

As presented above, the countermeasures of masking and leakage-resilient primitives are widely deployed and each of them has pros and cons in terms of security and performances. Most of the works on these subjects are completely specific to one of these countermeasures. Thus, in this last subject, we try to build a comparison between these protections in order to help industrial experts to protect their implementation at a given security level and with specific performances constraints. We limit our work to the application of both countermeasures on PRFs and PRGs using the AES in both scenarios. The main difference between these two primitives for this context is that PRGs maintain a state between two invocations whereas PRFs return outputs which depend only on the current input and the secret key. In particular, contrary to PRFs, PRGs limit the number of measurements of the same secret key by construction.

To compare these countermeasures, we investigate the possibility to obtain security bounded implementations, that is, whether we can guarantee a security level whatever the number of measurements. While this bounded security is easily achievable for PRGs thanks to their changing internal state, the situation is much more complicated for PRFs. In practice, only specific constructions may achieve this bounded security. For instance, the PRF proposed by Medwed Standaert and Joux in [MSJ12] exploits the data parallelism to achieve this property. From these observations and many simulations in different contexts (*e.g.,* hardware and software, with or without different maskings), we notice that the best protection with our security/efficiency trade-off is the use of fresh re-keying (*i.e.,* leakage-resilient primitive) for PRGs and masking alone for PRFs. In particular, we show that the use of a single countermeasure in each situation is generally more efficient than their combination.

This work is described in Part III, Chapter 5 and was published in the journal Cryptography and Communications [BGS15].

3.3 Publications

3.3.1 International Conferences Proceedings

[1] Sonia Belaïd, Jean-Sébastien Coron, Benoît Gérard, Pierre-Alain Fouque, Jean-Gabriel Kammerer, and Emmanuel Prouff. Improved Side-Channel Analysis of Finite-Field Multiplication. To appear in *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015. Proceedings*, 2015.

[2] Michel Abdalla, Sonia Belaïd, David Pointcheval, Sylvain Ruhault, and Damien Vergnaud. Robust pseudo-random number generators with input secure against side-channel attacks.

To appear in *Applied Cryptography and Network Security - ACNS 2015 - 13th International Conference, Columbia, New York, US, June 2-5, 2015. Proceedings*, 2015.

[3] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. 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 457–485, 2015.

[4] Sonia Belaïd, Pierre-Alain Fouque, and Benoît Gérard. Side-channel analysis of multiplications in GF(2128) - application to AES-GCM. In *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 II*, pages 306–325, 2014.

[5] Michel Abdalla, Sonia Belaïd, and Pierre-Alain Fouque. Leakage-resilient symmetric encryption via re-keying. In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, pages 471–488, 2013.

[6] Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. Differential Power Analysis of HMAC SHA-2 in the Hamming weight model. In *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*, pages 230–241, 2013.

3.3.2 Journal Articles

[1] Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. Differential Power Analysis of HMAC SHA-1 and HMAC SHA-2 in the Hamming weight model. To appear in *E-Business and Telecommunications - International Joint Conference, ICETE 2014, Revised Selected Papers*, Springer, 2015.

[2] Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? In *Cryptography and Communications*, 7(1):163–184, 2015.

[3] Sonia Belaïd, Fabrizio De Santis, Johann Heyszl, Stefan Mangard, Marcel Medwed, Jörn-Marc Schmidt, François-Xavier Standaert, and Stefan Tillich. Towards fresh re-keying with leakage-resilient PRFs: cipher design principles and analysis. In *Journal of Cryptographic Engineering*, 4(3):157–171, 2014.

Part II

Cryptanalysis of Multiplications

Chapter 1

Introduction

This chapter aims to introduce the cryptanalysis problem we tackled, that is, recovering the 128-bit secret key used in a field/ring multiplication with the limited knowledge of the noisy Hamming weight outputs. Two algorithms are targeted: the AES-GCM with a multiplication in $\mathsf{GF}(2^8)$ and the fresh re-keying of Medwed, Standaert, Großschädl and Regazzoni with a different ring multiplication. The first attack mainly reveals the vulnerabilities of such multiplications against side-channel attacks and the second one finally leads to key recovery on real hardware 128-bit implementations (*i.e.,* with realistic levels of noise).

Contents

1.1 Motivation

The cornerstone of side-channel analysis is that information about some key-dependent variable x leaks through *e.g.,* the power consumption or the electromagnetic information of the device manipulating x. A side-channel attack classically follows a *divide-and-conquer* approach and the secret is recovered by exhaustively testing the likelihood of every possible value for every secret piece. This *modus operandi* implicitly assumes that x depends on a short portion of the secret (for example, only 8 bits if x corresponds to the output of the AES Sbox). It is particularly suited to the context of software implementations where the processing is sequentially splitted into operations on data whose size depends on the device architecture (*e.g.,* 8 bits or even 32 bits for smart cards).

However, in hardware implementations, many operations are performed simultaneously and the leakage at each time may depend on data whose size is much larger (*e.g.,* 128 bits). If some bits of the data being observed depend only on a few bits of the secret, a side-channel attack can still apply with a *divide-and-conquer* strategy. But in this part, we focus on 128-bit multiplications whose the main advantage for cryptographic algorithms is to diffuse the secret. Particularly, we mainly focus on the multiplication in $\mathsf{GF}(2^{128})$ used for instance in the authentication encryption mode AES-GCM and we also study the modular multiplication in $\mathsf{GF}(2^8)[y]/\left(y^{16}+1\right)$ introduced for fresh re-keying purposes.

The main motivation here is to show that such multiplications, although manipulating huge part of the secret at once, can be attacked by a side-channel adversary. Hence, in this analysis we consider that the multiplication is an atomic operation (that is performed using a 128-bit multiplier), which is the worst case for an attacker.

For the sake of clarity, we mainly focus on the application to AES-GCM, proposed by McGrew and Viega in [MV05] and standardized by NIST since 2007. This authenticated encryption algorithm aims to provide both confidentiality and integrity. It combines an encryption based on the widely used AES algorithm in counter mode and an authentication based on the GHASH function involving multiplications in $\mathsf{GF}(2^{128})$. The latter mixes ciphertexts, potential additional data and a secret parameter derived from the encryption key in order to produce a tag. The security of the algorithm has been analyzed by many authors but despite significant progress in these attacks [PC14, HP08, Saa12], there is currently no real attack on this mode. The most efficient attacks are described by Ferguson when the tag is very short (32 bits) [Fer05] and by Joux when the nonces are reused [Jou06]. In this particular case of AES-GCM, attacking the multiplier will provide to the attacker the knowledge of the authentication key k. Due to the malleability of the counter mode, the knowledge of the authentication key induces a huge security breach and thus protecting the multiplier is of real importance in contexts where a side-channel attacker is considered. Notice that if the multiplication is protected, then the simple additional countermeasure that consists in masking the tag register is enough to thwart the proposed attack.

We eventually show how to apply our attacks on a different multiplication which is used in the fresh re-keying primitive proposed by Medwed *et al.* in [MSGR10]. In this context, an additional operation is performed before the call to the block cipher to derive a new session key for each encryption block. Thus, the block cipher is generally expected to be resistant to SPA and the security relies on the re-keying primitive. Since the latter requires less mathematical properties than the block cipher, it is generally easier to protect. In this part, we not only exhibit an attack on this protocol but we also show that despite the aforementioned security separation, we can perform a DPA by only observing leakage in the block cipher since it also manipulates secret-dependent chunks.

1.2 Related Work

Concerning the AES-GCM, Jaffe describes in [Jaf07] a very efficient DPA on its encryption counter mode. Basically, the main idea is to use a DPA attack on the two first rounds of the AES block cipher. Then, as most of the plaintext is the same between two evaluations, it is possible to recover the secret key by guessing parts of the first and second round subkeys. This attack is particularly efficient since it also recovers the counter if it is hidden. However, the implementations of AES are now well protected using masking and many papers proposed such countermeasures [CJRR99, RP10, Cor14, GPS14], so that we can assume that it is not possible to recover the secret key on the encryption part.

Regarding the multiplication-based fresh re-keying, Dobraunig *et al.* describe in [DEMM14] a black-box collision attack. The latter relies on the possibility to reuse the same plaintexts with different session keys. However, since the re-keying primitive is assumed to be protected, as far as we know, no side-channel attack has been proposed on this scheme so far.

Some of the algorithms that we consider here to recover the secret key come from the coding theory and we think it is a nice view to cast many side-channel attacks. Indeed, a secret value k for instance is encoded as the different leakage values obtained by the adversary. Usually, these

leakages allow to recover k, but here for 128 bits, the Hamming weight does not give enough information. Moreover, we only get noisy versions of the leakage values and these values form the codeword with errors. The errors are independent from each other and the noise level is rather high as in many stream cipher cryptanalysis. Given these values, the goal of the adversary is to recover the original message k and the adversary faces a classical decoding problem.

1.3 Contributions

In this part, we consider a particular leakage model where only the storage of values leaks information to the adversary. We assume that, each time a value is written in a large register, a noisy version of the Hamming distance or the Hamming weight of this value can be learned by the adversary. For instance, in the context of the AES-GCM authentication, the Hamming weight of the multiplication result between the authentication key k and some known value a can be learned the first time the register is written.

Relying on this leakage model, our contribution is to attack the multiplication when $n = 128$ following two different methods. In a first attempt, we remark that the least significant bit of the Hamming weight of $k \cdot a$ can be expressed as a linear function of the bits of k. With 128 such equations and without noise, it is thus straightforward to recover k. But in presence of noise, the resulting equations contain errors. Thus, we show that relying on LPN and decoding algorithms, we can still recover the key for high signal-to-noise ratios (*e.g.,* low levels of noise). *This work was presented at Asiacrypt 2014 [BFG14].*

In a second attempt, we exhibit a different method which consists in filtering the multiplication's results based this time on the most significant bits of the Hamming weight. As the previous method, this filtering results in noisy linear equations on the key bits which may be solved using LPN algorithms. This second method benefits from being quite insensitive to noise and leads to key recovery even for low SNR. *This work was presented at CHES 2015 [BCF+15a].*

1.4 Outline

Chapter 2 describes the AES-GCM and the multiplication-based fresh re-keying, the attacker context and defines the problem we tackle. Chapter 3 describes the first attack based on the less significant bit of the Hamming weight. Then, Chapter 4 describes the second attack based on the most significant bits of the Hamming weight. Eventually, Chapter 5 concludes and gives some perspectives.

Chapter 2

Background, Leakage and Attacker Models

In this chapter, we define the theoretical background required to understand the two power-analysis attacks that will be described in Chapter 3 and Chapter 4. Namely, both targeted algorithms are described, the attacker capabilities are given and the problem is settled.

Contents

2.1 Two Multiplication-Based Algorithms

In this section, we describe the two algorithms which will be targeted by our power-analysis attacks. They both contain a multiplication involving a 128-bit secret key.

2.1.1 AES-GCM description

AES-GCM is an authenticated encryption algorithm which aims to provide both confidentiality and integrity. It combines an encryption based on the widely used AES algorithm in counter mode and an authentication with the Galois mode. The so-called *hash key* k used for the authentication is derived from the *encryption key* k_{enc} as $k = \mathsf{AES}_{k_{\text{enc}}}(0^{128})$. The ciphertext of the encryption is denoted as $c^{(1)}, \ldots, c^{(\ell)}$ where the blocks $c^{(i)}$ have 128 bits length except $c^{(\ell)}$ which is of size u ($u \leqslant 128$). Similarly, the additional *authenticated data* is composed of 128-bit blocks $d^{(1)}, \ldots, d^{(m)}$ where the last one has size ν ($\nu \leqslant 128$). Eventually, we denote by $\{z^{(i)}\}_{0 \leqslant i \leqslant m+\ell+1}$ the intermediate results of function GHASH with $z^{(m+\ell+1)}$ being the output exclusively ored with an encryption of the initial counter to form the tag. Figure 2.1 illustrates

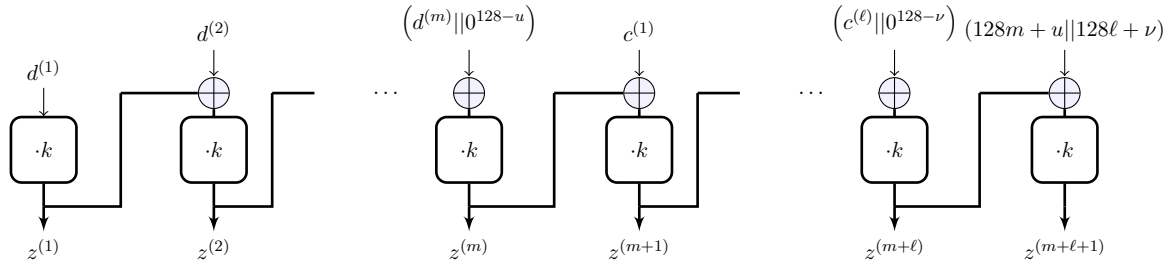


Figure 2.1: AES-GCM authentication

the main part of the procedure with the previously defined notations. For the sake of simplicity we will use a single letter a for both kinds of inputs (d or c). Then, the definition of the GHASH function can be simply described by the following recursion

$$z^{(i+1)} = \left(z^{(i)} \oplus a^{(i+1)}\right) \cdot k, \tag{2.1}$$

where $\cdot$ is the Galois Field multiplication described below.

Notation. *In the context of AES-GCM and when it is not explicitly specified, the a^i's (resp. the z^i's) will denote the known variable inputs (resp. outputs) of the multiplication per k, that are generally the different inputs (resp. outputs) of the first block.*

Galois Field Multiplication

For any positive integer n, the finite field of 2^n elements is denoted by $\mathsf{GF}(2^n)$ and the n-dimensional vector space over $\mathsf{GF}(2)$ is denoted by $\mathsf{GF}(2)^n$. Choosing a basis of $\mathsf{GF}(2^n)$ over $\mathsf{GF}(2)$ enables to represent elements of $\mathsf{GF}(2^n)$ as elements of $\mathsf{GF}(2)^n$ and *vice versa*. In the following, we assume that the same basis is always used to represent elements of $\mathsf{GF}(2^n)$ over $\mathsf{GF}(2)$.

This chapter analyses the multiplication in the field $\mathsf{GF}(2^n)$, with a particular focus on $n = 128$. For the AES-GCM protocol, the Galois' extension is defined by the primitive polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$. We denote by α a root of this polynomial P. An element a in $\mathsf{GF}(2^{128})$ can be represented by a vector of coefficient $(a_0, a_1, ..., a_{127})$ where $a = \bigoplus_{0 \leqslant i \leqslant 127} a_i \cdot \alpha^i$. If $a = (a_0, a_1, \cdots, a_{127})$ and $k = (k_0, k_1, \cdots, k_{127})$ are two elements of $\mathsf{GF}(2^{128})$ viewed as 128-bit vectors, the multiplication $a \cdot k$ can be represented by a matrix/vector product in the following way:

$$\begin{pmatrix} a_0 & a_{127} & \cdots & a_1 \oplus a_{127} \oplus a_{126} \\ a_1 & a_0 \oplus a_{127} & \cdots & a_2 \oplus a_{123} \oplus a_1 \oplus a_{127} \oplus a_{122} \\ \vdots & \vdots & \ddots & \vdots \\ a_{127} & a_{126} & \cdots & a_0 \oplus a_{127} \oplus a_{126} \oplus a_{121} \end{pmatrix} \cdot \begin{pmatrix} k_0 \\ k_1 \\ \vdots \\ k_{127} \end{pmatrix} = \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{127} \end{pmatrix} \tag{2.2}$$

where the product $\cdot$ is processed over $\mathsf{GF}(2)$.

2.1.2 Multiplication-Based Fresh Re-keying

The core idea of the fresh re-keying countermeasure originally proposed in [MSGR10] for block cipher algorithm is to create a new session key from a public nonce for each new processing of the encryption algorithm. It guaranties that the secret (master) key is never used directly. To allow for the decryption of the ciphertext, the latter one is sent together with the nonce. For soundness,

the key randomization (aka fresh re-keying) must satisfy two properties. First, it must be easy to protect against side channel attacks. Secondly, it must have a good *diffusion* so that each bit of the new (session) key depends on a large number of bits of the master key, rendering attacks based on key-hypotheses testing inefficient. To satisfy the first property, [MSGR10] proposes to base the randomization on linear functions. Efficient techniques are indeed known to secure the latter functions against SCA (*e.g.* higher-order masking can be efficiently applied, as it has linear complexity for linear functions [ISW03, CGP+12]). To additionally satisfy the second property, [MSGR10] proposes to define the linear functions from *circulant* matrices deduced from the random nonce.

Ring Multiplication

Let $k \in \mathsf{GF}(2^8)^m$ denote the master key which must be protected and let $a \in \mathsf{GF}(2^8)^m$ denote the nonce (generated at random). The square matrix whose lines correspond to all the rotations of the byte-coordinates of a (*e.g.* the ith row corresponds to the vector a right-rotated i times) is denoted by $\mathsf{circ}(a_0, \cdots, a_{m-1})$. It satisfies:

$$\mathsf{circ}(a_0, \cdots, a_{m-1}) = \begin{pmatrix} a_0 & a_{m-1} & a_{m-2} & \cdots & a_1 \\ a_1 & a_0 & a_{m-1} & \cdots & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_0 \end{pmatrix} , \tag{2.3}$$

and the session key k' is deduced from (k, a) as follows:

$$k' = \mathsf{circ}(a_0, \cdots, a_{m-1}) \cdot k \ , \tag{2.4}$$

where $\cdot$ denotes the scalar product in $\mathsf{GF}(2^8)^m$. Equation 2.4 implies in particular that the i^{th} byte of k' satisfies:

$$k'_i = \sum_{j=0}^{m-1} a_{i+j \bmod m} \otimes k_j \ , \tag{2.5}$$

where $\otimes$ denotes the multiplication on $\mathsf{GF}(2^8)$.

Notation. *In the following, since the analysis of both multiplications are distinct, we equivalently denote by $\cdot$ both multiplications.*

2.2 Attacker Context

2.2.1 Leakage Model

A usual assumption (see Part I, Section 2.2.2) when there is no information on the implementation is to consider that the processing leaks a noisy observation of the Hamming weight of the manipulated values. Namely, for such manipulated value $z \in \mathsf{GF}(2)^n$, it is assumed that the adversary obtains the following observation $\mathcal{L}(z)$:

$$\mathcal{L}(z) = \mathsf{HW}(z) + \varepsilon \ , \tag{2.6}$$

with an independent Gaussian noise ε satisfying $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

A common generalization of this leakage model when the attacker is given the successive stored variables is to consider the *Hamming distance* (HD) between two consecutive data $z^{(i-1)}$ and $z^{(i)}$:

$$L_i^{(\mathsf{HD})} = \mathsf{HD}(z^{(i)}, z^{(i-1)}) + \varepsilon_\sigma = \mathsf{HW}(z^{(i)} \oplus z^{(i-1)}) + \varepsilon.$$

This generalization depends on the implementation. If the register is initialized to zero before storing variable $z^{(i)}$, the Hamming distance between both stored data is exactly the Hamming weight of $z^{(i)}$. Otherwise, we can reasonably assume that the new computed variable overwrites the stored one (intermediate result), leaking the Hamming distance between them.

In the following, the level of noise in the observations is rather quantified with the signal-to-noise ratio (SNR, defined in Part I), that is the ratio between the signal variance and the noise variance. In this work, we assume that data are manipulated by blocks of n bits and that they leak their noisy Hamming weight. Therefore, we consider as signal the Hamming weight of n-bit variables, which follows a binomial law $\mathcal{B}(n, 1/2)$ with variance equal to $n/4$, and as noise the Gaussian noise ε defined in Equation (2.6) which follows a normal law $\mathcal{N}(0, \sigma^2)$ with variance σ^2. Consequently, in this context, the signal-to-noise ratio is equal to $\frac{n/4}{\sigma^2} = \frac{n}{4\sigma^2}$. This value is a useful notion to compare different contexts where the variances of both the signal and the noise are different (which will be the case when comparing our simulations in the Hamming weight model and our experiments on a real device).

In both attacks, the main purpose is to show that we can recover the key k when we only observe $\mathcal{L}(k \cdot a^i)$ for many known a^i's. Thus, we assume that we do not have access to the internal leakage of the field multiplication $k \cdot a^i$ and we also assume that the n-bit results are stored in n-bit registers, which is the worst case from an attacker point of view.

2.2.2 Attacker Model

Now we have defined the models for information leakage, we discuss the attacker capabilities. From the axiom *only computation leaks* of Micali and Reyzin [MR04], we only give the attacker the leakage of the manipulated data. Furthermore, in most of this part, we restrict the leaking data to the multiplication's output to cover all the implementations. We now discuss the three characteristics that define the attacker model.

Known/Chosen Inputs. For the known operands of the Galois field multiplication, we consider the two classical attacker models namely the known message model (e.g., ciphertexts) and the chosen message model (e.g., additional data to authenticate).

Limited/Unlimited Queries. The attacker may face limitations in the number of queries. Such limitations may be due to time constraints but we may also consider an attacker querying for forged tag verifications in which case an error-counter may limit the number of invalid tag verifications.

Enabled/Disabled Averaging. Eventually, the attacker may be able to average traces obtained for the same computation. This is the case in the chosen messages setting but it may also be the case in the known messages setting when for instance the first blocks to authenticate have a specific format. If such feature is available, then the attacker may execute λ times each computation and average the corresponding traces. Since the leakage model considers an additive Gaussian noise, this decreases the standard deviation of the noise from σ to $\sigma/\sqrt{\lambda}$. Notice that this is purely theoretical and in practice one may experiment limitations to this idealized model. This implies that even in a *chosen* setting, the noise may remain high.

In the following, we aim at presenting various ways of exploiting the leakage. Each of the presented techniques may be relevant in at least one combination of the three aforementioned criteria.

2.3 Learning Parities with Noise.

As briefly explained in the introduction, the problem of recovering a secret k from noisy observations of $\mathsf{HW}(a \cdot k)$ relates to the well known LPN problem whose definition is recalled hereafter.

Definition 1 (Learning Parity with Noise (LPN) Problem)**.** *Let $k \in \mathsf{GF}(2)^n$ and $p \in [0, 1/2]$. Given a family of ν values $(a^i)_{0 \leqslant i < \nu}$ in $\mathsf{GF}(2)^n$ and the family of corresponding observations $(b^i = \langle a^i, k \rangle + e^i)_{0 \leqslant i < \nu}$, where $\langle \cdot, \cdot \rangle$ denotes the scalar product $\in \mathsf{GF}(2)^n$ and where the a^i are drawn uniformly in $\mathsf{GF}(2^n)$ and the e^i are generated according to Bernoulli's distribution $\mathsf{Ber}(p)$ with parameter p, recover k. We classically denote by δ the bias equal to $1 - 2p$.*

We denote by $\mathsf{LPN}(n, \nu, p)$ an instance of the LPN problem with parameters (n, ν, p). In the following, the noisy equations $\langle a^i, k \rangle + e_i$ will come from the noisy observations of a device performing field (or ring) multiplications in the form $z = a \cdot k$ in $\mathsf{GF}(2^n)$.

2.3.1 Blum-Kalai-Wassermann Algorithm

Blum, Kalai and Wassermann described in [BKW00] a sub-exponential algorithm for solving the LPN problem. This algorithm, referred to as BKW, consists in performing a clever Gaussian elimination by using a small number of linear combinations in order to reduce the dimension of the problem. In particular, given as input $b_i = \langle a^i, k \rangle + e_i$ for known a^i's, the goal of the BKW algorithm is to find linear combinations of the a^i's with ℓ terms such that:

$$a^{i_1} \oplus \cdots \oplus a^{i_\ell} = \mathrm{u}_j \ , \tag{2.7}$$

where $(\mathrm{u}_j)_{1 \leq j < n}$ is the canonical basis, that is u_j has its j^{th} coordinate equal to 1 and the other coordinates are 0. Thus we get:

$$\langle \mathrm{u}_j, k \rangle = k_j = \bigoplus_{r=1}^{\ell} b_{i_r} \oplus \bigoplus_{r=1}^{\ell} e_{i_r}.$$

From this equation, we can compute the new bias of the linear combination of equations using the Piling-Up lemma. Since for ℓ variables $e_1, \ldots, e_\ell$, $\mathbb{P}[e_i = 1] = p = (1 - \delta)/2$, we can verify that $\mathbb{P}[e_1 \oplus \cdots \oplus e_\ell = 1] = \frac{1 - \delta^\ell}{2}$. Therefore, if we sum ℓ error terms, the resulting error term e is such that $\mathbb{P}[e = 1] = (1 - \delta')/2$ with $\delta' = \delta^\ell$. If ℓ is not too large, then the bias of the error term $\bigoplus_{r=1}^{\ell} e_{i_r}$ is also not too large and with enough such equations and a majority vote, we can recover the j^{th} coordinate of k.

To find linear combinations satisfying Equation (2.7), the first step is to split the a^i's into α blocks of β bits, where $n = \alpha \cdot \beta$ (*e.g.,* for $n = 128$ we can take $\alpha = 8$ and $\beta = 16$). Initially, there are ν vectors a^i. These vectors are first sorted into 2^β classes according to the value of their rightmost β bits. Afterwards, in each class, all the elements are xored with a single element of it. This element is then discarded. Hence, there are at least $\nu - 2^\beta$ new vectors $a^{i^{(1)}}$, whose rightmost β bits are zero; these $a^{i^{(1)}}$ are the xor of 2 initial vectors a^i. The algorithm continues recursively. For the next block of β bits we get at least $\nu - 2 \cdot 2^\beta$ vectors $a^{i^{(2)}}$ whose rightmost 2β bits are zero; these new vectors $a^{i^{(2)}}$ are the xor of 4 initial vectors a^i. Stopping at the last-but-one block, we get at least $\nu - (\alpha - 1) \cdot 2^\beta$ vectors, for which only the first β-bit block is possibly non-zero, and which are the xor of $2^{\alpha - 1}$ initial vectors a^i. Among these $\nu - (\alpha - 1) \cdot 2^\beta$ vectors, we select the ones equal to the basis vectors u_j and we perform a majority vote. With the xor of $\ell = 2^{\alpha - 1}$ vectors, the bias is $(1 - 2p)^{2^{\alpha - 1}}$. Therefore, the majority vote roughly needs $c/(1 - 2p)^{2^{\alpha - 1}}$ such vectors, for some logarithmic factor c [BKW00].

2.3.2 Improved Algorithms

A variant of BKW algorithm is described by Levieil and Fouque in [LF06]. It finds linear combinations similarly, however at the end, it uses a Walsh Transform to recover the last b bits of the secret key k at once. In an other approach, Kirchner proposed to use the secret-error switching lemma [Kir11, ACPS09] to further improve the method. Later, Arora and Ge in [AG11] proposed an algebraic approach in a specifically structured noise and recently, Guo *et al.* proposed to use error-correcting code [GJL14].

2.4 Hidden Multiplier Problem

We assume in the following that when performing a multiplication $a \cdot k$ over $\mathsf{GF}(2^n)$ (or equivalently the fresh re-keying modular multiplication) for some known a, only the Hamming weight of the result $a \cdot k \in \mathsf{GF}(2^n)$ is leaking, with some noise; the goal is to recover the secret multiplier k. Formally, the SCA amounts to solve the following problem:

Definition 2 (Hidden Multiplier Problem)**.** *Let $k \leftarrow \mathsf{GF}(2^n)$. Let $\ell \in \mathbb{N}$. Given a sequence $(a^i, \mathcal{L}^i)_{1 \leq i \leq \ell}$ where $a^i \leftarrow \mathsf{GF}(2^n)$ and $\mathcal{L}^i = \mathsf{HW}(a^i \cdot k) + \varepsilon^i$ where $\varepsilon^i \leftarrow \mathcal{N}(0, \sigma^2)$, recover k.*

Chapter 3

Power Analysis on Multiplications Based on LSB

In this chapter, we propose a first method to solve the Hidden Multiplier Problem (Definition 2). We mainly assume that the multiplication is performed in $\mathsf{GF}(2^{128})$ in the context of the AES-GCM authentication but we also consider another modular multiplication used for fresh re-keying purposes in the last section. Our key point is that the least significant bit of the Hamming weight's multiplication output can be expressed as a linear function of the bits of the secret key k. If we are able to find 128 such equations, then it is easy to recover k. However, in side-channel attacks, we only access to noisy versions of the Hamming weight and then, the problem becomes more difficult. Classically, this problem has been known as the Learning Parities with Noise (LPN) [BKW03] and it is famous to have many applications in cryptography. We then consider many attacker models, according to whether the inputs a are known, can be chosen and repeated. If we consider only the tag generation algorithm, additional authentication data can be input to the encryption and these values are first authenticated. We think that this model is powerful and allows us to consider many attacks on different implementations. For instance, since we only consider the writing in the accumulator of the polynomial evaluation, we do not take into account the way the multiplication is implemented and our attack also works even though the multiplication is protected against side-channel attacks as long as its results are stored clear.

Contents

3.1 Attack on Known Inputs

In this section, we present our new power-analysis attack on the multiplication of AES-GCM. We first explain how to build a system of equations linear in the secret key bits but containing

some errors. Then, we show how to solve this system according to the level of noise and the number of equations.

3.1.1 Construction of a Linear System in the Key Bits

We first present the key idea of our attacks.

Main Observation. The cornerstone of the attacks presented in this chapter is the fact that the less significant bit (further referred to as LSB) of the Hamming weight of a variable (equivalently distance between two variables) is a linear function of its bits. While a side-channel attacker generally uses a divide-and-conquer strategy to recover small parts of the key by making guesses, it is not possible anymore as the size of chunks gets large. In the case of the Galois field AES-GCM multiplication between a known input and a secret key, each bit of the result depends on all the 128 bits of the secret key. Nevertheless, we noticed that the Hamming weight of this result's LSB is a linear function of the input and the key. If we denote by $\mathsf{lsb}_0(\mathsf{HW}(z))$ (or also b_0) the bit 0 of the Hamming weight of the product $k \cdot a$, we get

$$\mathsf{lsb}_0(\mathsf{HW}(z)) = \bigoplus_{i=0}^{127} \left(\bigoplus_{j=0}^{127-i} a_j \right) k_i. \tag{3.1}$$

This is precisely what is exploited in the attacks we present. Obviously this work can also be applied to any algorithm in which such multiplication appears and is not restricted to AES-GCM.

First Block. Observing Equation (2.1), we only know the input of the multiplication with k for the first block of data $a^{(1)}$ since the input of further blocks will depend on k. Moreover, since $z^{(0)}$ is assumed to be zero, we are in a context where Hamming distance and Hamming weight leakage are equivalent and we thus only refer to the Hamming weight in the following. While the linearity of its parity bit is a very good thing from an attacker point of view, the drawback is that this value is highly influenced by the measurement noise. We propose here to use the LLR statistics (for Log Likelihood Ratio) to guess the bit b_0 from a leaked Hamming weight. This statistics is extensively used in classical cryptanalysis (an application to linear cryptanalysis can be found in [BJV04]) since the Neyman-Pearson lemma states that for a binary hypothesis test the optimal[1] decision rule is to compare the LLR to a threshold. The LLR of a leakage ℓ is given by:

$$\mathsf{LLR}(\ell) = \log(\mathbb{P}[b_0 = 0|\ell]) - \log(\mathbb{P}[b_0 = 1|\ell]).$$

The bit b_0 is equally likely to be equal to 0 or 1 since the secret follows *a priori* a uniform distribution. Thus, using Bayes relation we obtain that

$$\begin{aligned} \mathsf{LLR}(\ell) &= \log\left(\mathbb{P}[\ell|b_0 = 0]\right) - \log\left(\mathbb{P}[\ell|b_0 = 1]\right) \quad \text{with} \\ \mathbb{P}[\ell|b_0 = i] &= \sum_{w=0}^{128} \mathbb{P}[\ell|b_0 = i, \mathsf{HW}(\ell) = w]\mathbb{P}[\mathsf{HW}(\ell) = w]. \end{aligned}$$

If the result of $\mathsf{LLR}(\ell)$ is positive, it means that the parity bit is likely to be equal to 0. Otherwise, we should assume that it is equal to 1. We thus define:

$$\widehat{b_0} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \mathsf{LLR}(\ell) \geqslant 0, \\ 1 & \text{otherwise.} \end{cases}$$

[1] For more precisions about this lemma and the meaning of optimal refer to [CT91].

Table 3.1: Bernoulli parameter p for several SNRs with - when $p > 0.5 - 10^{-3}$ and ε when $p \ll 10^{-3}$

SNR	Bernoulli parameter p							
	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7
128	0.31	0.16	0.080	0.040	0.023	0.022	0.022	ε
32	$0.5 - 0.0046$	0.37	0.19	0.096	0.055	0.053	0.053	ε
8	-	$0.5 - 0.0032$	0.38	0.20	0.11	0.11	0.11	ε
2	-	-	$0.5 - 0.003$	0.38	0.21	0.19	0.19	ε

We thus obtain a noisy bit of information that we can also model as follows:

$$\widehat{b_0} = \mathsf{lsb}_0\left(\mathsf{HW}(a^{(0)} \cdot k)\right) \oplus b_{\mathcal{N}}. \tag{3.2}$$

where the error-bit $b_{\mathcal{N}}$ is the potential error due to the Gaussian noise. This error-bit follows a Bernoulli distribution with a parameter p representing the probability of error. In the first column of Table 3.1 we provide a few values of this Bernoulli parameter for several standard deviations. Note that we generally evaluate the complexity of an attack according to the signal-to-noise ratio. For 8-bit implementations[2], we consider this SNR around 0.2 [Man04, BGS15] which is a typical value both for hardware [KSK+10] and software implementations [DRS+12]. It corresponds to a signal variance of 8 (with the chosen leakage model) and a noise variance of 10 (standard deviation around 3). While we do not have reference measurements for 128-bit implementations, we can assume that the noise standard deviation is close, that is around 3.

Other Blocks. The generation of traces is expensive for an attacker and in some models it may also be limited. Therefore the number of traces is generally the main criteria when evaluating the complexity of an attack. In the context of the AES-GCM, the authentication is performed through a chained sequence of multiplications. This is quite frustrating for an attacker to only consider the first block when so much information is available. We will discuss in Section 3.1.2 and Section 3.2 how to exploit some of the following multiplications to obtain more bits of information from a single trace.

Other Leakage Bits. As mentioned above, we only exploit the LSB of the leakage for the attacks because it directly depends on a linear combination of the key bits. However, it is also strongly impacted by the noise which involves multiple errors in the system. In this paragraph, we discuss the complexities of considering further bits of leakage. We first focus on the impact of noise on each of them. Table 3.1 gives the error probabilities $p_i = \mathbb{P}\left[\tilde{b_i} \neq b_i\right]$ corresponding to the bits $\{b_i\}_{0 \leqslant i \leqslant 7}$ for a few values of $\mathsf{SNR}(= 32/\sigma^2)$.

The success of the attack is directly related to the number of errors in the system which decreases together with the increase in the bits indexes. However, the resulting systems are made of equations of higher degrees (exponential with the index):

$$\forall\ 0 \leqslant i \leqslant 7, \quad b_i = \bigoplus_{0 \leqslant j_1 < \cdots < j_{2^i} \leqslant 127} \left(\prod_{\ell=1}^{2^i} \bigoplus_{h=0}^{127} \left((a \cdot \alpha^k)_{j_\ell}\ k_h \right) \right)$$

and thus are more complicated to solve. In particular, the methods capturing the errors removal like LPN and linear decoding unfortunately do not apply on non-linear systems[3]. We thus have

[2] Notice that our attacks also work on 8-bit implementations where they are more efficient since the attacker can capture intermediate leakage on 8-bit values.

[3] A linearized system would involve too many variables to be efficiently solved.

to consider first a solver on the error-free system of equations and then complete its complexity with the errors removal. To the best of our knowledge, one of the most efficient solver is the algorithm F5 [Fau02] provided by Faugère and based on the Gröbner bases. While the solving complexity of the (error-free) quadratic system may be reasonable, it gets computationally impractical when considering the most significant bits[4].

3.1.2 Solving the System

As described in the previous section, for each observed first multiplication, an attacker obtains a noisy Hamming weight value of the output. The LSB of the Hamming weight being linearly dependent on the key (see Equation (3.1)), the attacker can gather many measurements to form a linear system having the authentication key k as solution. In this section, we discuss different techniques to solve this noisy linear system. First, we propose a naive procedure for the attacker to recover the key. Then, we investigate enhancements and other techniques that help decreasing the attack complexity in presence of higher noise.

Naive Attack

From Equation (3.1), the (noisy) linear system formed from t messages $(a^\ell)_{0\leqslant\ell\leqslant t}$ can be written as follows:

$$\mathcal{S} = \begin{cases} \bigoplus_{i=0}^{127} \left(\bigoplus_{j=0}^{127-i} a_j^0 \right) & k_i = \widehat{b_0}^0 \\ \bigoplus_{i=0}^{127} \left(\bigoplus_{j=0}^{127-i} a_j^1 \right) & k_i = \widehat{b_0}^1 \\ & \cdots \\ \bigoplus_{i=0}^{127} \left(\bigoplus_{j=0}^{127-i} a_j^{t-1} \right) & k_i = \widehat{b_0}^{t-1}. \end{cases} \tag{3.3}$$

The values $\widehat{b_0}^\ell$ are obtained as in Equation (3.2), that is, we simply compute the LLR of the observations to approximate the least significant bits. Once the system $\mathcal{S}$ is correctly defined in $\mathsf{GF}(2)$, we can efficiently and directly solve it (*e.g.,* calling `mzd_solve_left` of the library `m4ri` [AB09]) if $\mathcal{S}$ gathers at least n (here 128) linearly independent equations and there is no error in the bits $\widehat{b_0}^\ell$ (*i.e.,* $\widehat{b_0}^\ell = b_0^\ell$).

First, the probability of obtaining a full-rank matrix from n n-bit messages, is

$$\prod_{i=0}^{n-1} (1 - 2^{i-n}).$$

In our context (that is $n = 128$), this probability is close to 0.3. To obtain a full-ranked matrix with high probability (say 0.9) the number of additional messages m must satisfy

$$1 - \prod_{j=1}^{2^m} \left(1 - \prod_{i=0}^{n-1} \left(1 - 2^{i-n} \right) \right) \geqslant 0.9.$$

For $n = 128$, a single additional message ($m = 1$) makes the equation holds. Note however that the full rank condition does not need to be fulfilled to recover the key. If it is not, the attacker should first recover a solution of the system and the kernel of the linear application then test all the solutions to eventually recover the key.

[4]The complexities related to each bit can be computed with the package [Bet].

Second, we consider the negative impact of the measurement noise. The latter introduces errors in the system which thus cannot be solved with classical techniques. A simple (naive) solution is to consider that one of the $\widehat{b_0}^i$'s is erroneous and to solve n times the system with the n corrected vectors. If the key is not found, we can incrementally test all other numbers of errors until the correct key is found. Notice that the inversion is only done once: solving the system with a different second member only requires a matrix/vector multiplication. If e errors are made among the n messages, then the correct key will be found in at most $C_n^{(e)}$ matrix/vector products:

$$C_n^{(e)} = \sum_{i=0}^{e} \binom{n}{i}. \tag{3.4}$$

When the number of errors grows, it quickly becomes computationally hard. For instance, for $e = 6$ and $n = 128$, $C_n^{(e)} \approx 2^{32}$. In the next section, we investigate techniques to decrease the number of errors in $\mathcal{S}$.

Improved Attack

In this section, we propose two improvements for the attack. The first one consists in an optimal decision to guess the Hamming weight LSB. This criteria can also be used to advantageously select 128 traces among many more to limit the errors. The second improvement is to show that an attacker can actually use the leakage obtained from the two first multiplications and not only from the first one.

Optimal Decision Rule. As described above, we use the LLR statistics to approximate as well as we can the lsb of the Hamming weight values. When more than n traces are available, it would be of interest to only select the n most reliable ones to decrease the number of errors in the system. Basically, we would like to take into account the confidence we have in a given bit. For instance, assuming a 0 parity bit from a leakage 64.01 seems more reliable than for a leakage equal to 64.49. Interestingly, the higher the absolute value of the LLR is for a given trace, the more confident we are in the choice. Therefore, an attacker should select the n samples with the highest LLR values to form the system. The point is that those n samples may not be linearly independent. Two solutions can then be chosen:

i) consider a subset of these n samples, solve the system and brute force the remaining bits,

ii) or choose n linearly independent samples from the highest LLR values.

Finding the set of n linearly independent samples maximizing this sum is a combinatorial optimization problem which may be quite hard, thus we use a simple "first come/first selected" algorithm that provides a set of n samples. The algorithm iteratively looks for the sample with the highest absolute LLR value that increases the system rank. Note that the resulting Bernoulli parameters following this algorithm are nearly indistinguishable from the optimal ones (having the highest LLR values but not necessarily linearly independent). Figure 3.1 represents the averaged experimental values (on 10,000 samples) of the Bernoulli parameter p for 500 messages in two scenarios. The blue curve represents the error probability without selection of the best traces while the red one integers this selection among the first linearly independent traces. We can see that the chosen selection of the best LLR allows the attack lower values of SNR to achieve the same Bernoulli parameter.

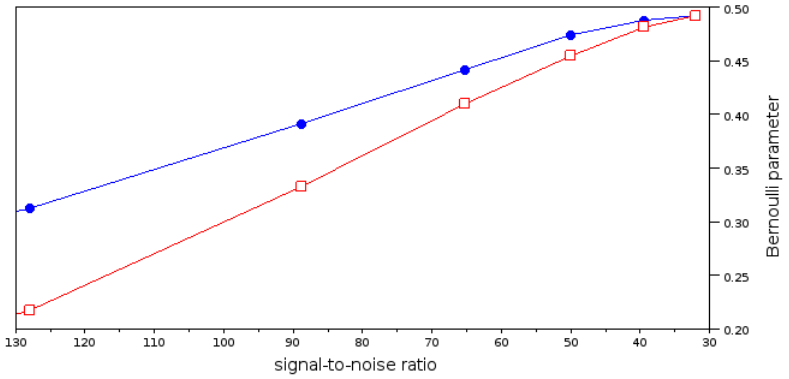


Figure 3.1: Bernoulli parameter with LLR (blue circles) and traces selection (red squares)

Saving Traces with Further Blocks. Up to now we only considered the first multiplication since not knowing k implies not knowing the input of the second multiplication (indeed $z^{(2)} = (a^{(1)} \cdot k \oplus a^{(2)}) \cdot k$). Nonetheless, re-writing this equality as

$$z^{(2)} = a^{(1)} \cdot k^2 \oplus a^{(2)} \cdot k,$$

we observe that $z^{(2)}$ also is a linear function of k since squaring is linear over GF(2). Denoting by $\mathbf{s}$ the matrix corresponding to the squaring operation, then

$$z^{(2)} = (a^{(1)} \cdot \mathbf{s} \oplus a^{(2)}) \cdot k.$$

Thus a linear relation can also be obtained from the second multiplication substituting $a^{(1)} \cdot \mathbf{s} \oplus a^{(2)}$ to a in Equation (3.1). And this is also true in the Hamming distance model since $z^{(1)} \oplus z^{(2)} = (a^{(1)} \cdot \mathbf{s} \oplus a^{(1)} \oplus a^{(2)}) \cdot k$. This observation is of great importance since it significantly improves the complexity of the attacks with a number of required traces divided by a factor two.

LPN and Linear Decoding Algorithms

There are many algorithms to recover the authentication key from noisy Hamming weight LSBs. In the case where more than n multiplications are observed, the attacker will obtain an over-defined linear system. In other words, the attacker will get redundant linear relations involving bits of the key k. Guessed LSBs extracted from leaking multiplication can thus be seen as forming a noisy codeword that encodes the authentication key k using the code defined by the linear relations of the form of Equation (3.1). Recovering the key is then equivalent to decoding the noisy codeword.

Learning Parities with Noise Algorithms. As mentioned in Chapter 2, the most efficient algorithms to solve the LPN problem are based on the Blum-Kalai-Wasserman algorithm [BKW03]. This algorithm tries to perform Gaussian elimination in a smart way but canceling many bits with one single XOR. The idea is to use many samples and XORing those that have many bits in common. However, this algorithm is exponential in the number of samples, time and memory of order $2^{O(k/\log k)}$ where n is the size of the secret values. This algorithm has been improved by Fouque and Levieil in [LF06] with a reduction of the constant in the exponent. In practice, it requires a huge number of samples but since here the size is relatively short $n = 128$, we could use such algorithms. However, since the noise involves a Bernoulli parameter p getting closer to $1/2$, it expects 2^{40} bytes of memory and 2^{34} queries when the SNR equals 128 (standard deviation σ equals 0.5), while it grows to 2^{241} bytes of memory and 2^{334} queries when

Table 3.2: Complexities of recovering the key with LLR and Equation (3.4), LPN and linear decoding according to the signal-to-noise ratio

SNR / *Method*	3200 C_s/C_t	800 C_s/C_t	200 C_s/C_t	128 C_s/C_t
LLR and Equation (3.4)	$2^8/2^{21}$	$2^8/2^{21}$	$2^8/2^{65}$	$2^8/2^{107}$
LPN (LF Algo)	$2^7/2^{21}$	$2^7/2^{23}$	$2^{32}/2^{34}$	$2^{48}/2^{50}$
Linear decoding	$2^6/2^6$	$2^6/2^7$	$2^8/2^{25}$	$2^9/2^{62}$

SNR equals 8. Lyubashevsky gave in [Lyu05] a variant of BKW with running time $2^{O(n/\log\log n)}$ for $n^{1+\varepsilon}$ samples. A further modification proposed more recently by Kirchner in [Kir11] achieves better runtime for small values of p. This algorithm runs in time $O(2^{\sqrt{n}})$ with $O(n)$ samples when $p = O(1/\sqrt{n})$.

Linear Decoding. Since inputs are not controlled by the attacker, the corresponding linear code is random. Decoding over random linear codes is known to be a hard problem (NP problem). The currently best algorithm that solves this problem is the one presented by Becker *et al.* in [BJMM12] which has complexity $O(2^{0.0494\ell})$ (where ℓ is the code length). Nevertheless, using such algorithm only makes sense if the noise is low enough to ensure that the actual key-codeword is the closest to the noisy word obtained by the attacker. Indeed if the noise is too high, then the channel capacity will decrease below the code rate and thus the closest codeword to the obtained noisy one may not be the one the attacker looks for. Using the binary symmetric channel model[5] we obtain that for a SNR of 128 the code length should be at least $\frac{128}{0.107} \approx 1200$ which would yield a complexity $2^{59.28}$ using [BJMM12]. Obviously the attacker has better using less than 1200 relations and test more than a single key candidate. To do so she will need a list-decoding algorithm. For cryptographic parameters (that is a key that can be very badly ranked), the only known solution is to see the linear code as a punctured Reed-Muller code and to use a Fast Walsh transform to obtain probabilities for each possible codeword. Since this technique has complexity $O(\ell 2^\ell)$ with ℓ the code dimension, it is not straightforwardly applicable here. We discuss in Section 3.2 how we can take profit of controlling inputs to use such decoding algorithm.

Complexities Evaluation

In this section, we built a system of equations from a new trick, that is the use of a single leakage bit. Then, we discussed methods to solve it involving step by step decoding (Equation (3.4)), LLR statistics and the existing tools: LPN and linear decoding. We now propose a comparison of the methods complexities through Table 3.2 both in terms of number of samples C_s and of computation time C_t.With respect to the LLR method combined with the step by step error removal (Equation (3.4)), the time complexity C_t includes not only the errors removal $C_n^{(e)}$ but also the linear system solving (a single inversion in n^3 and $C_n^{(e)} - 1$ matrix/vector products in n^2). As for the number of samples C_s, it is divided by two in all methods thanks to the smart use of the second GCM block $z^{(2)}$. We remark that for higher levels of SNR (at least until SNR = 200), linear decoding is the best method to choose both in terms of number of samples and time complexity. Afterwards, it depends on the number of available samples. Concretely, the more traces we have the less time we need.

[5]That is using the aforementioned Bernoulli parameter as error probability.

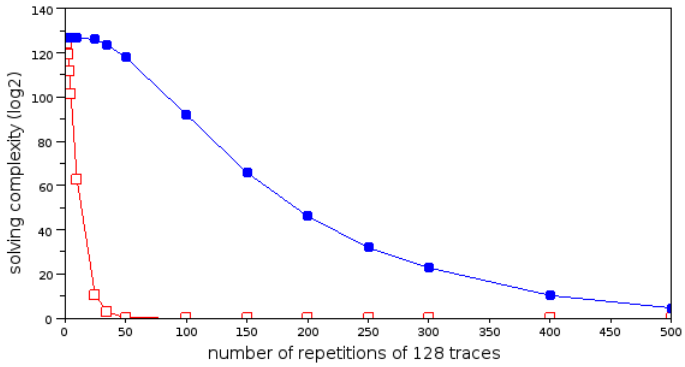


Figure 3.2: Solving complexities for several repetitions numbers with $\mathsf{SNR} = 32$ (blue, circles) and $\mathsf{SNR} = 2$ (red, squares)

3.2 Extension to Chosen Inputs

In this section, we investigate techniques that may be used to recover the key in the model where the attacker is able to control multiplication inputs. A first idea is that in such context averaging should be considered as obviously enabled[6] and thus measurement noise could be decreased by repeating inputs. A second idea is to structure the messages to make the system easier to solve. Eventually, a third idea is to take advantage of the AES-GCM algorithm and to exploit more than two multiplications. The following is dedicated to the discussion of these three ideas.

3.2.1 Averaging Traces

In the context where the attacker can monitor few multiplications with the same input, we can also consider another commonly used method which consists in averaging the traces. Theoretically, repeating the traces m times allows to divide the noise standard deviation by $\sqrt{m}$. Figure 3.2 gives the complexity of removing the errors (averaged from 10,000 tests computed from Equation (3.4)) according to the number of repetitions of 128 traces for two levels of SNR. Note that the full complexity of the attack also includes the system solving (a single inversion in n^3 and $C_n^{(e)} - 1$ matrix/vector products in n^2). Considering it, we can claim that with less than 2^{16} traces (*i.e.*,, 500 repetitions), the attacker can practically recover the key for an SNR equal to 2. In order to verify this theoretical result, we performed practical experiments.

Settings. We implemented the GHASH function on the Virtex-5 FPGA of a SASEBO board and acquired traces from an EM probe. We obtained 10^5 traces that we separated in two $5 \cdot 10^4$ trace sets (Set 1 and Set 2). We then built templates using Set 1 and a projection obtained using the same technique as in [DSV+14]. Afterwards we performed the first part of the attack (that is guessing parity bits of Hamming weights with the LLR technique) using this template. We then attacked both sets of traces.

Results. In Table 3.3 we provide the results we obtained. For a given number of averaging (*av.*) we report (*i*) the signal-to-noise ratio SNR, (*ii*) the simulated error rate obtained from this standard deviation (10^7 simulations) and (*iii*) the error rates obtained when applying the template to Set 1 and Set 2. First, we see that doubling the number of averaging roughly leads to a reduction of noise standard deviation by a factor of $\sqrt{2}$ as expected. Second, the attack performs better on the first set since it is the one that has been used to build templates. For Set

[6]Except maybe in pathological cases.

Table 3.3: Signal-to-noise ratios and error rates obtained from EM traces.

av.	SNR	error rates		
		simulation	Set 1	Set 2
1	8.347	n/c	n/c	n/c
2	19.32	n/c	n/c	n/c
3	28.32	0.5 - 2.26 10^{-3}	n/c	n/c
4	41.14	0.486	0.483	0.495
6	53.97	0.466	0.454	0.467
8	78.86	0.414	0.407	0.457
10	95.45	0.378	0.370	0.422
12	118.3	0.333	0.338	0.404

1, the error rates actually correspond to theoretical approximations based on the SNR. We also see that the error rates obtained for Set 2, while obviously deviating from expected values, are significantly decreasing with the number of averaging. Indeed, when averaging is possible, the obtained features show that an attack can be mount easily. As one can see, Table 3.3 does not contain data for error rates corresponding to less than 4 repetitions. This is due to the fact that the deviation from 0.5 is too small to be estimated using 50,000 traces. We did not managed to get more traces since experiments with higher levels of averaging confirm our predictions.

3.2.2 Structured Messages

In Section 3.1.2, we saw that recovering the key could be seen as a decoding problem. The difficulty arose from the fact that the linear code corresponding to our attack is random and has a high dimension (128). Assuming the attacker is now able to control inputs of the multiplication, she may choose the underlying code.

Choice of the Code

We now discuss the choice of the code for our dedicated problem.

List Decoding. First, an attacker aims at recovering the key. She has computing power and can enumerate many key candidates before finding the correct one. Such a feature means that a list decoding algorithm should be available for the chosen code. Moreover, the list size is not of the same order of magnitude that can be found in coding theory. Ideally, we would like to obtain a list of all key candidates ordered by probabilities of being the correct one. Obviously such a list cannot be created since its size would be 2^{128}. Nevertheless, using the key enumeration algorithm of [VGRS13], an attacker can enumerate keys from ordered lists of key chunks. If the linear code underlying the attack is a concatenated code then such algorithm can be used. Indeed, the corresponding matrix of the system would be a block diagonal matrix. Each block corresponds to a smaller linear code that may be fully decoded, that is the attacker obtains a list of all possible keys with the corresponding probabilities.

Soft Information. Second, since the noise may be high, we would like to take profit of the whole available information and not only consider obtained bits $\widehat{b_0}$. We illustrated in Section 3.1.2 the gain obtained when considering LLR statistics to take into account the relations reliabilities. Here, we would like a code which decoding algorithm can exploit such soft information.

Taking into account the aforementioned constraints, we chose a concatenating code of smaller random linear codes. The latter can efficiently be decoded using a Fast Walsh Transform. We thus aim at obtaining a matrix corresponding to the system $\mathcal{S}$ of the form:

$$\begin{pmatrix} \boxed{\mathcal{S}_0} & & \\ & \boxed{\mathcal{S}_1} & \\ & & \ddots \end{pmatrix} \cdot \begin{pmatrix} k \end{pmatrix} = \begin{pmatrix} \widehat{b_0} \\ \vdots \\ \widehat{b_t} \end{pmatrix}. \tag{3.5}$$

Generating Structured Inputs

To generate the inputs that yield a matrix similar to the one in Equation (3.5), the attacker has to consider the application:

$$\begin{aligned} \varphi &: a \mapsto (v_0, \ldots, v_{127}) \quad \text{with} \\ v_i &= \bigoplus_{0 \leqslant j \leqslant 127} \mathsf{lsb}_j \left(a \cdot \alpha^i \right), \quad \forall\ 0 \leqslant i \leqslant 127 \end{aligned}$$

that maps an input a to the corresponding vector of coefficients for the system $\mathcal{S}$. To generate the bloc $\mathcal{S}_c$, she chooses inputs in the kernel of $\varphi_{|\mathcal{I}_c}$ where indexes in $\mathcal{I}_c$ correspond to columns outside block $\mathcal{S}_c$. A basis of these kernels are efficiently computed using Gauss eliminations.

Simulations

To illustrate the method results, we give two graphs. The left one presents the averaged rank of the correct key among all the 2^{128} possible ones from the key chunks probabilities according to the signal-to-noise ratios for 256 samples (blue) and 1024 samples (red). The right one is a security graph [VGS13] which draws the evolution of the bounds of the correct key rank according to the number of samples for $\mathsf{SNR} = 512$.

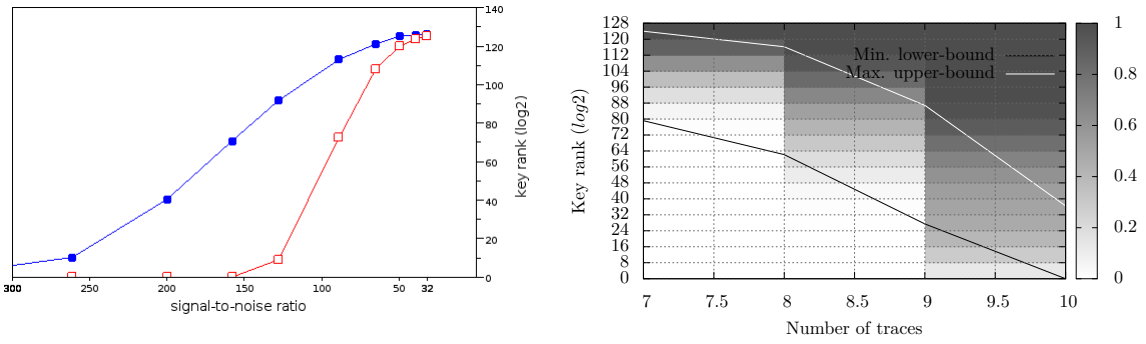


Figure 3.3: Key rank for 256 (blue, circles) and 1024 (red, squares) samples (left) and security graph for $\mathsf{SNR} = 128$ ($\sigma = 0.5$) (right)

3.2.3 Saving Traces

A second way, for the attacker, to take profit of the control she has on inputs, is to leverage on Ferguson observation [Fer05]. During the specification process of AES-GCM, Ferguson observed

that it was possible to obtain a tag that is linearly dependent on the authentication key k in the particular case where the polynomial corresponding to the tag only has non-zero coefficients in positions where the exponent of k is a power of two. This observation relies on the linearity of the squaring operation as mentioned in Section 3.1.2.

We saw that this observation allows to exploit the two first multiplications but if the attacker has the control on the inputs she can choose them to do more. Again, this trick can be used either in case of Hamming weight or Hamming distance. The only limitation is that the number of blocks to authenticate grows exponentially in the number of exploitable multiplications. The trade-off will depend on the available time for getting traces and on a potential limitation in the number of queries. To illustrate this, we show how an attacker can exploit three multiplications in a single trace. From Equation (2.1), we obtain the expression of the four first $z^{(i)}$'s when $a^{(2)}$ is set to zero:

$$\begin{aligned} z^{(1)} &= a^{(1)} \cdot k, \\ z^{(2)} &= a^{(1)} \cdot k^2, \\ z^{(3)} &= a^{(1)} \cdot k^3 \oplus a^{(3)} \cdot k, \\ z^{(4)} &= a^{(1)} \cdot k^4 \oplus a^{(3)} \cdot k^2 \oplus a^{(4)} \cdot k. \end{aligned}$$

We see that relations obtained from $z^{(1)}$, $z^{(2)}$ and $z^{(4)}$ only involve power-of-two of k which means that the relation is a linear function of k. For instance $z^{(4)} = (a^{(1)} \cdot \boldsymbol{s}^2 \oplus a^{(3)} \cdot \boldsymbol{s} \oplus a^{(4)}) \cdot k$, and because she knows $\boldsymbol{s}$ and can choose $a^{(4)}$, the attacker can obtain the input of its choice for the fourth multiplication.

3.3 Adaptation to Fresh Re-keying

It can be checked that the attack described in Section 3.1.2 does not apply to this modular multiplication. If we write the equation involving the LSB b_0 of the Hamming weight of the result, we obtain:

$$\mathsf{lsb}_0 \left(\mathsf{HW} \left[\left(\bigoplus_{0 \leqslant i \leqslant m-1} a_i \right) \otimes \left(\bigoplus_{0 \leqslant j \leqslant m-1} k_j \right) \right] \right) = b_0$$

with $\otimes$ representing the modular multiplication in $\mathsf{GF}(2^8)$. As we can see, only the sum of all the key bytes can be recovered if the attack is successful. However, no individual key bit can be determined. If we extended the attack to more leakage bits, we could (at most) successively recover all the bits of the Hamming weight of the master key.

However, looking further in the fresh re-keying protocol may make the attack still practicable. Until now, we have made the assumption that the multiplication output was stored in a 128-bit register, which essentially corresponds to a hardware implementation and is the worst case from the attacker point of view. If we switch to a software implementation (*e.g.,* running on a w-bit architecture), then, the attacker can target the manipulation of w-bit sub-parts of the refresh key k' which puts him in a more favorable context. By moreover assuming that k' is used as a secret parameter of a block cipher like AES (as proposed in [MSGR10]), the attacker can exploit information leakage when the byte-coordinates of k' are manipulated separately. In this case, the LSB of the Hamming weight of the first byte k'_0 can be written as follows:

$$\mathsf{lsb}_0 \left(\mathsf{HW} \left[\bigoplus_{0 \leqslant i \leqslant m-1} a_i \otimes k_i \right] \right) = b_0$$

Table 3.4: Bernoulli parameter p for the LSB under several SNRs

SNR	8	2	0.5	0.125
p	$6.3\ 10^{-5}$	0.46	0.31	0.5-0.0046

and we finally obtain a linear equation on the key bits. In this case, if the level of noise is the same than for 128-bit data (*e.g.,* if the data is still stored in 128-bit registers), then the error probability p will be the same as the one reported in Table 3.1. If the data is, as assumed, stored in 8-bit registers, then we can consider that the SNR is equivalent. In this case, the standard deviation on 8 bits will be four times lower than its equivalent on 128 bits (due to the difference in the signal variance) and the probability of error will be much lower as depicted in Table 3.4.

Chapter 4

Power Analysis on Multiplications Based on MSB

In this chapter, we describe a new algorithm for solving the hidden multiplier problem, in which we use the most significant bits of the Hamming weight instead of the single least significant bit. We show that much smaller values of SNR can then be tolerated ($\mathsf{SNR} \simeq 8$), which increases the practicability of the attack. The main idea relies on only keeping the observations with small or high Hamming weights. Namely, if $\mathsf{HW}(a \cdot k)$ is close to 0 (resp. n), this means that most of the n bits of $a \cdot k$ are equal to 0 (resp. 1). This can be written as a system of n equations over the bits of k, all equal to 0 (resp. 1), where some of the equations are erroneous. Hence, this filtering provides us an instance of the LPN problem. To solve it, we use BKW style algorithms [BKW03]. Their main drawback is the huge samples requirement that makes them unpractical for side-channel attacks. Thus, in this chapter, we propose some improvements to reduce the query complexity using Shamir-Schroeppel [SS79], the variant proposed by Howgrave-Graham and Joux in [HGJ10] and the secret-error switching lemma [Kir11, ACPS09] to further reduce the time complexity. Afterwards, we describe another attack when the messages a^i can be chosen. In that case, the attack becomes much more efficient. We also attack the multiplication-based fresh re-keying scheme proposed in [MSGR10] to defeat side-channel cryptanalysis. Whereas the re-keying primitive (itself) is not vulnerable to the technique used in [BFG14], we demonstrate that our attack enables to recover the secret key very efficiently.

Contents

4.1 Attack on Known Inputs

In this section, we describe the second side-channel attack on the result of the multiplication in $\mathsf{GF}(2^n)$, which benefits from being weakly impacted by the observation noise. As in the previous chapter, we aim at recovering the n-bit secret key k from a sequence of t queries $(a^i, \mathsf{HW}(k \cdot a^i) + \varepsilon^i)_{0 \leqslant i < t}$ where the a^i are drawn uniformly in $\mathsf{GF}(2^n)$ and the ε^i are drawn from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

The cornerstone of the attack is to filter the collected measurements to keep only the lowest and the highest Hamming weights. Then we assume that for each lower (resp. higher) Hamming weight, the multiplication result is exactly n bits of zeros (resp. ones). As a consequence, each filtered observation of $z^i = a^i \cdot k$ gives n equations each with some error probability p. In our context, the equations correspond to the row-by-column scalar products in Equation (2.2) and the binary error associated to the i^{th} equation is denoted by e_i, with $\mathbb{P}[e_i = 1] = p$.

Therefore, given t messages and corresponding measurements, we get an instance of the $\mathsf{LPN}(n, n \cdot t, p)$ problem that we can solve using techniques described in Section 4.1.2. To correctly scale the latter techniques, we need to know the error probability p with good precision. In the next section we show how to compute p from the filtering threshold and the measurement noise σ in Equation (2.6).

4.1.1 Filtering

We show hereafter how we filter the lowest and highest leakage and we compute the error probabilities of our final set of equations. In order to catch the extreme Hamming weight values of the multiplications results, we choose a threshold real value λ and we filter all the observations below $n/2 - \lambda s$ and above $n/2 + \lambda s$, with $s = \sqrt{n}/2$ the standard deviation of the leakage deterministic part (here the Hamming weight). In the first case, we assume that all the bits of the multiplication result are zeros and in the second case we assume that they are all set to one. In both cases, we get n linear equations on the key bits, each having the same error probability p.

We first compute the proportion of filtered acquisitions. Let $z = k \cdot a$ be the result of a finite field multiplication; since $z \sim \mathcal{U}(\mathsf{GF}(2)^n)$, we deduce $\mathsf{HW}(z) \sim \mathcal{B}(n, 1/2)$. Moreover since

$$\mathcal{L}(z) = \mathsf{HW}(z) + \varepsilon \ ,$$

with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, we obtain that the pdf h of $\mathcal{L}(z)$ is defined over $\mathbb{R}$ by:

$$h(x) = 2^{-n} \sum_{y=0}^{n} \binom{n}{y} \phi_{y,\sigma}(x) \ . \tag{4.1}$$

Since our filtering rejects the observations with leakage $\mathcal{L}(z)$ between $n/2 - \lambda s$ and $n/2 + \lambda s$ for some parameter λ, the proportion of filtered acquisition $F(\lambda)$ is then:

$$\forall \lambda \in \mathbb{R}, \quad F(\lambda) = 1 - 2^{-n} \sum_{y=0}^{n} \binom{n}{y} \int_{n/2-\lambda s}^{n/2+\lambda s} \phi_{y,\sigma}(t) dt \ . \tag{4.2}$$

After filtering, our attack consists in assuming that the n bits of z are all zeros if $\mathcal{L}(z) < n/2 - \lambda s$, and are all ones if $\mathcal{L}(z) > n/2 + \lambda s$. Therefore, in the first case out of the n equations, $\mathsf{HW}(z)$ equations are erroneous, whereas in the second case $n - \mathsf{HW}(z)$ equations are erroneous. In the first case, this corresponds to an error probability $\mathsf{HW}(z)/n$, while in the second case,

Table 4.1: Error probability p and λ w.r.t. the filtering proportion $F(\lambda)$ and the SNR

$\log_2(1/F(\lambda))$	30	25	20	15	10	5
$\mathsf{SNR} = 128$, $\sigma = 0.5$						
λ	6.00	5.46	4.85	4.15	3.29	2.16
p	0.23	0.25	0.28	0.31	0.34	0.39
$\mathsf{SNR} = 8$, $\sigma = 2$						
λ	6.37	5.79	5.14	4.39	3.48	2.28
p	0.25	0.27	0.29	0.32	0.35	0.40
$\mathsf{SNR} = 2$, $\sigma = 4$						
λ	7.42	6.73	5.97	5.09	4.03	2.64
p	0.28	0.30	0.32	0.34	0.37	0.41
$\mathsf{SNR} = 0.5$, $\sigma = 8$						
λ	10.57	9.58	8.48	7.21	5.71	3.73
p	0.34	0.36	0.37	0.39	0.41	0.44

this corresponds to an error probability $1 - \mathsf{HW}(z)/n$. On average, over filtered observations, we obtain an error probability equal to:

$$p(\lambda) = \frac{1}{F(\lambda)} \sum_{y=0}^{n} \frac{\binom{n}{y}}{2^n} \left(\frac{y}{n} \int_{-\infty}^{n/2-\lambda s} \phi_{y,\sigma}(t)dt + \left(1 - \frac{y}{n}\right) \int_{n/2+\lambda s}^{+\infty} \phi_{y,\sigma}(t)dt \right). \tag{4.3}$$

This error probability $p(\lambda)$ (or p for short) is a crucial parameter as it gives the error probability in the LPN problem. Our goal is to minimize p in order to minimize the complexity of solving the LPN problem. This can be done by increasing the filtering threshold λ; however a larger λ implies that a larger number of observations must be obtained initially. Therefore a trade-off must be found between the error probability p in the LPN problem and the proportion $F(\lambda)$ of filtered observations.

The main advantage of this attack is that this error probability p is quite insensitive to the noise σ in the observations, as illustrated in Table 4.1. For $n = 128$ and for various values of σ, we provide the corresponding filtering threshold λ that leads to a filtering probability $F(\lambda)$, expressed with $\log_2 1/F(\lambda)$; we then give the corresponding error probability p. For example, for $\mathsf{SNR} = 128$, with $\lambda = 6.00$, we get a filtering probability $F(\lambda) = 2^{-30}$, which means that on average, 2^{30} observations are required to get $n = 128$ equations for the LPN problem; in that case, the error probability for the LPN problem is $p = 0.23$. We see that this error probability does not grow too fast as SNR decreases, as we get $p = 0.25$ for $\mathsf{SNR} = 8$, $p = 0.28$ for $\mathsf{SNR} = 2$, and $p = 0.34$ for $\mathsf{SNR} = 0.5$.

Study in the general case

For completeness, we exhibit hereafter the expressions of the probabilities $F(\lambda)$ and $p(\lambda)$ when the leakage satisfies Equation (2.6) for another function than $\mathsf{HW}(\cdot)$. If we relax the Hamming weight assumption but still assume that the noise is independent and additive, we get the following natural generalization of Equation (2.6):

$$\mathcal{L}(z) = \varphi(z) + \varepsilon \ , \tag{4.4}$$

where $\varphi(z) \doteq \mathbb{E}(\mathcal{L}(Z) \mid Z = z)$ and $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. This leads to the following generalization of Equation (4.2):

$$\forall \lambda \in \mathbb{R}, \quad F(\lambda) = 1 - \sum_{y \in \mathrm{Im}(\varphi)} \mathbb{P}(\varphi(Z) = y) \int_{-\lambda s}^{\lambda s} \phi_{y,\sigma}(t + \mu) dt \ , \tag{4.5}$$

where μ and s respectively denote the mean and the standard deviation of $\varphi(Z)$. Analogously, we get:

$$p(\lambda) = \frac{1}{F(\lambda)} \sum_{y=0}^{n} \frac{\binom{n}{y}}{2^n} \left(\frac{y}{n} \int_{-\infty}^{\lambda s} g_y(t + \mu) dt + \left(1 - \frac{y}{n}\right) \int_{\lambda s}^{+\infty} g_y(t + \mu) dt \right) \ , \tag{4.6}$$

where, for every y, the pdf $g_{\mathcal{L}|\mathsf{HW}(Z)=y}$ is defined by:

$$g_y(\ell) = \binom{n}{y}^{-1} \sum_{z \in \mathsf{HW}^{-1}(y)} \phi_{\varphi(z),\sigma}(\ell) \ . \tag{4.7}$$

In the case $\varphi = \mathsf{HW}$ (*i.e.,* when the device leaks perfectly in the Hamming weight model), it can be checked that g_y is simply the pdf of $\mathcal{N}(\mathsf{HW}(y), \sigma^2)$, otherwise it is a Gaussian mixture. In Section 4.4, we will approximate it by a Gaussian pdf with mean $\mathbb{E}(\mathcal{L}(Z) \mid \mathsf{HW}(Z) = y)$ and standard deviation $\sqrt{\mathrm{Var}(\mathcal{L}(Z) \mid \mathsf{HW}(Z) = y)}$.

4.1.2 Solving the LPN Problem

Numerous algorithms for solving LPN are known in the literature; a good survey is given by Pietrzak in [Pie12]. They generally require a huge number of LPN equations. However in our context, these equations come from side-channel acquisitions and thus remain in a rather scarce number. A well-known result of Lyubashevsky reduces the sample complexity, but its limitations on the noise render it inapplicable to our problem [Lyu05]. In this section, we summarize the ideas we set-up for solving the LPN problem with a reduced number of samples and under reasonable levels of noise.

We take the point of view of an attacker: she has a limited quantity of side-channel information thus a limited number of initial LPN samples. She also has a limited computing power and (most importantly) memory. She has two goals: (*i*) she wants to make sure that the attack will indeed be feasible in theory (this depends on the final number of sparse equations), thus, she must compute it as exactly as possible (she cannot afford to miss one bit of complexity in the computations) and (*ii*) she has reasonable but limited resources and wants to make the attack as efficient as possible.

Algorithm sketch

The main parameter of the algorithm is the initial bias: it determines the number of linear combinations steps we will be able to do before the final bias explodes (see Section 2.3). In our case, we have 128-bit equations which must be reduced to around 32 bits to finally recover the secret key. Thus, for the bias to remain reasonable, we choose to remove 32 bits at each step with 3 reductions, thus $2^3 = 8$ linear combinations. We look for small-weight linear combinations of initial equations that have their most significant bits cancelled. Unfortunately, there are not enough initial LPN equations to use BKW or LF1 algorithms directly (they do not remove enough bits per iteration).

We thus first (rather artificially) square the number ν of LPN samples: for all elements a^i in the initial set, with error probability p (bias $\delta = 1 - 2p$), we build the set $(a^{i,j})_{j \neq i} \doteq (a^i \oplus a^j)_{i,j}$ of what we call amplified equations. Therefore, we have only 2 reductions left. However, on the one hand, BKW-like algorithms will still not find enough reduced equations. On the other hand, exhaustively looking for reduced equations among all linear combinations of at most 4 (corresponding to 2 reductions) amplified equations would not be very efficient. Consequently, we apply two steps of a generalized birthday paradox-like algorithm [Wag02].

Then, assume that we obtain w-bits reduced equations. Once enough equations are found (this depends on the final bias of the equations, which is δ^8), we can directly apply a Walsh-Hadamard transform (WHT) to recover the w less significant bits of the secret if the attacker memory is greater than 2^w w-bits words. If we can only obtain equations reduced to $w' > w$ bits, we can simply guess the $w' - w$ bits of the secret and do a WHT on the last w bits. In this case, the search space can be reduced using the error/secret switching idea at the very beginning of the algorithm.

The algorithm steps as well as its time and space complexities are analyzed in details in [BCF+15b]. From a practical perspective, the optimal choice depends on several parameters: number of traces, filtering ratio, level of noise, available memory, computing power. Several trade-offs are thus available to the attacker. The most obvious one is to trade side-channel measurements against computing needs. Using more traces either makes it possible to reduce the bias of the selected equations, or increases their number, reducing the reduction time (birthday paradox phase). In a nutshell, the more traces are available, the better.

Given a fixed number of traces (order of magnitude 2^{20} to 2^{24}), the attacker fixes the filtering threshold λ. Increasing λ reduces the bias of the selected equations. As a consequence, less reduced equations are required for the WHT to correctly find w bits of the secret. Nonetheless, increasing λ also reduces the number of initial equations and thus makes the birthday paradox part of the algorithm slower.

Concerning the reduction phase, it is well known that balancing the two phases of the generalized birthday paradox is the best way to reduce its complexity.

Finally doubling the memory makes it possible to recover one bit more with the WHT, while slightly more than doubling its time complexity: we fill the table with equations that are 1 bit less reduced, halving the time needed by the birthday paradox phase.

4.1.3 Comparison with the LSB-Based Attack

Compared to the LSB-based attack, this new attack performs better except for high SNRs, such as $\mathsf{SNR} = 128$, and when the number of available queries is very limited by the context. Indeed, for $\mathsf{SNR} = 128$, the previous attack requires only 128 observations to get 128 equations with error probability 0.31 whereas this one requires 2^{15} observations to achieve the same error probability. In the other contexts (*i.e.,* for higher levels of noise), the LSB-based attack faces strong limitations. Concretely, recovering the secret key becomes very hard if the inputs are not chosen. On the contrary, since the MSB-based attack benefits from being quite insensitive to noise, it stays successful even for higher noise levels.

4.2 Extension to Chosen Inputs

In this section, we present a key-recovery techniques which can be applied when the attacker is able to control the public multiplication operands a_i. It is based on comparing the leakage for related inputs.

4.2.1 Comparing Leaks

In the so-called *chosen message model*, the attacker chooses ν messages $(a^i)_{0\leqslant i<\nu}$ in $\mathsf{GF}(2^n)$ and gets the corresponding leakages $\mathcal{L}(k \cdot a^i)$, where

$$\mathcal{L}(k \cdot a^i) = \mathsf{HW}(k \cdot a^i) + \varepsilon$$

From the underlying associative property of the field $\mathsf{GF}(2^n)$, we remark[1] that the relation $(2 \cdot a^i) \cdot k = 2 \cdot (a^i \cdot k)$ stands for every query a^i. If the most significant bit of $a^i \cdot k$ is zero, then the latter relation implies that the bits of $a^i \cdot k$ are simply shifted when computing $2 \cdot (a^i \cdot k)$ which results in $\mathsf{HW}((2 \cdot a^i) \cdot k) = \mathsf{HW}(a^i \cdot k)$. However, if the most significant bit of $a^i \cdot k$ is one, then the bits are also shifted but the result is summed with the constant value 23. The latter corresponds to the decimal representation of the binary coefficients of the non-leading monomials of the polynomial $x^{128} + x^7 + x^2 + x + 1$ involved in the representation of the field $\mathsf{GF}(2^{128})$ in AES-GCM. In this case, the two Hamming weight values $\mathsf{HW}((2 \cdot a^i) \cdot k)$ and $\mathsf{HW}(a^i \cdot k)$ are necessarily different. Indeed, the bits are shifted, the less significant bit is set to one and the bits of $(a^i \cdot k)$ at positions 0, 1 and 6 are flipped. Thus, the absolute value of the difference between both Hamming weights is either equal to 3 with probability 1/4 or to 1 with probability 3/4.

Without noise, we can perfectly distinguish whether both Hamming weight values are equal or not, and thus, we can get knowledge of the most significant bit of $a^i \cdot k$. Repeating the experiment for every power of two until 2^{128}, that is with 128 queries, gives us the knowledge of every bit of the multiplication result and thus, the recovery of k. In presence of noise, the recovery is no longer straightforward. To decide whether the noisy Hamming weights are equal or different, we fix a threshold τ depending on the SNR. Namely, if the distance $|\mathcal{L}((2 \cdot a^i) \cdot k) - \mathcal{L}(a^i \cdot k)|$ is greater than τs where s is the signal standard deviation (here the standard deviation of $\mathsf{HW}(Z)$, say $\sqrt{n}/2$), then we decide that $\mathsf{HW}((2 \cdot a^i) \cdot k) \neq \mathsf{HW}(a^i \cdot k)$ and thus that the most significant bit of $(a^i \cdot k)$ equals one. The type I error probability p_{I} associated to this decision (namely the probability of deciding that the Hamming weight values are different while they are equal $p_{\mathrm{dif|eq}}$) satisfies the equality:

$$\begin{aligned} p_{\mathrm{I}} &= \mathbb{P}\left[|\mathcal{L}((2 \cdot a^i) \cdot k) - \mathcal{L}(a^i \cdot k)| > \tau s \mid \mathsf{HW}((2 \cdot a^i) \cdot k) = \mathsf{HW}(a^i \cdot k)\right] \\ &= \mathbb{P}\left[|\varepsilon^{i+1} - \varepsilon^i| > \tau s\right] \\ &= 1 - \int_{-\tau s}^{\tau s} \phi_{\sigma\sqrt{2}}(u)du. \end{aligned}$$

Similarly, the type II error probability p_{II} (of deciding that the Hamming weight values are equal when they are different) satisfies the equality:

$$p_{\mathrm{II}} = \frac{3}{8}\left(\int_{-\tau s-1}^{\tau s-1} \phi_{\sigma\sqrt{2}}(u)du + \int_{-\tau s+1}^{\tau s+1} \phi_{\sigma\sqrt{2}}(u)du\right) + \frac{1}{8}\left(\int_{-\tau s-3}^{\tau s-3} \phi_{\sigma\sqrt{2}}(u)du + \int_{-\tau s+3}^{\tau s+3} \phi_{\sigma\sqrt{2}}(u)du\right).$$

Since, the key bits are all assumed to be balanced between one and zero, the probability of error p for each key bit is equal to $\frac{1}{2}(p_{\mathrm{I}} + p_{\mathrm{II}})$. Table 4.2 gives the thresholds τ which minimize the error probability for different values of standard deviations.

[1] We can simply choose a^i equal to 1.

Table 4.2: Optimal threshold and probability of deciding correctly w.r.t. the SNR

SNR	128	8	2	0.5
σ	0.5	2	4	8
τ	0.094	0.171	0.301	0.536
p	0.003	0.27	0.39	0.46

Note that we did not consider so far the bias induced by the recovery of the less significant bits (whose values have been altered by previous squarings) but in practice, we checked that it is negligible and does not change the numeric values of p.

Comparing to Table 4.1, the error probabilities in Table 4.2 are much more advantageous since we only need 129 queries. If we are not limited in the number of queries, we can even average the traces to decrease the noise standard deviation and thus improve the success rate. Another improvement is to correlate not only two consecutive powers of 2 but also non-consecutive ones (*e.g.,* 2^j and 2^{j+2}). Without noise, we do not get more information but in presence of noise, we can improve the probability of deciding correctly.

4.2.2 Key Recovery

With the method described above, we only get 128 different linear equations in the key bits. Thus, we cannot use an LPN solving algorithm to recover the secret key in presence of errors. However, since we can average the measurements to decrease the number of errors, we can significantly reduce the level of noise in order to remove the errors almost completely. For instance, with an SNR of 128 (which can also be achieved from an SNR of 2 and 64 repetitions), we get an average of $128 * 0.003 = 0.384$ errors. Solving the system without error is straightforward when we use the powers of two since we directly have the key bits. Thus, inverting all the second members of the equations one-by-one to remove a single error, leads to a global complexity of 2^7 key verifications. This complexity is easily achievable and is completely reasonable to recover a 128-bit key.

4.3 Adaptation to Fresh Re-keying

It may be checked that the attack described in Section 4.1 applies against the multiplication specified by Equation (2.4) similarly as for the multiplication in Equation (2.2). Indeed, the matrix-vector product defined in Equation (2.4) over $\mathsf{GF}(2^8)$ can be rewritten over $\mathsf{GF}(2)$ expressing each bit of k' as a linear combination of the bits of k with coefficients being themselves linear combinations of the bits of $a \in \mathsf{GF}(2)^{128}$. Eventually, exactly like in previous section, for ν filtered messages the attack leads to an instance of the $\mathsf{LPN}(128, 128\nu, p)$ problem.[2] Actually, looking further in the fresh re-keying protocol, we can improve the attack by taking advantage of the context in which the fresh re-keying is used.

Making the same assumption as in Chapter 3, according to which we switch to a software implementation for the block cipher, the attacker can exploit information leakage when the byte-coordinates of the session key k' are manipulated separately, as in the first round of the AES. Observing the manipulation of each of the sixteen 8-bit chunks separately gives, for a same filtering ratio, a much lower error probability on the equations that what was achieved

[2] As observed in Chapter 3, the LSB-based attack does not directly apply to the multiplication specified by Equation (2.4), essentially because of the circulant property of the matrix.

$\log_2(1/F(\lambda))$	10	5	4	3	2	1
$\mathsf{SNR} = 128$, $\sigma = 0.125$						
λ	2.93	2.15	2.02	1.47	1.33	0.71
p	$2.8 \cdot 10^{-19}$	$9.4 \cdot 10^{-2}$	0.11	0.17	0.21	0.28
$\mathsf{SNR} = 8$, $\sigma = 0.5$						
λ	3.25	2.26	1.97	1.63	1.24	0.74
p	$5.9 \cdot 10^{-3}$	0.10	0.14	0.18	0.23	0.29
$\mathsf{SNR} = 2$, $\sigma = 1$						
λ	3.88	2.62	2.28	1.89	1.42	0.83
p	$6.5 \cdot 10^{-2}$	0.16	0.19	0.22	0.26	0.32
$\mathsf{SNR} = 0.5$, $\sigma = 2$						
λ	5.66	3.73	3.22	2.66	1.99	1.17
p	0.17	0.25	0.28	0.30	0.33	0.37

Table 4.3: Error probability p according to the proportion of filtered acquisitions $F(\lambda)$ when $\mathsf{SNR} = 128$, $\mathsf{SNR} = 8$, $\mathsf{SNR} = 2$ and $\mathsf{SNR} = 0.5$

in the previous (hardware) context. This can be explained by the fact that exhibiting extreme Hamming weight values is obviously much more easier on 8 bits than on 128 bits. For instance, filtering one observation over 2^{10} (*i.e.,* $F(\lambda) = 2^{-10}$) with an SNR equal to 2, results in an error probability of $p = 0.28$ for $n = 128$ and $p = 0.065$ for $n = 8$, that is more than four times less. Table 4.3 gives the error probability p according to the proportion of filtered acquisitions $F(\lambda)$ for $\mathsf{SNR} = 128$, $\mathsf{SNR} = 8$, $\mathsf{SNR} = 2$ and $\mathsf{SNR} = 0.5$ (as in Table 4.1) and $n = 8$. This confirms on different parameters that with much fewer observations, we have smaller error probabilities. Therefore, even for $F(\lambda) = 0.5$ (*i.e.,* we only filter one observation over two), the system can be solved to recover the 128-bit key. Furthermore, it is worth noting that this new attack on an AES using a one-time key allows us to recover the master key without observing any leakage in the fresh re-keying algorithm.

4.4 Practical Experiments

We showed in previous sections how to mount efficient side-channel attacks on finite-field multiplication over 128-bit data in different scenarios according to the attacker capabilities. In order to verify the truthfulness of our leakage assumptions, we have mounted a few of these attacks in practice and made some simulations. Namely, we implemented both the AES-GCM multiplication and the fresh re-keying protocol on an ATMega328p and have measured the leakage thanks to the ChipWhisperer kit [OC14]. We also obtained the 100.000 traces of 128-bit multiplication corresponding to the EM radiations of an FPGA implementation on the Virtex 5 of a SASEBO board.

We first illustrate the behavior of the leakage we obtained on the ATMega328p. Then we present experimental confirmations that the filtering step actually behaves as expected. With this result in mind we report simulated attacks on 96-bit multiplication and expected complexities for 128-bit attacks. Afterwards, we present the attack on chosen inputs using the leakage obtained with the ATMega328p. We finally show how efficient is the attack on fresh re-keying when the attacker is able to exploit 8-bit leakages of the first round of AES.

4.4.1 ATMega328p Leakage Behavior

The aim of these experiments is to be easily reproducible since all the material required to make the measurements is easily accessible. Since we are in software on an 8-bit implementation, we simulate a 128-bit leakage by summing the intermediate leakage on 8-bit parts of the result[3]. We randomly generated $100,000$ vectors $a \in \mathsf{GF}(2)^{128}$ and, for a fixed key k, we measured the leakage during the processing of $z = a \cdot k$ as specified in AES-GCM (see Equation (2.2)). Each measurement was composed of $4,992$ points among which we detected 16 points of interest by following a T-test approach as *e.g.,* described in [GJJR11]. We afterwards verified that these points corresponded to the manipulation of the byte-coordinates $Z[i]$ of z after the multiplication processing.

In the top of Figure 4.1, we plot for each $i \in [1..16]$ the distribution of our estimates of the mean of the leakage $\mathcal{L}(Z[i])$ knowing either $Z[i] = z \in \mathsf{GF}(2)^8$ (left-hand figure) or $\mathsf{HW}(Z[i]) = y \in [0..8]$ (right-hand figure). First, it may be observed that all the byte-coordinates, except the first one, leak quite similarly. For each $i \in [1..16]$, we denote by $g_{\mathrm{Id},i}(z)$ the function $z \mapsto \mathbb{E}\left(\mathcal{L}(Z[i]) \mid Z[i] = z\right)$ and by $g_{\mathrm{HW},i}(y)$ the function $y \mapsto \mathbb{E}\left(\mathcal{L}(Z[i]) \mid \mathsf{HW}(Z[i]) = y\right)$. The average mean and standard deviation of the functions $g_{\mathrm{ID},i}$ are -0.0301 and 0.0051 respectively. They are $-0,0291$ and $0,0092$ for the functions $g_{\mathrm{HW},i}$. While the left-hand figure shows that the distributions of values differ from normal distributions, the right-hand figure exhibits a strong dependency between them and the distribution of the Hamming weight values of $Z[i]$. This shows that our implementation is a good target for our attack which requires that the deterministic part of the leakage monotonously depends on the Hamming weight of the manipulated data. Eventually, we plot in the bottom-left figure an estimate (with kernel methods) of the distribution of the values $\mathbb{E}\left(\mathcal{L}(Z)\right) \mid \mathsf{HW}(Z) = y)$ when y ranges in $[0..128]$ and $\mathcal{L}(Z) \doteq \sum_{i=1}^{16} \mathcal{L}(Z[i])$. Once again, the distribution is a not a perfect binomial one, but the figure shows that the deterministic part of the leakage monotonously depends on the Hamming weight of the manipulated data. The mean and the standard deviation of the plotted distribution are -0.1781 and $0,2392$ respectively. For completeness, we also plot in the bottom-right of Figure 4.1 the distribution of the leakage values (after combining the 16 point of interest): the distribution looks very close to a Gaussian one, with mean $-0,4823$ and standard deviation 0.0629.

[3]We could have chosen to adapt our results to target the 8-bit chunks directly, which would correspond to applying our attack to a software implementation of AES-GCM. Our purpose was however to test the practical soundness of our theoretical analyses; we hence chose to artificially build a 128-bit leakage. The application of our attack against 8-bit chunks is the purpose of Section 4.4.4 where it is shown that this situation is much more favorable to the attacker.

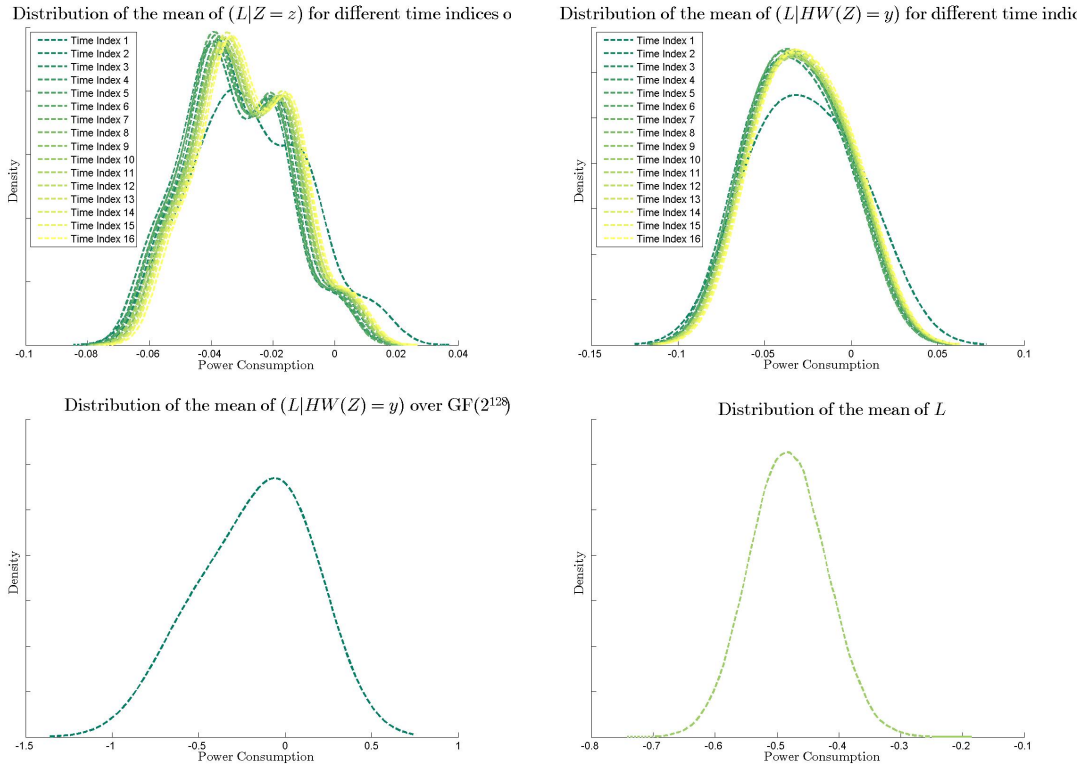


Figure 4.1: Behaviour of the leakage w.r.t. the manipulated data Z

4.4.2 Attack on the AES-GCM Multiplication's Output with Known Inputs

Experiments on Filtering

In this section, we explain our filtering, as described in Section 4.1, for different scenarios: in software and in hardware, and with a 128-bit key or a 96-bit key. We filtered the observations as described in Section 4.1.

ATMega328p (128-bit). In this context, the leakage $\mathcal{L}(Z)$ is built by summing the sixteen leakages $\mathcal{L}(Z[i])$, with $i \in [1..16]$. Theoretically, summing the sixteen intermediate Hamming weight values gives us exactly the Hamming weight value of the multiplication result. And summing the sixteen noise of standard deviation σ_8 results in a Gaussian noise of standard deviation $\sigma_{128} = 4 \cdot \sigma_8$. In practice, we get an SNR of 8.21 on the 128-bit simulated leakage.

In Table 4.4 we provide the experimental bounds λ_{exp} and error probabilities p_{exp} corresponding to a few levels of filtering. We also indicate the theoretical estimates λ_{the} and p_{the} obtained by applying formulas (Equation (4.5)) and (Equation (4.6)) to the template we obtained using the same set of traces. As it can be observed, the theoretical estimates are very close to the ones obtained experimentally (which validates our theoretical analysis, even for non Hamming weight model).

Remark 1. *It must be noticed that a* SNR *equal to* 8.21 *in our experiments (with a noise standard deviation* 0.0206*) corresponds to a noise with standard deviation* $\sigma = \sqrt{32/8.21} = 1.97$ *in the theoretical Hamming weight model over* 128*-bit data.*

Table 4.4: Experimental and theoretical parameters corresponding to filtering proportion $F(\lambda)$ on the ATmega for 128-bit AES-GCM

SNR $= 8.21$, $\sigma = 0.0206$							
$\log_2(1/F(\lambda))$	14	12	10	8	6	4	2
λ_{exp}	4.37	3.96	3.49	3.05	2.54	1.97	1.22
p_{exp}	0.383	0.386	0.393	0.407	0.420	0.434	0.452
λ_{the}	4.27	3.90	3.51	3.08	2.59	2.00	1.24
p_{the}	0.381	0.390	0.399	0.409	0.421	0.435	0.453

Table 4.5: Error probabilities obtained from real traces.

λ	0.906	1.270	1.645	2.022	2.409	2.794	3.165	3.847
p_{the}	0.442	0.431	0.419	0.407	0.395	0.382	0.369	0.357
p_{exp}	0.441	0.430	0.418	0.405	0.392	0.379	0.370	0.361

Virtex 5 (128-bit). We additionally performed filtering on the traces from [BFG14] obtained from an FPGA implementation of GCM. Hereafter we provide theoretical (p_{the}) and experimental (p_{exp}) error probabilities for different values of the filtering parameter λ (Table 4.5). It must be noticed that experimental results correspond to expectations. The largest deviation (for $\lambda = 3.847$) is due to the fact that only 20 traces were kept.

Remark 2. *It must be noticed that, surprisingly, we also obtained an* SNR *equal to* 8.21 *in FPGA experiments but corresponding to a noise standard deviation of* 7.11.

ATMega328p (96-bit). As in the 128-bit case, the 96-bit leakage is simulated by summing the twelve intermediate 8-bit leakage of the multiplication result. Table 4.6 gives the bounds q and the error probabilities p corresponding to some levels of filtering.

Remark 3. *A* SNR *equal to* 8.7073 *in our experiments (with a noise standard deviation* 0.0173*) corresponds to a noise with standard deviation* $\sqrt{24/8.7073} = 1.66$ *in the theoretical Hamming weight model over* 96*-bit data.*

LPN Experiments

Attack on Simulated Traces (96-bit). We successfully performed our new attack on AES-GCM for a block-size reduced to 96 bits. We generated a secret key k of 96 bits, then generated 2^{20} uniform randomly generated a^i. We then simulated a leakage corresponding to the one obtained on the ATMega328p (*i.e.,* with the same statistics) and chose λ equal to 3.80 (thus filtering with probability 2^{-10} and having an error probability of 0.387).

Table 4.6: Experimental and theoretical parameters corresponding to filtering proportion $F(\lambda)$ on the ATmega for 96-bit AES-GCM

SNR $= 8.7073$, $\sigma = 0.0173$						
$\log_2(1/F(\lambda))$	12	10	8	6	4	2
λ_{exp}	4.27	3.80	3.29	2.76	2.14	1.31
p_{exp}	0.377	0.387	0.402	0.414	0.429	0.449

This amounted to keeping 916 relations, the less noisy one having weight 25 that is an error rate of 0.260. We used this relation for secret/error permutation. All in all, we got $87,840 \approx 2^{16,42}$ LPN equations. After 6 hours of parallelized generalized birthday computation on a 32-core machine using 200 gigabytes of RAM, we had found more than 2^{39} elements reduced down to 36 bits. After a Walsh transform on the 36 bits (approximatively 2,000 seconds on the same machine), we recovered the 36 least significant bits of the error that are converted in 36 bits of the secret. This heavy computation corresponds to the most complex part of the attack and validates its success. We can afterwards recover the remaining bits by iterating the attack with the knowledge of the (already) recovered bits. This is a matter of minutes since it corresponds to an attack on a 60-bit key which is much less expensive than the 96-bit case.

Expected Attack Complexities (128-bit). We provide here theoretical complexities for the key-recovery attack on 128-bit secret key. We can see on Figure 4.2 the evolution of the time complexity as a function of the available memory for the attack. Plots are provided for three different data complexities.

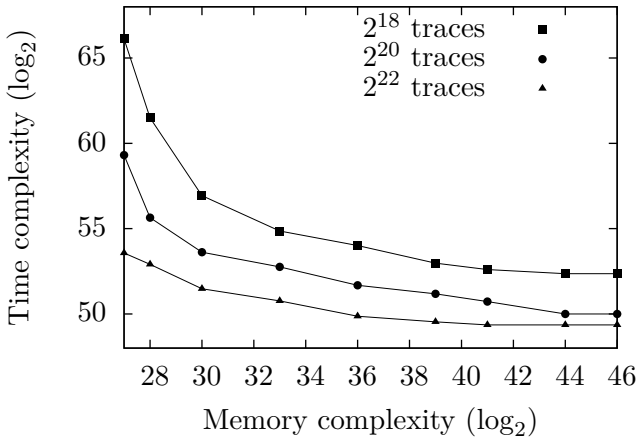


Figure 4.2: Estimated complexities of the 128-bit attack ($\mathsf{SNR} = 8.21$).

We notice that the time/memory trade-off is only exploitable up to one point. This is due to the fact that when lots of memory is available, one may perform a larger Walsh-Hadamard transform to obtain more reduced equations. At some point, the time complexity of this transform will be predominant compared to the birthday paradox step and thus there will be no gain in increasing the Walsh size. Here are few examples of time/memory trade-offs obtained for 2^{20} side-channel acquisitions.

- time complexity: $2^{59.31}$ and memory complexity: $2^{27.00}$
- time complexity: $2^{51.68}$ and memory complexity: $2^{36.00}$
- time complexity: $2^{50.00}$ and memory complexity: $2^{44.00}$

4.4.3 Attack on the AES-GCM Multiplication's Output with Chosen Inputs

We have mounted an attack on the AES-GCM multiplication's result with chosen inputs as described in Section 4.2.

Table 4.7: Experimental parameters for the attack on 128-bit AES-GCM (chosen inputs).

number of repetitions α	1	2^5	2^{10}	2^{11}
number of traces	129	$\approx 2^{12}$	$\approx 2^{17}$	$\approx 2^{18}$
$\mathsf{SNR} = 8.21$, $\sigma = 0.0206$				
τ	1.440	0.265	0.080	0.052
p	0.49	0.39	0.016	0.0

Leakage Acquisition on ATMega328p

Instead of generating random vectors, we measured this time the leakage during the processing of $z = a \cdot k$ for 129 vectors a corresponding to all the powers of two between 0 and 128. We simulated the leakage on 128 bits by summing the intermediate 8-bit leakage as for the attack on known inputs.

Comparison

In the case of chosen inputs, we do not need to filter. We compare two consecutive measurements and if the absolute value of their difference overpasses the optimal threshold, we assume that the targeted bit was set to one. Otherwise, we assume that it was zero. We thus obtain 128 equations of the 128 key bits with a certain error probability. We can directly use them or we can repeat the measurements on the same set of 129 vectors in order to reduce the level of the noise (*i.e.,* repeating α times the same computation results in a standard deviation divided by $\sqrt{\alpha}$). Table 4.7 gives the empirical thresholds and error probabilities according to the number of repetitions of our set of 129 measurements.

Key Recovery

We can recover the key very efficiently under the context of the last column of Table 4.7, that is with 2^{18} measurements and more precisely 2^{11} repetitions of 129 traces. In this case, the error probability is very close to zero which means that we usually directly get all the 128 key bits.

4.4.4 Attack on Fresh Re-keying

We detail here the attack that aims at recovering the master key from the leakages corresponding to the first round of the AES when the secret key is generated by the fresh re-keying primitive described in Section 4.3.

Leakage Acquisition

We randomly generated 15,000 vectors $a \in \mathsf{GF}(2)^{128}$ and 15,000 vectors $b \in \mathsf{GF}(2)^8$. We then measured the 8-bit leakage during the processing of $\mathsf{Sbox}(z_0 \oplus b)$ with z_0 the first byte of the multiplication between a and k.

Filtering

We filtered the extreme consumption measurements in order to exhibit the extreme Hamming weight values. Table 4.8 gives the empirical error probabilities according to the proportion of filtering on the 15,000 observations. As explained in Section 4.3, the error probabilities are naturally much lower than for a 128-bit leakage.

Table 4.8: Error probability p according to the proportion of filtered acquisitions $F(\lambda)$ on the ATMega328p for the fresh re-keying with known inputs

$\log_2(1/F(\lambda))$	9	8	7	6	5	4	3	2	1
$\mathsf{SNR} = 8.6921$, $\sigma = 0.0165$									
λ	0.555	0.514	0.473	0.432	0.391	0.349	0.288	0.226	0.123
p	0.0	0.013	0.056	0.089	0.11	0.15	0.18	0.22	0.29

Key Recovery

With a sufficient (but still reasonable) filtering, we can directly recover the key by inverting the linear system of equations. For instance, in our experiments, filtering one observation over 2^9 gives $33 \times 8 = 264$ linear equations on the bits of k without a single error. Thus, inverting the system directly gives us the correct key.

Chapter 5

Conclusion and Perspectives

5.1 Conclusion

In this part, we introduced two new methods to attack the authentication of AES-GCM in spite of the large size of the operands. We suggested to exploit either the LSB or the MSB of the leakage which led to a reasonable trade-off between the removal of errors induced by the noise and the solving complexity of the system to recover the secret hash key. We presented different attack scenarios according to the attacker capabilities and we proposed different techniques to improve the attacks efficiency.

In order to evaluate the applicability of our attacks on other algorithms, we also studied a different multiplication used for re-keying. We showed that the difference in the multiplication definition made our attacks fail when using the LSB and work when using the MSB. Furthermore, in both cases, the attacks are successful when the leakage is observed when the session key is manipulated through 8-bit chunks in the block cipher.

5.2 Perspectives

In a further step, we could analyze other kinds of multiplications which diffuse large secrets. While the first technique based on the LSB of the Hamming weight seems very specific to the multiplication in $\mathsf{GF}(2^{128})$, the second technique based on the MSB could apply to a large class of modular multiplications. As soon as the device leaks an information close to the noisy Hamming weight of the manipulated variable and the bits of the multiplication result are linear function of the secret key, we should be able to mount this second attack.

In a different direction, we could try to optimize again the algorithms used to solve the LPN by completely specifying them for our problem. Indeed, we already started to do that with the number of queries in this part. Nevertheless, if we figured out the optimal value of all the parameters for this context, we could probably still improve the solving complexities.

Part III

Countermeasures: masking and leakage-resilient primitives

Chapter 1

Introduction

Contents

1.1 Motivation

Since the discovery of Differential Power Analysis, a huge number of attacks have been revealed. In parallel, many countermeasures with different security levels have appeared to protect the broken implementations. Among these countermeasures, the use of leakage-resilient primitives and masking are the most commonly used and achieve the highest levels of security. At the beginning of this thesis, we noticed three important gaps on these countermeasures in the literature.

Firstly, the use of leakage-resilient primitives, although proven secure in the leakage-resilient cryptography model, generally comes with huge loss of performances. Therefore, the resulting implementations are often unpractical and cannot be used in real products.

Secondly, the masked implementations much more practical, have revealed unexpected security issues. While a first-order masked implementation is relatively easy to check, the higher-order masked implementation are much more complex to analyze. That is why, several works only prove small block functions and implicitly assume the security of their composition. Unfortunately, the past years show that such simple compositions may induce security flaws.

Thirdly, these two countermeasures of leakage-resilient functions and masking use to be considered very separately. Thus, in an industrial context, it may be really hard to tell which one of them should be used in an implementation with particular constraints.

1.2 Contributions

Our contributions on this subject concern each of the three aforementioned issues.

In a first attempt, we focus on the leakage-resilient primitives. We propose two leakage-resilient encryption schemes based on the fresh re-keying using the primitive AES. The first one is just the juxtaposition of a PRF for the re-keying and a block-cipher while the second

one is tweaked to be more efficient. Both of them are proved secure in the leakage-resilient cryptography model. *This work was presented at CHES in 2013 [ABF13].*

In a second attempt, we focus on the masking countermeasure. As explained above, some masking schemes defined a few years ago are now revealed insecure because of missing points or mistakes in the security proofs. In order to verify the security of the corresponding implementations without repeating the same errors, we build an automatic verifier. The latter determines the security of a C implementation masked at a reasonable order t in the value-based or the transition-based model. If it is not secure, it exhibits an attack path. *This work was presented at Eurocrypt in 2015 [BBD+15b].*

Finally, in a third attempt, we aim to understand why the existing schemes sometimes fail to achieve global security. Since most of them made assumptions on the security of the composition of secure functions, we investigate a secure way to connect secure blocks in order to guarantee that the result is secure too. We finally build a compiler which, from an unprotected implementation, generates its secure masked version at any order. *This work is available on ePrint [BBD+15a].*

One step further, since we missed clue to choose between leakage-resilient primitive or masking, we try to build a comparison between both countermeasures. The goal is to help industrial cryptographers to select the most relevant protection in order to achieve a fixed security level according to device constraints. We limit the study to two main primitives: the pseudo-random functions and the pseudo-random generators both instantiated with the AES. To compare the protections, we try to achieve bounded security whatever the number of measurements. Since this property is easy to achieve for PRGs, it is much more harder for PRFs which do not limit the number of measurements with the same key by design. Finally, we show that, usually, the best protection for PRG is the only use of leakage-resilient primitives while the best protection for PRF is the only use of masking. *This work was published in Cryptography and Communications in 2015 [BGS15].*

1.3 Outline

Chapter 2 introduces a new leakage-resilient and efficient encryption scheme based on the AES primitive. Chapter 3 and Chapter 4 focus on masking by describing the new verifier and the new compiler of secure implementations. Chapter 5 gives a framework to choose the most appropriate countermeasure according to a set of constraints between the use of leakage-resilient primitives, the masking or the combination of both.

Chapter 2

Leakage-Resilient Encryption Scheme

In this chapter, we aim to study the possibility to construct an efficient leakage-resilient symmetric scheme using the AES block cipher. While several fresh re-keying schemes suffer from synchronization issues with the permanent key change, we introduce a new encryption scheme based on skip-list data structures which improve the efficiency in such a context. We also build a complete game proof to guarantee its security in the leakage-resilient cryptography model.

Contents

2.1 Introduction

2.1.1 Motivation

As discussed in introduction (Part I), several pseudo-random functions, generators and permutations have already been proposed in the leakage-resilient cryptography model [DP10, FPS12,

Pie09, YS13, SPY+10]. Unfortunately, these primitives are not always relevant in practice. They are often associated to large complexities mainly because of proof artifacts.

In this chapter, we aim to build a more efficient and provably secure symmetric encryption based on fresh re-keying. This technique has first been investigated in [AB00] in the context of increasing the lifetime of a key and in [Koc03] to thwart side-channel attacks by updating regularly the secret key. Such schemes have already been designed [MSGR10] but no security proof has been given. Here, we rather focus on a mode of operation provably secure in the leakage model. A first requirement for the security is to encrypt each block of message with a different session key. As formally described in [AB00], the session keys can be generated either from the same master key (in parallel by applying a pseudo-random function on the index with a part of the master key) or sequentially, using the previous session key to derive the current one. Although the choice of the model depends on the underlying primitive, the second one is more suited to avoid DPA as it changes the key at each execution in the re-keying part also. However, the sequential method faces a problem of efficiency when a client and a server need to re-synchronize. For example, servers that operate with many clients (as in electronic cash applications) cannot pre-compute all the possible session keys. They have to derive them, that is operate as many operations as the difference of indexes between the key they currently have and the key they need. As a result, the process of re-keying suffers from the time complexity of the number of similar operations required to obtain the correct session key.

2.1.2 Related Work

The first construction of leakage-resilient symmetric encryption has only been recently proposed by Hazay *et al.* in [HLWW12]. However, the main objective of the authors was to propose a generic scheme based on minimal assumptions and the efficiency was not their priority. Besides, the security relies on a global leakage bound for all the queries. There are several other works on the construction of leakage-resilient symmetric primitives such as block ciphers or stream ciphers [Pie09, DP10, DP08]. One of the main assumptions to design such schemes is that AES implementations are heuristically secure against SPA [BK07, SLFP04, Mor12, MME10, GS12] or AES can be implemented to be a leakage-resilient block cipher if the number of queries with the same key is small. A recent work of Veyrat-Charvillon, Gérard and Standaert [VGS14] shows that an AES key may be recovered with a very few consumption traces. However, these attacks are limited to software and rely on the same assumption as template attacks that are rarely fulfilled by an attacker. Consequently, the AES block cipher remains the main practical building block used by theoreticians to instantiate their constructions and namely in [Pie09], Pietrzak proposes to use $\mathrm{AES}(0\|p)\|\mathrm{AES}(1\|p)$ for constructing a weak PRF with $2n$ bits of outputs. A weak PRF is a PRF when the adversary cannot choose the inputs but only has access to random evaluations of the function. Such a weak PRF is a critical ingredient of the design of GGM (Goldreich, Goldwasser and Micali) leakage-resilient PRF [GGM84] which is resistant for many queries and not only two. To construct a leakage-resilient block cipher, Faust, Pietrzak and Schipper propose to use this PRF in a three Feistel rounds in [FPS12] but the overall construction has been shown to be inefficient by Bernstein at the CHES'12 rump session [Ber12]. The GGM construction is however very inefficient and in an effort to improve it, Medwed, Standaert and Joux in [MSJ12] propose a version of the tree by considering byte rather than bit (see Part I, Section 2.4.2). They analyze the security of this variant with the AES block cipher and lead to the conclusion that AES is not well-suited for this purpose. Indeed, even though the adversary does not control the inputs, she can still efficiently recover the secret key of the first round byte after byte. A similar conclusion has been made by Jaffe in [Jaf07] on the AES-CTR mode of operation. Constructing a leakage-resilient PRF is a crucial issue since

the construction of a leakage-resilient block-cipher as in [DP10, FPS12] or a leakage-resilient stream-cipher [Pie09] requires this primitive. In this chapter, we investigate this issue in order to build a practical leakage-resilient symmetric encryption scheme.

2.1.3 Contributions

Our goal is to construct an efficient leakage-resilient symmetric encryption scheme using the AES block cipher without constructing a leakage-resilient block cipher. Since AES is only secure if a limited number of plaintexts is encrypted with the same key, the key should be regularly changed. Therefore, re-keying appears to be an interesting solution as it was earlier proposed by Kocher in [Koc03] to avoid DPA attacks. Here, we want to show that this idea leads to an efficient leakage-resilient symmetric encryption scheme. To this end, we need to construct an efficient re-keying scheme. However, to design such a scheme, one solution is to use a leakage-resilient PRF and we will be back to our initial problem since we want to use AES in an efficient leakage-resilient encryption scheme. Our first solution consists in showing that a leakage-resilient PRF combined with a block cipher is a leakage-resilient encryption scheme. For this purpose, we can use the GGM construction as proven by Faust *et al.* in [FPS12]. However, in order to build a more efficient scheme and to solve the synchronization problem, avoiding the computation of all the intermediate keys, we propose a new re-keying scheme. We show that we do not need a PRF to build a secure encryption scheme, but only a *leakage-resilient re-keying scheme*. We build it by using similar ideas from the skip-list data structure [Pug89] proposed by Pugh in the late eighties. In this list, one half of the main list elements is chosen randomly to constitute a new list and from this list, another smaller one is derived and so on using $O(\log n)$ stages with high probability for n elements. The idea to look for an element in this *sorted* list consists in beginning with the last floor and identifying the interval where the element is and then recurse in the next floor up to identifying the element or finding that it is not in the list. On average, the running time is also $O(\log n)$ which is asymptotically as efficient as a random binary tree. Our problem is similar to finding an element in a sorted list since we have the index of the key we are looking for. It turns out that this construction serves the same purpose than the one proposed by Kocher in [Koc03]. However, the latter does not share the same design mainly because of the multiple use of the same re-keying keys and suffers from the absence of security proof.

2.1.4 Outline

The description of leakage-resilient cryptography is recalled in Part I, Chapter 2. Section 2.2 gives the formal definitions and security notions. In Section 2.3, we describe our new leakage-resilient symmetric encryption. In Section 2.4, we provide the security proof of our construction while in Section 2.5, we evaluate its efficiency in practice.

2.2 Definitions and Security Notions

In this section, we formally define the functions which will be used in this chapter. But beforehand, we give a few notations.

Notation. *The set of uniformly random functions from* $\{0,1\}^m$ *to* $\{0,1\}^n$ *is denoted by* $\mathcal{R}_{m,n}$ *and for a set* $\mathcal{X}$*,* $X \xleftarrow{*} \mathcal{X}$ *denotes the sampling of a uniformly random* X *from* $\mathcal{X}$*.*

Secure and efficient cryptosystems require functions which are indistinguishable from equivalent random objects and which require only a few amount of randomness. A widely used function

which fills these criteria is the PRF. To define the security notion of such a PRF F, we consider a challenge oracle which is either the PRF $F(k,.)$ instantiated with a uniformly random key k (with probability 1/2) or a uniformly random function $R(.)$. As formalized in Definition 3, the PRF is secure if no adversary is able to distinguish both games with a significant advantage.

Definition 3. *A function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ is a (ε, s, q)-secure PRF if for any adversary (or statistical test) $\mathcal{A}$ of size s that can make up to q disjoint queries to its challenge oracle, we have:*

$$Adv_F^{prf}(\mathcal{A}) = | \mathop{\mathbb{P}}_{k \xleftarrow{*} \{0,1\}^n} [\mathcal{A}(F(k,.)) = 1] - \mathop{\mathbb{P}}_{R \leftarrow \mathcal{R}_{m,\ell}} [\mathcal{A}(R(.)) = 1]| \leqslant \varepsilon.$$

Remark 4. *The size s of adversary needs to be restricted. Otherwise, this adversary could contain in it a huge table with all the possible solutions for each possible key and then use this table to break efficiently the PRF.*

A weak PRF (wPRF) shares the same definition except that its inputs are chosen uniformly at random. Contrary to the PRFs and as proven in [Pie09], the wPRFs remain secure whenever they are used with the so-called low keys defined below.

Definition 4. *A α-low key $K \in \{0,1\}^n$ is a key with min-entropy equal to $n - \alpha$:*

$$\forall x \in \{0,1\}^n, \quad \mathbb{P}[K = x] \leqslant 2^{-(n-\alpha)}.$$

Both wPRFs and PRFs can be leakage-resilient secure. As explained in Part I, this second security notion requires the introduction of a second oracle referred to as leakage oracle and denoted by $F^f(k,.)$. It takes as inputs both the inputs of function F and a leakage function f, and it returns both the output of the function $F(k,X)$ and the corresponding leakage $f(k,X)$ on an input query X.

Definition 5. *A function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ is a (ε, s, q)-secure leakage-resilient PRF if for any adversary $\mathcal{A}$ of size at most s who can make up to q distinct queries to its two oracles, we have:*

$$\begin{aligned} Adv_F^{lr\ prf}(\mathcal{A}) = \quad & | \mathop{\mathbb{P}}_{k \xleftarrow{*} \{0,1\}^n} [\mathcal{A}(F(k,.), F^f(k,.)) = 1] \\ & - \mathop{\mathbb{P}}_{R \leftarrow \mathcal{R}_{m,\ell}, k \xleftarrow{*} \{0,1\}^n} [\mathcal{A}(R(.), F^f(k,.)) = 1]| \leqslant \varepsilon. \end{aligned}$$

Remark 5. *As explained in Section 2.4.2, we assume the leakage function f to be non-adaptive, that is, it must be determined by the adversary before the submission of the first query. Letting the adversary choose the leakage function (under some reasonable constraints) makes sense since the scheme should be secure independently of the inherent device (as soon as the leakage is bounded per invocation), and thus independently of this function's shape. Moreover, having the attacker make this choice before the first query is not only mandatory to be able to build practical schemes with security proofs but also much more realistic in our opinion. Indeed, we strongly believe that the leakage function should be fixed once for all at the very beginning since it is mainly determined by the targeted device. As an example, we often observe devices leaking the noisy Hamming weight of their manipulated data. We think that it would be totally unrealistic for such devices to leak these noisy Hamming weights during the execution of the first query and then to leak the less significant bit of each variable in the second one. Eventually, the only point, in our sense, which would justify the use of adaptive leakage functions would be to model specific attacks such as electromagnetic attacks where the adversary can change the antenna position*

between different executions. In this case, the inputs are different but the leakage's shape should still be the same. Thus, we suggest to increase the granularity with more blocks and thus more leakage functions (still fixed at the beginning) or to include all these possibilities in the leakage function (with eventually a larger λ bound).

Although they also provide pseudo-randomness, encryption schemes are stronger notions than PRFs. Given the ciphertexts of two different messages, no adversary can decide with a significant confidence which ciphertext is related to which plaintext. In this chapter, we focus on an equivalent security notion for encryption schemes introduced in [BDJR97], and recalled in introduction: the real-or-random indistinguishability. The security of an encryption scheme is then verified if no adversary can distinguish the encryption of a real query from the encryption of a random string.

Definition 6. *An encryption scheme $S : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m$ is (ε, s, q)-secure in the real-or-random sense, if any adversary $\mathcal{A}$ of size at most s who asks at most q distinct queries, has an advantage bounded as follows:*

$$Adv_S^{enc}(\mathcal{A}) = |\underset{k \xleftarrow{*} \{0,1\}^n}{\mathbb{P}}[\mathcal{A}(S_k(.)) = 1] - \underset{k \xleftarrow{*} \{0,1\}^n}{\mathbb{P}}[\mathcal{A}(S_k(\$)) = 1]| \leqslant \varepsilon$$

where $\$$ represents a random string in $\{0,1\}^m$.

Let us now define the leakage-resilient security of an encryption scheme. This notion ensures that even with additional leakage, no adversary should be able to distinguish both games with a significant advantage. Concerning the PRFs, we consider a leakage oracle referred to as $S_k^f(.)$ for a uniformly random key k.

Definition 7. *An encryption scheme $S : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m$ is a (ε, s, q)-secure leakage-resilient encryption scheme if any adversary $\mathcal{A}$ of size at most s who asks at most q distinct queries has an advantage bounded as follows:*

$$Adv_S^{lr\ enc}(\mathcal{A}) = \quad |\underset{k \xleftarrow{*} \{0,1\}^n}{\mathbb{P}}[\mathcal{A}(S_k(.), S_k^f(.)) = 1] - \underset{k \xleftarrow{*} \{0,1\}^n}{\mathbb{P}}[\mathcal{A}(S_k(\$), S_k^f(.)) = 1]| \leqslant \varepsilon.$$

In the following, we will consider a function secure if the advantage of the attacker is negligible in the key size n and if s and q are super-polynomial in n.

2.3 Leakage-Resilient Symmetric Encryption Scheme

In this section, we propose to build a non-adaptive leakage-resilient symmetric encryption scheme. As suggested by Kocher in [Koc03], this security can be achieved by key updates, also referred to as re-keying, combined with secure primitives. Following this design principle, we propose in a first part a construction based on a naLR naPRF (as defined in 2.4.2) and a block cipher which yields a naLR encryption scheme secure in the sense of Definition 7. In a second part, we focus on the instantiation of the inherent naLR naPRF. We start with the recent construction of [FPS12] since to the best of our knowledge, it is the most efficient proven one. Based on this construction, we try to improve the efficiency of the whole scheme in the context of re-synchronization. However through the improvements, we observe that the naLR naPRF is not a requirement to build a naLR encryption scheme. In fact, we introduce a new re-keying scheme which does not fulfill the properties of a PRF but surprisingly still yields a naLR encryption scheme. Furthermore, it improves significantly the efficiency of a sequential re-keying scheme when instantiated with the AES in the context of the synchronization issue exhibited in Section 2.1. Eventually, we conclude the section by discussing the generation and the repartition of the public random values used in the whole encryption scheme.

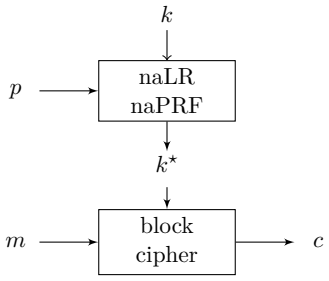


Figure 2.1: Non-adaptive leakage-resilient encryption scheme from a naLR naPRF.

2.3.1 Leakage-Resilient Encryption from a naLR naPRF

As outlined in [MSJ12], PRFs appear to be good candidates for integration in leakage-resilient re-keying schemes. In this chapter, we show that a naLR naPRF F associated with a secure block cipher β (in the sense of PRF in Definition 3) yields a naLR encryption scheme. For this purpose, Theorem 1 is proven is Section 2.4.

Theorem 1. *Let F denote a naLR naPRF and β a block cipher in the sense of PRF. Then the symmetric encryption scheme $S^{F,\beta}$ is a non-adaptive leakage-resilient encryption scheme. The amount of leakage λ tolerated per time step depends on the inherent naLR naPRF: $\lambda \in O(log(1/\varepsilon_F))$.*

The principle is as follows. From an initial secret key k and a public random value p, the PRF outputs a session key $k^\star = F(k, p)$ that is further used by the block cipher for a single block encryption. Figure 2.1 illustrates this process. Since each block is encrypted with a different secret key, the block cipher is not expected to be resistant against differential power analysis.

2.3.2 Leakage-Resilient Encryption Scheme from a Weak PRF

We have proposed the construction of a naLR encryption scheme from a naLR naPRF. Now we aim to instantiate this naLR naPRF in the most efficient way. Since it is proven secure, we start with the naLR naPRF introduced in [FPS12]. This construction relies on a binary tree where each node corresponds to an intermediate key and the edges represent the encryptions with public values. The root is the PRF key and the leaves are the outputs which are selected according to the chosen path. This path is determined by the plaintext bits: a zero leads the execution through the left edge and a one to the right edge. We observe that it has likable security features but it is not optimal in terms of efficiency since among all the nodes of the binary tree, only the leaves are finally exploited. As a consequence, we propose to improve the efficiency of the whole scheme by also using the intermediate nodes in a suitable order.

A solution to benefit from the intermediate nodes is to slightly change the inherent wPRF. In [FPS12], this wPRF outputs $2n$-bit values from n-bit inputs. In this chapter, we refer to as ϕ the wPRF we use to compute n-bit values from n-bit values and as ϕ_2 (resp. ϕ_3) the concatenation of two (resp. three) invocations of ϕ. Instead of deriving two keys generally from two random public values of same size, we could directly use the wPRF ϕ_3 to derive three keys using one more random value. Among these three keys, two would still be used to derive the subsequent keys in the tree while the third one would be processed in the block cipher. Although this solution exploits the intermediate nodes, it requires a more expensive derivation since the intermediate wPRF is expected to generate one more key with an additional amount of randomness.

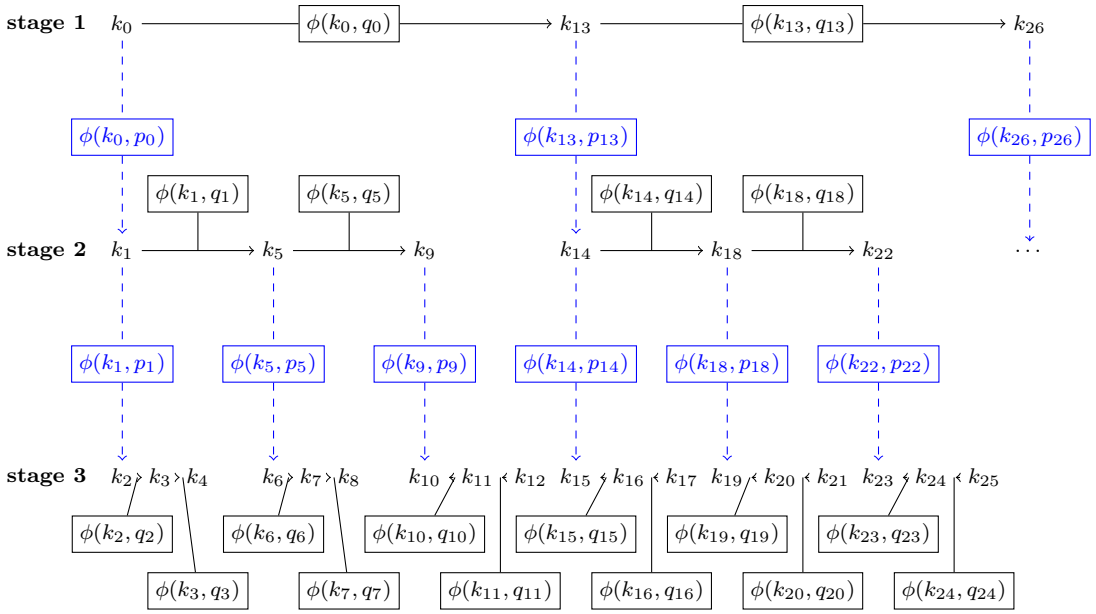


Figure 2.2: Stage representation of our new re-keying scheme R_ϕ with $x = 3$

A more efficient option is to maintain the binary tree construction with the wPRF ϕ_2 and to use directly the intermediate keys in the block cipher. In this case, a third random value can be used with the intermediate key and the output of the block cipher can be bitwise added to the chosen message[1]. However, the re-keying scheme involved in this new construction is not a PRF anymore, since an adversary can easily take advantage of its outputs to derive the following keys. One may consequently expect the encryption scheme (we refer to as $S^{R_\phi,\phi}$) not to be secure anymore. Surprisingly, this intuition is wrong. By Theorem 2 that will be proven in Section 2.4, we show that we are still able to build a naLR symmetric encryption scheme with relaxed properties on the re-keying scheme. However, unlike the previous scheme, the proof that we established, requires the block cipher to be the same primitive than the wPRF used to derive the keys.

Theorem 2. *Let ϕ be a secure wPRF. Let R_ϕ denote the re-keying scheme described above. Then $S^{R_\phi,\phi}$ is a naLR encryption scheme. The amount of leakage λ tolerated per time step depends on ϕ: $\lambda = O(log(1/\varepsilon_\phi))$.*

Now that we have presented the security aspects when exploiting all the nodes of the binary tree, we still have to fix a suitable order in the nodes to be as efficient as possible in the re-synchronization scenario. For this purpose, we need to define short-cuts between distant keys to avoid the expensive computation of all the intermediate keys. Inspired by the skip-list data structure introduced in [Pug89], we suggest to organize our re-keying scheme in x stages, $x \geqslant 2$ containing increasingly sequences of keys[2]. This organization in lists, illustrated in Figure 2.2 with p_i and q_i public random values, involves a more efficient lookup with a reduction of the complexity from linear to logarithmic. Nevertheless, it is worth noticing that unlike skip-lists, this structure does not expect additional relations between keys. There is still one single

[1]We could also have chosen to add the message to the random value before the encryption, referring to Lemma 3 from [Pie09] in the case of leakage. However, in case the public random values are known after the first encryption, the message blocks should have been chosen non-adaptively to avoid non random inputs.

[2]In this description, each node generates x nodes at the first upper stage. Although convenient for the analysis, another choice can be made.

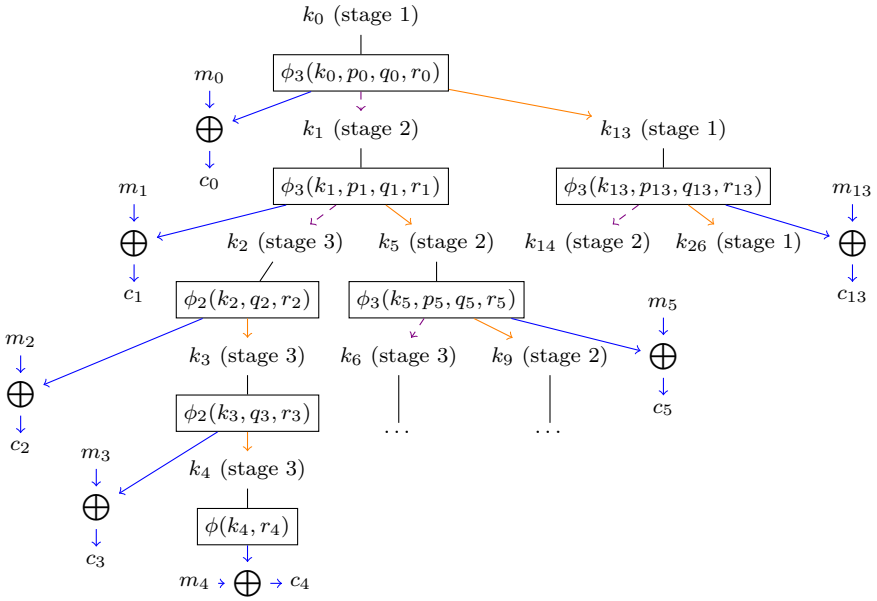


Figure 2.3: Tree representation of our new encryption scheme $S^{R_\phi,\phi}$ with three stages

computation path to derive each key. Figure 2.3 illustrates the whole encryption scheme using ϕ_3 for the concatenation of the block cipher and the wPRFs used for the derivation. The values r_i are the public random values used for the encryption, the values m_i represent the blocks of message to encrypt and the values c_i, the corresponding ciphertexts.

The very first key k_0 is the entry point of the first stage. Keys from the first stage allow to generate the following first stage keys or to go down to the second stage. When going down, the index of the key increases by one whereas when computing the next key in the same stage x_c, the index is increased by $1 + x + \cdots + x^{x-x_c}$. Each derivation requires one public value that we refer to as p_i when going down and q_i otherwise with i referring to the index of the key used for the derivation. In practice, the sender updates his own secret key after each operation, following the indexes order. When he wants to perform a transaction with the receiver, he just has to relay his current transaction counter. The receiver then performs a series of operations to derive the expected session key. For this purpose, he can either decide to always start with the very first key k_0 or to start with another key that he already computed and that is compliant with the new index. Algorithm 1[3] depicts the process for the first situation.

2.3.3 Efficient Generation of Random Values

For efficiency purpose, we try to minimize the generation of fresh randomness in our construction. We propose several methods to distribute the public random values. In a first attempt, we generate two fresh public random values p and q for the key derivation as illustrated in Figures 2.2 and 2.3 and one additional random value for the input message block. Although the encryption scheme is naLR, the solution is impractical. Another solution is to follow the method from [FPS12] and attribute two fresh random values by tree layer plus one for each block of message. This new proposal reduces the amount of randomness without loss of security since each path uses different random values for each time step.

The work of Yu and Standaert in [YS13] reduces even more the cost of the generation of

[3]The input x_c can also be directly computed from c.

Algorithm 1 Re-keying Scheme

Input: current key k_c, index c, stage x_c, new index i
Output: key k_i, index i, stage x_i

$(k, ind, x_t) \leftarrow (k_c, c, x_c)$
while $(ind \neq i)$ **do**
  while $(i > ind + \sum_{j=1}^{x-x_t} x^j)$ **do** ▷ Horizontal steps
    $k \leftarrow \phi(k, q_{ind})$
    $ind \leftarrow ind + \sum_{j=0}^{x-x_t} x^j$
  while $(i > ind)$ & $(i \leqslant ind + \sum_{j=1}^{x-x_t} x^j)$ **do** ▷ Vertical steps
    $k \leftarrow \phi(k, p_{ind})$
    $ind \leftarrow ind + 1$
    $x_t \leftarrow x_t + 1$
return (k, x_t)

randomness. It suggests to replace the randomness by pseudo-random values generated by a PRF in counter mode from a single public seed. This solution can directly be applied to ours and results in a significant improvement of the performances. The global scheme can still be proven naLR in the peculiar world of `minicrypt` [Imp95], that is either the scheme is secure or we can build public-key primitives from our symmetric functions, which is very unlikely.

Theorem 3. *Let ϕ be a weak PRF and G a PRF. Then the system $S^{R_\phi,\phi,G}$ described above is a naLR encryption scheme or we can build public-key primitives from the PRFs ϕ and G and the related leakage functions.*

2.4 Leakage-Resilient Security Analysis

In this section, we give the security proofs of the three theorems presented in Section 2.3.

2.4.1 Security Analysis of Theorem 1

In Section 2.3, Theorem 1 states the non-adaptive leakage-resilient security of an encryption scheme composed of a naLR naPRF and a block cipher, as illustrated in Figure 2.1. The following depicts the proof.

Proof of Theorem 1. From the granular leakage-resilient model, our scheme is split into time steps which leak independently. The attacker is allowed to choose non-adaptively a leakage function $f = (f_1, f_2)$ with components for each of these time steps: f_1 for the PRF and f_2 for the block cipher. Then, she can submit q distinct queries to her oracles which can be divided between challenge queries and leakage queries. For each challenge query, she gets back either the real output of her query or the encryption of a random string. For each leakage query, she gets both the real output of her query and the leakage which is exactly the output of the leakage function f she chose.

We now prove Theorem 1, that is, that the construction is a naLR encryption scheme, with a sequence of games. The first one, referred to as Game 0, represents the real case, that is when the attacker gets both the leakage and the real outputs of her queries. It directly corresponds

to the left-hand side probability in Definition 7 for an adversary $\mathcal{A}$ having access to challenge and leakage oracles:

$$\mathbb{P}_{k \xleftarrow{*} \{0,1\}^n}[\mathcal{A}(S_k(.), S_k^f(.)) = 1]. \tag{2.1}$$

We denote by G_0 the event $\mathcal{A}(S_k(.), S_k^f(.)) = 1$ which corresponds to Game 0. The last game Game N represents the random case, that is when all the challenge outputs are generated from random queries. It corresponds to the right-hand side probability in Definition 7 with \$ a random string in $\{0,1\}^n$:

$$\mathbb{P}_{k \xleftarrow{*} \{0,1\}^n}[\mathcal{A}(S_k(\$), S_k^f(.)) = 1]. \tag{2.2}$$

Similarly, we denote by G_N the event: $\mathcal{A}(S_k(\$), S_k^f(.)) = 1$. We aim to show that the difference between these two probabilities (2.1) and (2.2) (which is exactly the advantage of the attacker according to Definition 7) is negligible. To proceed, we go through intermediate games and we iteratively prove that the probability $\mathbb{P}(G_i)$ corresponding to Game i is negligibly close to the probability $\mathbb{P}(G_{i+1})$ corresponding to Game $i+1$ for $i \in \{0, \ldots, N-1\}$. The difference between two successive games is expected to be very small for the needs of the proof. In the following, we will use three kinds of games transition: the transitions based on indistinguishability, the transitions based on failure events and the so-called bridging steps.

Game 0 [REAL]

This game is referred to as the *real* game. In this game, the attacker $\mathcal{A}$ who submits adaptive leakage and challenge queries to her challenger, gets back both the leakage and the real outputs of her queries. We depict below the process in details.

Leakage functions. The adversary $\mathcal{A}$ first chooses a leakage function $f = (f_1, f_2)$ to observe the leakage during both the naLR naPRF and the block cipher executions. In our security model, we impose the leakage function to be chosen non-adaptively that is before seeing any leakage or outputs.

Challenge. Before everything, $\mathcal{A}$'s challenger chooses uniformly at random a key $k \in \{0,1\}^n$ for the naLR naPRF F. Subsequently, $\mathcal{A}$ is allowed to submit *adaptively* to her challenger q *distinct* queries whose q_0 are leakage queries $m_1, \ldots, m_{q_0}$ and $q_1 = q - q_0$ are challenge queries $m'_1, \ldots, m'_{q_1}$. Then, for each leakage query m_i, the challenger receives from $\mathcal{A}$, the challenger chooses uniformly at random an index ind_i as input for the PRF and returns both the real output c_i and the corresponding leakage:

$$c_i \leftarrow \beta_{k_i}(m_i) \quad \text{with} \quad k_i = F(k, ind_i) \qquad \text{and} \qquad f_1(k, ind_i), \quad f_2(k_i, m_i).$$

For each challenge query m'_i, the challenger also chooses uniformly at random an index ind'_i and returns to $\mathcal{A}$ only the real output:

$$c'_i \leftarrow \beta_{k'_i}(m'_i) \quad \text{with} \quad k'_i = F(k, ind'_i).$$

Game 1 [bridging step]

In Game 0, the challenger chooses uniformly at random an index as input for the naLR naPRF at each leakage or challenge query. We make here a conceptual change. Instead of choosing the indexes at each query, the challenger chooses now all the q indexes at the very beginning, that is before receiving the first query from the attacker. Since the indexes are chosen uniformly at

random, it does not change anything for the attacker. However, it is mandatory to use the naLR naPRF since it expect its inputs to be chosen non-adaptively that is before seeing any leakage or output. Therefore,

$$\mathbb{P}[G_0] = \mathbb{P}[G_1].$$

Game 2 [transition based on a failure event]

In Game 1, the challenger chooses uniformly at random the q indexes of the future queries. We now modify this first game so that we ensure that all the uniformly random indexes chosen by the challenger are pairwise distinct. It is easy to note that Games 1 and 2 proceed identically, unless there is a collision in the set of uniformly random indexes. Let us denote by E this specific event. If E does not occur ($\overline{E}$), then the output of both games is exactly the same. Equivalently, the following relation is satisfied:

$$G_1 \wedge \overline{E} \Leftrightarrow G_2 \wedge \overline{E}.$$

Then, from the common Lemma denoted by Difference Lemma in [Sho04], we have the following result:

$$|\mathbb{P}[G_1] - \mathbb{P}[G_2]| \leqslant \mathbb{P}[E].$$

Let us now determine the probability of event E. The q index values of Game 0 are sampled uniformly at random. Therefore, from the birthday bound, the probability of having at least a collision is less than $q(q-1)/2^{n+1}$ if $q \leqslant n$. This result gives us a bound on the difference between the probabilities of the events of $b = 1$ in both games:

$$|\mathbb{P}[G_1] - \mathbb{P}[G_2]| \leqslant \frac{q(q-1)}{2^{n+1}}.$$

Game 3 [transition based on indistinguishability]

We now make a small change to the above game. Namely, instead of computing the intermediate keys from the naLR naPRF F (for the challenge queries), we generate them using a random function:

$$R \leftarrow \mathcal{R}_{n,n} \qquad k'_i \leftarrow R(m'_i) \qquad c'_i \leftarrow \beta_{k'_i}(m'_i) \qquad \forall i \in [q_1]$$

or indistinguishably

$$k'_i \leftarrow \$ \qquad c'_i \leftarrow \beta_{k'_i}(m'_i) \qquad \forall i \in [q_1]$$

where \$ is a random string in $\{0,1\}^n$.

We now consider an adversary $\mathcal{A}$ who distinguishes Game 2 and Game 3. $\mathcal{A}$ interacts with a challenger $\mathcal{C}_\mathcal{A}$ and gets back either outputs using random keys if $b = 1$ or outputs using keys computed from the naLR naPRF F if $b = 0$. $\mathcal{A}$ then outputs a bit $b'_\mathcal{A}$ in view of the information she got that aims to be identical to b.

Let us now show that if $\mathcal{A}$ actually exists, we are able to build a new adversary $\mathcal{B}$, of size at most s, against the naLR naPRF F who uses $\mathcal{A}$. The process is as follows. As above, the adversary $\mathcal{A}$ first chooses a leakage function $f = (f_1, f_2)$ but this time, she sends it to the adversary $\mathcal{B}$ who transmits f_1 to her own challenger $\mathcal{C}_\mathcal{B}$. Then, as the challenger of adversary $\mathcal{A}$ in previous games, $\mathcal{C}_\mathcal{B}$ chooses q uniformly random *distinct* indexes and a uniformly random key k that she submits to the naLR naPRF leakage and challenge oracles with the leakage function f_1. The challenger then gets back from the leakage oracle and transmits to $\mathcal{B}$ both the leakage during the execution of this naLR naPRF:

$$f_1(k, ind_i) \quad \forall i \in [q_0]$$

and the keys corresponding to the leakage queries:

$$k_1, \ldots, k_{q_0}.$$

Then, $\mathcal{C}_{\mathcal{B}}$ gets back from the challenge oracle the keys $k'_i,\ i \in [q_1]$, representing either the real keys (if $b = 0$) or uniformly random strings in $\{0,1\}^n$ (if $b = 1$). Once these data collected, $\mathcal{B}$ is ready to answer the adaptive leakage and challenge queries of $\mathcal{A}$. For each leakage query m_i, $\mathcal{B}$ computes $\beta_{k_i}(m_i)$ and $f_2(k_i, m_i)$ and with the oracle previous replies sends back to $\mathcal{A}$:

$$f_1(k, ind_i) \qquad f_2(k_i, m_i) \qquad c_i = \beta_{k_i}(m_i).$$

For each challenge query $m'_i, i \in [q_1]$, $\mathcal{B}$ directly uses the key k'_i given by its challenge oracle and sends back to $\mathcal{A}$ according to the bit b:

$$\begin{array}{lll} \text{b=0:} & c'_i = \beta_{k'_i}(m'_i) \quad \text{with} & k'_i \leftarrow F(k, ind'_i) \\ \text{b=1:} & c'_i = \beta_{k'_i}(m'_i) \quad \text{with} & k'_i \leftarrow \$. \end{array}$$

Finally, $\mathcal{A}$ got the leakage outputs corresponding to its leakage queries and the outputs of the encryption scheme with its challenge queries either with the real keys if $b = 0$ or with uniformly random keys if $b = 1$. This situation perfectly simulates Game 2 with the real keys (when $b = 0$) and Game 3 with the uniformly random keys (when $b = 1$). As a result, the bit $b'_{\mathcal{B}}$ given by adversary $\mathcal{B}$ after its challenge on the naLR naPRF F is exactly the same than the bit $b'_{\mathcal{A}}$ given by adversary $\mathcal{A}$ to distinguish both Games.

Consequently, an adversary who aims to distinguish both games, has the same advantage than an adversary who aims to break the security of the naLR naPRF F. This is bounded by the naLR naPRF advantage ε_F. Eventually, we have:

$$|\mathbb{P}(G_2) - \mathbb{P}(G_3)| = \varepsilon_F.$$

Game 4 [transition based on a failure event]

We now modify Game 3 so that in addition to be uniformly random, the keys we use in challenge queries, are also pairwise distinct.

$$k'_i \leftarrow \$ \quad \text{s.t.} \quad k'_i \neq k'_j \ \text{ if } i \neq j \qquad c'_i \leftarrow \beta_{k'_i}(m'_i) \qquad \forall i, j \in [q_1].$$

It is worth noticing that Games 3 and 4 proceed identically, unless there is a collision in the set of uniformly random keys. Let us denote by E this specific event. If E does not occur, then the output of both games is exactly the same. As detailed in [Sho04] with the Difference Lemma and since the following relation is satisfied:

$$G_3 \wedge \overline{E} \Leftrightarrow G_4 \wedge \overline{E},$$

we have

$$|\mathbb{P}[G_3] - \mathbb{P}[G_4]| \leqslant \mathbb{P}(E).$$

We now determine the probability of event E. The q_1 key values of Game 1 are sampled independently and at random. Therefore, from the birthday bound, the probability of a collision is less than $(q_1(q_1 - 1))/(2^{n+1})$ if $q_1 \leqslant \sqrt{n}$. Straightforwardly, this result gives us a bound on the difference between the probabilities of the events of $b = 1$ in both games:

$$|\mathbb{P}[G_3] - \mathbb{P}[G_4]| \leqslant \frac{q_1(q_1 - 1)}{2^{n+1}}.$$

Game 5.0 [bridging step]

We now make a purely conceptual change to Game 4. In this game, the challenge outputs are computed from uniformly random keys without any intervention of the naLR naPRF:

$$\begin{array}{lll} k'_i & \leftarrow \$ \quad \text{s.t.} \quad k'_i \neq k'_j & \forall i,j \in [q_1],\ i \neq j \\ c'_i & \leftarrow \beta_{k'_i}(m'_i) & \forall i \in [q_1]. \end{array}$$

Since the keys are now uniformly random and pairwise distinct, the invocations to the block cipher are completely independent from each other. So, we can now consider them separately. Let us say that in this game, zero challenge query is computed from a random function and the q_1 others using the block cipher β. Clearly,

$$\mathbb{P}[G_4] = \mathbb{P}[G_{5.0}].$$

Game 5.t [transition based on indistinguishability]

We now modify Game 5.0 by replacing the t first invocations of the block cipher β for the challenge queries by invocations of a truly random function $R \leftarrow \mathcal{R}_{n,n}$:

$$\begin{array}{lll} & k'_i \leftarrow \$ \quad \text{s.t.} \quad k'_i \neq k'_j & \forall i,j \in [q_1],\ i \neq j \\ c'_i \leftarrow & R(m'_i) \quad \forall i \in [t] \quad \text{and} \quad c'_i \leftarrow \beta_{k'_i}(m'_i) & \forall i \in t+1,\ldots,q_1. \end{array}$$

We consider an adversary $\mathcal{A}$ who aims to distinguish Games 5.0 from Game 5.t. $\mathcal{A}$ first chooses a leakage function $f = (f_1, f_2)$ (respectively related to F and β) and sends it to her challenger. For each leakage query $m_i, i \in [q_0]$ she makes, $\mathcal{A}$ gets back:

$$f_1(k, ind_i) \qquad f_2(k_i, m_i) \qquad c_i = S^{F,\beta}(m_i) = \beta_{k_i}(m_i)$$

with k uniformly generated at random by the challenger. For the t first challenge queries $m'_1, \ldots, m'_t$ sent to her challenge oracle, $\mathcal{A}$ gets back (according to the game):

$$\begin{array}{ll} \text{Game 5.0:} & c'_i = \beta_{k'_i}(m'_i) \quad \forall i \in [t] \\ \text{Game 5.t:} & c'_i = R(m'_i)\ , \quad R \leftarrow \mathcal{R}_{n,n} \quad \forall i \in [t]. \end{array}$$

The other challenge queries are equivalent to Game 5.0. Eventually from these data, $\mathcal{A}$ output a bit $b'_{\mathcal{A}}$ indicating which game is played.

Now, let us show that if such an adversary $\mathcal{A}$ exists, we can build an adversary $\mathcal{B}$ against the block cipher β that uses $\mathcal{A}$. $\mathcal{B}$ proceeds as follows. $\mathcal{B}$ gets from $\mathcal{A}$ the leakage function $f = (f_1, f_2)$ and transmits it to her challenger $\mathcal{C}_{\mathcal{B}}$. $\mathcal{C}_{\mathcal{B}}$ generates the master k and the indexes uniformly at random and uses them to compute the leakage of F and the intermediate keys. For each leakage query $m_i, i \in [q_0]$ from $\mathcal{A}$, $\mathcal{C}_{\mathcal{B}}$ directly computes the leakage of the whole encryption. $\mathcal{B}$ gets back the corresponding leakage and outputs and sends to $\mathcal{A}$:

$$f_1(k, ind_i) \qquad f_2(k_i, m_i) \qquad c_i = \beta_{k_i}(m_i).$$

For each of the t first challenge queries from $\mathcal{A}$, $\mathcal{B}$ sends both the query and the corresponding intermediate key to her challenge oracle. According to the bit b, $\mathcal{B}$ gets back and returns to $\mathcal{A}$:

$$\begin{array}{ll} \text{b=0:} & c'_i = \beta_{k'_i}(m'_i) \\ \text{b=1:} & c'_i = \$. \end{array}$$

The other challenge queries are directly computed by $\mathcal{B}$ and sent back to $\mathcal{A}$.

Eventually, this experience perfectly simulates Games 5.0 when $b = 0$ and Game 5.t when $b = 1$. The adversary $\mathcal{B}$ faces the same challenge to distinguish the real output of the block cipher and a random string than $\mathcal{A}$ to distinguish both games. Their output bits $b'_{\mathcal{B}}$ and $b'_{\mathcal{A}}$ are perfectly equal which allows us to conclude that the difference between probabilities of events of both games when $b = 1$ is directly based on the security parameter of the block cipher as a PRF for each invocation:

$$\begin{aligned} |\mathbb{P}[G_{5.0}] - \mathbb{P}[G_{5.t}]| &= |\mathbb{P}[G_{5.0}] - \mathbb{P}[G_{5.1}] + \mathbb{P}[G_{5.1}] - \cdots - \mathbb{P}[G_{5.t}]| \\ &\leqslant |\mathbb{P}[G_{5.0}] - \mathbb{P}[G_{5.1}]| + \cdots + |\mathbb{P}[G_{5.t-1}] - \mathbb{P}[G_{5.t}]| \\ &\leqslant t \cdot \varepsilon_\beta. \end{aligned}$$

from the triangular inequality.

Game 6 [RANDOM]

In this game, the invocations of the block cipher corresponding to all the challenge queries are replaced by the invocations of uniformly random functions.

$$\begin{array}{rcll} k'_i \leftarrow \$ \quad \text{s.t.} & & k'_i \neq k'_j & \forall i, j \in [q_1],\ i \neq j \\ c'_i & \leftarrow & R(m'_i) & \forall i \in [q_1] \end{array}$$

or equivalently

$$\begin{array}{rcll} k'_i \leftarrow \$ \quad \text{s.t.} & k'_i & \neq k'_j & \forall i, j \in [q_1],\ i \neq j \\ c'_i & \leftarrow & \$ & \forall i \in [q_1]. \end{array}$$

This last game represents the right-hand side of the advantage of the attacker in Definition 7. Let us now compute the difference of the probability of G_0 corresponding to Game 0 and the probability of G_6 corresponding to this last game:

$$\begin{aligned} |\mathbb{P}[G_0] - \mathbb{P}[G_6]| &= |\mathbb{P}[G_0] - \mathbb{P}[G_1] + \mathbb{P}[G_1] - \mathbb{P}[G_2] + \mathbb{P}[G_2] - \mathbb{P}[G_3] + \mathbb{P}[G_3] - \mathbb{P}[G_4] \\ &+ \mathbb{P}[G_4] - \mathbb{P}[G_{5.0}] + \mathbb{P}[G_{5.0}] - \mathbb{P}[G_{5.t}] + \mathbb{P}[G_{5.t}] - \mathbb{P}[G_6]|. \end{aligned}$$

From the triangular inequality, we obtain:

$$\begin{aligned} |\mathbb{P}[G_0] - \mathbb{P}[G_6]| &\leqslant |\mathbb{P}[G_0] - \mathbb{P}[G_1]| + |\mathbb{P}[G_1] - \mathbb{P}[G_2]| + |\mathbb{P}[G_2] - \mathbb{P}[G_3]| + |\mathbb{P}[G_3] - \mathbb{P}[G_4]| \\ &+ |\mathbb{P}[G_4] - \mathbb{P}[G_{5.0}]| + |\mathbb{P}[G_{5.0}] - \mathbb{P}[G_{5.t}]| + |\mathbb{P}[G_{5.t}] - \mathbb{P}[G_6]| \\ &\leqslant 0 + \frac{q(q-1)}{2^{n+1}} + \varepsilon_F + \frac{q_1(q_1-1)}{2^{n+1}} + 0 + t \cdot \varepsilon_\beta + (q_1 - t) \cdot \varepsilon_\beta. \end{aligned}$$

Eventually, the advantage of an attacker against the encryption scheme described in Theorem 1 is bounded by:

$$\frac{q(q-1) + q_1(q_1-1)}{2^{n+1}} + \varepsilon_F + q_1 \cdot \varepsilon_\beta$$

which is negligible. □

2.4.2 Security Analysis of Theorem 2

Unfortunately, our new re-keying scheme is not a PRF since the adversary can easily take advantage of the outputs of the intermediate nodes to recover the following keys. However, we prove hereafter that, instantiated with a specific wPRF, it still yields a naLR encryption scheme.

Proof of Theorem 2. To prove Theorem 2, we first prove the security of the independent time steps in the new re-keying scheme. Let us consider an intermediate node of the re-keying scheme. If a challenge or a leakage query is asked on this node, the related operation will be the concatenation ϕ_3 of three wPRFs: two for the derivation of the next keys and one for the encryption. In the case no query is defined on this node, the concatenation ϕ_2 of only two wPRFs is required. As a result, we prove that the concatenation of two or three invocations of the wPRF ϕ using the same key (two for the derivation and in some cases one for the block cipher) still forms a secure wPRF.

Proposition 1. *Let $\phi : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a $(\varepsilon, s, 3q)$-secure wPRF.*

$$\begin{array}{rcl} \phi_3 : \{0,1\}^n \times \{0,1\}^{3n} & \to & \{0,1\}^{3n} \\ (k,p,q,r) & \mapsto & (\phi(k,p)\|\phi(k,q)\|\phi(k,r)) \end{array}$$

is then a (ε', s, q)-secure weak PRF with $\varepsilon' \leqslant \frac{3q(3q-1)}{2^{n+1}} + \varepsilon$ [4].

Proof of Proposition 1. As in the previous section, we organize this proof as a sequence of games. But this time, we aim to show the security of a function as a weak PRF in the sense of Definition 3. We start with Game 0 which represents the real game in which the attacker gets exclusively the real outputs of her queries.

Game 0 [REAL]

In this game the attacker gets the real outputs of her queries. Contrary to the previous proof, there is no leakage involved in the proposition, thus, we only consider q challenge queries.

Challenge. As usual, we consider an attacker $\mathcal{A}$ and her challenger $\mathcal{C}_{\mathcal{A}}$. The attacker $\mathcal{A}$ asks her challenger for q queries. The challenger $\mathcal{C}_{\mathcal{A}}$ then chooses uniformly at random an initial key $k \in \{0,1\}^n$ and q queries $m_1, \ldots, m_q \in \{0,1\}^{3n}$ and returns to $\mathcal{A}$ both the queries and their real outputs:

$$(m_i, \phi_3(k, m_i)) \;=\; ((p_i, q_i, r_i), (\phi(k,p_i)\|\phi(k,q_i)\|\phi(k,r_i))), \qquad \forall i \in [q].$$

We will now slightly transform this game until we reach the random game in which the adversary only gets random values in answer to her challenge queries. At each step, we show that the probability of $b = 1$ is negligibly close between consecutive games.

Game 1 [transition based on a failure event]

In Game 0, the challenger chooses q random and distinct challenge queries $(p_i\|q_i\|r_i)_{1 \leqslant i \leqslant q}$. We now slightly modify this game to ensure that the intermediate queries $\{p_i\}$, $\{q_i\}$ and $\{r_i\}$ for $i \in [q]$ are all pairwise distinct. From the birthday bound and the Difference Lemma in [Sho04], we obtain a bound on the difference between the probabilities of the events of $b = 1$ in both games:

$$|\mathbb{P}[G_0] - \mathbb{P}[G_1]| \leqslant \frac{3q(3q-1)}{2^{n+1}}.$$

[4] Similarly, the security parameter of ϕ_2 is bounded by $\frac{2q(2q-1)}{2^{n+1}} + \varepsilon$.

Game 2 [transition based on indistinguishability]

Game 1 is different from the original game since all the intermediate parts of the q queries are pairwise distinct. Let us now consider instead of the weak PRF ϕ_3, three invocations of a random function in $\{0,1\}^n$.

We consider an adversary $\mathcal{A}_3$ who is able to distinguish both games. We show now that we are able to build a new adversary $\mathcal{A}$ against the weak PRF ϕ who uses $\mathcal{A}_3$. The process is as follows. The challenger $\mathcal{C}_\mathcal{A}$ of the adversary $\mathcal{A}$ replaces the challenger of $\mathcal{A}_3$ to generate the initial key k and the random queries. She then submits to its challenge oracle $\mathcal{O}_\phi$ this initial key and the $3q$ distinct and uniformly random intermediate queries. According to the random bit b, she gets back and gives to $\mathcal{A}$ either the real outputs of the weak PRF ϕ or random values. Thus, $\mathcal{A}$ returns to $\mathcal{A}_3$:

$$\begin{array}{ll} b=0: & (m_i, \phi_3(k,m_i)) \;=\; ((p_i,q_i,r_i),(\phi(k,p_i)\|\phi(k,q_i)\|\phi(k,r_i))) \\ b=1: & ((p_i,q_i,r_i),(R(p_i)\|R(q_i)\|R(r_i))). \end{array}$$

with R a random function in $\{0,1\}^n$. This situation perfectly simulates Game 1 with the real outputs ($b=0$) and Game 2 with the random ones ($b=1$). The adversary $\mathcal{A}$ faces the same challenge in distinguishing the real and random outputs than $\mathcal{A}_3$ to distinguish between Games 1 and 2. Consequently, we have

$$|\mathbb{P}[G_1] - \mathbb{P}[G_2]| = \varepsilon_\phi.$$

Game 3 [transition based on a failure event]

In Game 1, we modified the inputs to ensure the absence of collision between the intermediate values. Now we switched to random functions, we can come back to the initial random values in order to reach the final game according to Definition 3. The difference of probabilities between these two games is straightforwardly the same than between Games 0 and 1:

$$|\mathbb{P}[G_2] - \mathbb{P}[G_3]| \leqslant \frac{3q(3q-1)}{2^{n+1}}.$$

Game 4 [REAL]

For the security definition to be perfectly verified, we conceptually modify Game 3 to only consider one invocation of a random function in $\{0,1\}^{3n}$ instead of three in a smaller range. Both games are equivalent and we can observe that this new one represents exactly the right-hand side of Definition 3. As a result, we obtain:

$$\begin{array}{rcl} |\mathbb{P}[G_0] - \mathbb{P}[G_4]| & = & |\mathbb{P}[G_0] - \mathbb{P}[G_1] + \mathbb{P}[G_1] - \mathbb{P}[G_2] \\ & + & \mathbb{P}[G_2] - \mathbb{P}[G_3] + \mathbb{P}[G_3] - \mathbb{P}[G_4]| \\ & \leqslant & \dfrac{3q(3q-1)}{2^n} + \varepsilon_\phi \end{array}$$

which concludes the security proof. □

Now that we have independent time steps, we can build the proof on the security model previously established. The attacker is still allowed to choose a global leakage function $f = (f_1, f_2)$ but this time, f_1 is related to the invocations of weak PRF ϕ_2 during the re-keying R_ϕ and f_2 is related to the final encryption. Then, she submits q_0 distinct leakage queries and $q_1 = q - q_0$ different and pairwise distinct challenge queries. For each leakage query, the adversary gets the leakage of the intermediate nodes computed with function ϕ_2 and both the

leakage and the output of the last node computed with function ϕ_3. For each challenge query, she receives either the real output or the encryption of a random string of the input's size. We now give the proof. As detailed in Section 2.4.1, the first game refers to the left-hand side probability of Definition 7 whereas the last game refers to the right-hand side probability in the same definition. By showing that the games are negligibly close, we prove that the advantage of the attacker in distinguishing between them is negligible and, as a result, that the scheme is leakage-resilient secure.

Game 0 [REAL]

In this game, the attacker gets from her challenger both the leakage and the real outputs of her leakage and challenge queries. The process is formalized below.

Leakage functions. The adversary $\mathcal{A}$ is allowed to choose a leakage function $f = (f_1, f_2)$ giving the leakage of the key derivation and the leakage of the encryption. Recall that the naLR security notion requires that the adversary choose this leakage function non-adaptively, that is before seeing any leakage or output.

Challenge. In this game, $\mathcal{A}$'s challenger chooses uniformly at random an initial key $k \in \{0,1\}^n$. Then, for each leakage query m_i chosen by $\mathcal{A}$, the challenger chooses uniformly at random an index ind_i for the naLR naPRF and returns the real output and the corresponding leakage:

$$c_i \leftarrow \phi(k_i, r_i) \oplus m_i \quad \text{and} \quad f_1(k, ind_i) \quad \text{and} \quad f_2(k_i, m_i)$$

with

$$r_i \xleftarrow{*} \{0,1\}^n \quad \text{and} \quad k_i \leftarrow R_\phi(k, ind_i).$$

For each challenge query m'_i, he also chooses uniformly at random an index ind'_i and only returns the real output:

$$c'_i \leftarrow \phi(k'_i, r'_i) \oplus m'_i \quad \text{with} \quad r'_i \xleftarrow{*} \{0,1\}^n \quad \text{and} \quad k'_i \leftarrow R_\phi(k, ind'_i).$$

We denote this first game by Game 0 and the related event by G_0. We will now transform this real game into a new one whose probability will be negligibly close.

Game 1 [bridging step]

In Game 0, the indexes for the PRF are chosen by the challenger at each query. In this game, we make a conceptual change consisting in choosing all the indexes before the first query. Since they are chosen uniformly at random, it does not change the advantage of the attacker. However, it is mandatory for the use of the naLR naPRF which requires non-adaptive inputs. We have:

$$\mathbb{P}[G_0] = \mathbb{P}[G_1].$$

Game 2 [transition based on indistinguishability]

In this game, we modify all the nodes involved in leakage queries including the intermediate ones. Since the related keys leak, we replace all the invocations of the related weak PRFs ϕ_2 and ϕ_3 by truly random functions: $R \leftarrow \mathcal{R}_{n,n}$:

$$\begin{aligned} k'_j &\leftarrow \$ && \text{for all keys generated from low keys} \\ c'_i \leftarrow \phi(k'_i, r'_i) \oplus m'_i & && \text{with} \quad r'_i \xleftarrow{*} \{0,1\}^n \quad \text{and} \quad k'_i \leftarrow (\phi \circ R)^\star(k, ind'_i) \end{aligned}$$

with $(\phi \circ R)^\star$ representing the combination of invocations of function ϕ and random function according to the nodes involved in leakage queries. To perform the reduction, we use a lemma from [Pie09] that we recall here.

Lemma 1 (Lemma 2 from [Pie09]). *For any $\alpha > 0$ and $t \in N$, if $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a (ε, s, q)-secure wPRF (for uniform keys), then it is a (ε', s', q')-secure wPRF with α-low keys if the following holds:*

$$\begin{aligned} q &\geqslant q'.t \\ \varepsilon &\leqslant \varepsilon'/2^{\alpha+1} - q^2/2^{n+1} - 2\exp(-t^2\varepsilon'^2/8) \\ s &\geqslant s'.t. \end{aligned}$$

Now, we consider an adversary $\mathcal{A}$ who aims to distinguish Games 1 and 2. $\mathcal{A}$ first chooses a leakage function $f = (f_1, f_2)$ and sends it to her challenger $\mathcal{C}$. For each leakage query m_i, $i \in [q_0]$ $\mathcal{A}$ submits, she gets back:

$$\begin{array}{rcl} & f_1(k, ind_i) & f_2(k_i, m_i) \\ c_i = \phi(k_i, r_i) \oplus m_i \text{ with } & r_i & \xleftarrow{*} \{0,1\}^n \text{ and } k_i \leftarrow R_\phi(k, ind_i). \end{array}$$

For each challenge query m'_i, $i \in [q_1]$, $\mathcal{A}$ gets back according to the game:

$$\begin{aligned} \text{Game 1} &: c'_i = \phi(k'_i, r'_i) \oplus m'_i \text{ with } r'_i \xleftarrow{*} \{0,1\}^n \text{ and } k'_i \leftarrow R_\phi(k, ind'_i) \\ \text{Game 2} &: c'_i = \phi(k'_i, r'_i) \oplus m'_i \text{ with } r'_i \xleftarrow{*} \{0,1\}^n \text{ and } k'_i \leftarrow (\phi \circ R)^\star(k, ind'_i). \end{aligned}$$

Now, we demonstrate that if such an adversary $\mathcal{A}$ exists, we are able to build an adversary $\mathcal{B}$ against the weak PRF that uses $\mathcal{A}$ as follows. $\mathcal{B}$ gets from $\mathcal{A}$ the leakage function $f = (f_1, f_2)$ and transmits it to her challenger $\mathcal{D}$. $\mathcal{D}$ generates the master key k and q_0 indexes (for each leakage query) uniformly at random and uses them to compute the leakage and the intermediate keys. Then for each leakage query m'_i submitted by $\mathcal{A}$, she sends the results to $\mathcal{B}$ who returns to $\mathcal{A}$:

$$\begin{array}{rcl} & f_1(k, ind_i) \text{ and } & f_2(k_i, m_i) \\ c_i = \phi(k_i, r_i) \oplus m_i \text{ with } & r_i & \xleftarrow{*} \{0,1\}^n \text{ and } k_i \leftarrow R_\phi(k, ind_i). \end{array}$$

For each challenge query m_i submitted by $\mathcal{A}$ and transmitted by $\mathcal{B}$, $\mathcal{D}$ computes the derivations for all the nodes not involved in the transformation and sends the data to her challenge oracle for the others. According to the bit b representing the choice of her oracle, $\mathcal{B}$ gets back the results, computes the encryptions and returns to $\mathcal{A}$:

$$\begin{aligned} b = 0 : \quad c'_i &= \phi(k'_i, r'_i) \oplus m'_i \text{ with } r'_i \xleftarrow{*} \{0,1\}^n \text{ and } k'_i \leftarrow R_\phi(k, ind'_i) \\ b = 1 : \quad c'_i &= \phi(k'_i, r'_i) \oplus m'_i \text{ with } r'_i \xleftarrow{*} \{0,1\}^n \text{ and } k'_i \leftarrow (\phi \circ R)^\star(k, ind'_i). \end{aligned}$$

Eventually, this experience perfectly simulates Game 1 when $b = 0$ and Game 2 when $b = 1$. Indeed, the adversary $\mathcal{B}$ faces the same challenge to distinguish between the real output of the weak PRF and a random string than $\mathcal{A}$ to distinguish both games. Thus, from this reduction, the probabilities related to both games are negligibly close. The difference directly comes from Lemma 2 in [Pie09] and the number of nodes involved in leakage queries.

$$\begin{aligned} |\mathbb{P}[G_1] - \mathbb{P}[G_2]| &\leqslant 2^{3\lambda+1}(\varepsilon_{\phi_2} + q_0^2/2^{n+1} + 2\exp(-\varepsilon_{\phi_2}^2/8)) \cdot (v_0 - q_0) \\ &+ 2^{3\lambda+1}(\varepsilon_{\phi_3} + q_0^2/2^{n+1} + 2\exp(-\varepsilon_{\phi_3}^2/8)) \cdot q_0 \end{aligned}$$

with v_0 the number of nodes involved in leakage queries. Note that the presence of two terms is related to the use of function ϕ_2 for keys derivation only and ϕ_3 at the last node of the query for also an encryption.

Let us now compute the bound on the number of nodes involved in leakage queries according to the parameters x (number of stages) and l (number of children of a node at the upper stage)[5]. We consider the worst case, that is when we always start from the initial key without storing any node, when no node is used in several leakage queries and we take the average index 2^{n-1}. We obtain the bound:

$$v_0 < q_0(\frac{2^{n-1}}{\sum_{j=0}^{x-1} l^j} + x \cdot l)$$

For instance, choosing $x = n$ and $l = 2$, gives us the following bound:

$$v_0 < q_0(\frac{2^{n-1}}{2^n - 1} + 2n) < q_0(1 + 2n).$$

Game 3 [transition based on indistinguishability]

In this game, we modify all the nodes involved in challenge queries except those involved in leakage queries and already transformed. We replace the weak PRFs instantiated with the corresponding keys by random functions $R \leftarrow \mathcal{R}_{n,n}$:

$$\begin{array}{ll} k'_j \leftarrow \$ & \text{for all keys involved in queries} \\ & c'_i \leftarrow \$ \oplus m'_i. \end{array}$$

Let us consider an attacker $\mathcal{A}$ who is able to distinguish Game 3 from Game 2. $\mathcal{A}$ first chooses a leakage function f and sends it to her challenger. For each leakage query m_i that $\mathcal{A}$ submits, she gets back:

$$\begin{array}{rcl} & f_1(k, ind_i) & f_2(k_i, m_i) \\ c_i = \phi(k_i, r_i) \oplus m_i \text{ with } & r_i & \xleftarrow{*} \{0,1\}^n \text{ and } k_i \leftarrow R_\phi(k, ind_i). \end{array}$$

For each challenge query m'_i, $\mathcal{A}$ gets back according to the game:

$$\begin{array}{ll} \text{Game 2} & : \; c'_i = \phi(k'_i, r'_i) \oplus m'_i \text{ with } r'_i \xleftarrow{*} \{0,1\}^n \text{ and } k'_i \leftarrow (\phi \circ R)^\star(k, ind'_i) \\ \text{Game 3} & : \; \$ \oplus m'_i. \end{array}$$

with $(\phi \circ R)^\star$ the combination of invocations ϕ and random functions (for the node involved in leakage queries) corresponding to Game 2.

We now show that if such an attacker exists, we can build an attacker $\mathcal{B}$ who is able to break the weak PRFs ϕ_2 and ϕ_3 using $\mathcal{A}$. The process is as follows. $\mathcal{A}$ first chooses a leakage function f and submits it to $\mathcal{B}$. $\mathcal{B}$ sends it to her challenger. The latter generates the master key k and the q_0 indexes for the leakage queries uniformly at random. He then computes both the leakage and the intermediate keys. Then, for each leakage query m_i that A submits to B, she transmits it to her challenger who computes and returns the following results:

$$\begin{array}{rcl} & f_1(k, ind_i) \text{ and } & f_2(k_i, m_i) \\ c_i = \phi(k_i, r_i) \oplus m_i \text{ with } & r_i & \xleftarrow{*} \{0,1\}^n \text{ and } k_i \leftarrow R_\phi(k, ind_i). \end{array}$$

[5] In Figure 2.2, $x = 3$ and $l = 3$.

Then, for each challenge query m'_i that $\mathcal{A}$ submits, $\mathcal{B}$ sends it to her challenger. The challenger sends the indexes to $\mathcal{B}$'s challenge oracle and $\mathcal{B}$ gets back, according to the bit b, the real outputs of the weak PRFs or random values. She sends back to $\mathcal{A}$:

$$\begin{array}{ll} b = 0 : & c'_i = \phi(k'_i, r'_i) \oplus m'_i \text{ with } r'_i \xleftarrow{*} \{0,1\}^n \text{ and } k'_i \leftarrow (\phi \circ R)^\star \\ b = 1 : & \$ \oplus m = \$. \end{array}$$

This experience perfectly simulates Game 2 when $b = 0$ and Game 3 when $b = 1$. So if the adversary $\mathcal{A}$ exists, we can build an adversary $\mathcal{B}$ who is able to break the previous weak PRF. As a result, we obtain the following equality:

$$|\mathbb{P}[G_2] - \mathbb{P}[G_3]| \leqslant (v_1 - q_1) \cdot \varepsilon_{\phi_2} + q_1 \cdot \varepsilon_{\phi_3}$$

with v_1 the number of nodes involved in challenge queries. We apply the same computation than in Game 2 to bound this value:

$$v_1 \leqslant q_1 (\frac{2^{n-1}}{\sum_{j=0}^{x-1} l^j} + x \cdot l).$$

Still instantiating x by n and l by 2, we obtain:

$$v_1 \leqslant q_1 (\frac{2^{n-1}}{2^n - 1} + 2n) < q_1(1 + 2n).$$

Game 4 [RANDOM]

In this last Game, all the nodes involved in challenge queries have been replaced by random functions. Hence, this game represents exactly the right-hand side probability in Definition 7. From the above sequence of games, we are able to compute a bound on the advantage of an attacker against this non-adaptive leakage-resilient encryption scheme.

$$\begin{array}{rcl} |\mathbb{P}[G_0] - \mathbb{P}[G_4]| & = & |\mathbb{P}[G_0] - \mathbb{P}[G_1] + \mathbb{P}[G_1] - \mathbb{P}[G_2] \\ & + & \mathbb{P}[G_2] - \mathbb{P}[G_3] + \mathbb{P}[G_3] - \mathbb{P}[G_4] \\ & \leqslant & |\mathbb{P}[G_0] - \mathbb{P}[G_1]| + |\mathbb{P}[G_1] - \mathbb{P}[G_2]| \\ & + & |\mathbb{P}[G_2] - \mathbb{P}[G_3]| + |\mathbb{P}[G_3] - \mathbb{P}[G_4]| \\ & \leqslant & 2^{3\lambda+1}(\varepsilon_{\phi_2} + q_0^2/2^{n+1} + 2\exp(-\varepsilon_{\phi_2}^2/8)) \cdot (v_0 - q_0) \\ & + & 2^{3\lambda+1}(\varepsilon_{\phi_3} + q_0^2/2^{n+1} + 2\exp(-\varepsilon_{\phi_3}^2/8)) \cdot q_0 \\ & + & (v_1 - q_1)\varepsilon_{\phi_2} + q_1\varepsilon_{\phi_3}. \end{array}$$

Note that the advantage of the attacker depends on the number of nodes involved in leakage and challenge queries. This number depends on the parameters of the skip-lists: the number of stages x and the number of children for each node l. In Figure 2.1, we chose to order the keys linearly but we could also have chosen to jump in the first stage by powers of two if it was relevant for our implementation. In any case, we can find appropriate parameters which maintain the advantage of the attacker negligible enough so that the whole encryption scheme is still secure. As illustrated in the proof, $x = n$ and $l = 2$ give interesting bounds. □

2.4.3 Security Analysis of Theorem 3

In 1995, Impagliazzo defined five complexity worlds [Imp95]: `algorithmica` in which P=NP with all the amazing consequences, `heuristica` world in which on the contrary NP-complete

problems are hard in the worst-case (P≠NP) but are efficiently solvable on average and three worlds on the existence of the cryptographic functions. In the `pessiland` world, there exist average-case NP-complete problems but one-way functions do not exist, which implies that we cannot generate hard instances of NP-complete problem with known solution. In the `minicrypt` world, one-way functions exist but public-key cryptographic schemes are impossible and finally in the `cryptomania` world, public-key cryptographic schemes exist and secure communication is possible. These worlds have been used positively to establish security proofs in many papers such as [Pie06, PS08, YS13].

In this section, we follow the work of Yu and Standaert who show in [YS13] how to improve the efficiency of our re-keying scheme, maintaining its leakage-resilient security in the `minicrypt` world. In fact, our new construction currently requires a large amount of fresh randomness since we need to generate a new fresh random value for each new session key. Yu and Standaert show that tweaking a similar design to use only a small amount of randomness can still be leakage-resilient in the world of `minicrypt`. That is, either the new design is leakage-resilient or it becomes possible to build public-key primitives from the involved symmetric-key blocks and the related leakage functions, which is very unlikely. Their technique directly applies to our symmetric encryption scheme and only requires a public seed s that is randomly chosen. Instead of being randomly generated, our public values p_i's and q_i's are now computed from a PRF G in counter mode.

Proof of Theorem 3 from [YS13]. The scheme is trivially secure if the seed is secret since it is like replacing the outputs of the PRF G by true random values. Let us now prove the leakage-resilient security when the seed is public. For this purpose, we assume by contradiction that there exists an adversary $\mathcal{A}$ against our scheme. If the scheme is not a naLR encryption scheme, there exists an adversary able to distinguish with a significant advantage the encryption of a real query from the encryption of a random string with the same size, given the previous leakage and outputs. Let us now consider a protocol between Alice and Bob who want to communicate over an authenticated channel. The protocol is a secure bit-agreement if the adversary, Eve, cannot recover the output bit of Alice. We construct it as follows:

1. Bob generates a random initial key for the re-keying scheme.

2. Alice generates the public random seed s and computes the required amount of public values using the PRF G. She sends these values to Bob.

3. Bob encrypts the message using the random values in the encryption scheme. He obtains the ciphertext c. He then generates a random bit b_B and sends to Alice either c if $b_B = 0$ or the encryption of a random value otherwise and in both cases the current view containing the leakage.

4. Alice finally fixes the bit b_A with the result of the distinction between the true output and the encryption of a random input.

As Eve only has access to the communication, she only gets knowledge of the intermediate public value (but not the seed), the current view and the correct or false result of the encryption. Hence, she cannot guess the bit b_A without breaking the scheme with secret seed. From the non negligible advantage of the adversary $\mathcal{A}$, the bit agreement that we established achieves correlation ($\mathbb{P}[b_A = b_B]$ is greater than $1/2$). As a consequence, this protocol is equivalent to a bit-PKE in which the secret key corresponds to the seed generated by Alice and the public key to the intermediate public values. □

2.5 Practical Aspects

In previous sections, we have shown that our construction instantiated with a weak PRF ϕ and combined with a PRF G yields a non-adaptive leakage-resilient encryption scheme. We now focus on the practical aspects.

2.5.1 Instantiation

Our encryption scheme is built with two primitives: a weak PRF ϕ for the derivation and a block cipher and a PRF G for the generation of random values.

Weak PRF ϕ

The concatenation of invocations of the weak PRF ϕ with random inputs is a suitable solution for the key derivation and the block cipher. Such a weak PRF can be built from any secure block cipher, like AES. Hence, inspired by [Pie09], we propose the constructions $\phi_2(k,p) = \mathrm{AES}(k, p\|0)\|\mathrm{AES}(k, p\|1)$, for the key derivation and $\phi_3(k,p,r) = \phi_2(k,p)\|\mathrm{AES}(k,r)$, for also the encryption which benefit from the reuse of one public random input.

PRF G

Following [YS13], G is instantiated with a secure block cipher, *e.g.,* the AES. Since the AES is already implemented for the weak PRF ϕ, this choice benefits from limiting the code size. As proved in [YS13], only $\log(1/\varepsilon)$ bits of fresh pseudo-randomness are required for each public value, with ε the security parameter of the weak PRF ϕ (e.g. AES). As a consequence, we only need one additional call of the AES every $\lfloor n/\log(1/\varepsilon)\rfloor$ invocations of ϕ.

2.5.2 Complexity Evaluation

Let us now focus on the complexity of encrypting a n-block message using our construction. We denote by τ_{AES} the complexity in time of one AES call either as a PRF for the re-keying or as a block cipher for the encryption. First, note that without updating the secret key and without any mode of operation, the complexity of the encryption is exactly $\mathcal{C} = n \cdot \tau_{\mathrm{AES}}$. Then, let us compute the same complexity in our leakage-resilient construction by first omitting the generation of randomness. For the sake of simplicity and because it is negligible, the complexity of the bitwise addition which is performed once per block encryption will be omitted as well. Furthermore, we will start with the initial key k_0 without loss of generality since what counts is the distance between the current index and the targeted one. We recall that the distance between two keys indexes from the same stage x_c is equal to $1 + x + \cdots + x^{x-x_c}$. We denote by N_x this distance which is also the number of children of a key from the same stage plus one. As a result, the number of AES executions N required to reach the key k_i is bounded as follows: $\frac{i}{N_1} \leqslant N \leq \frac{i}{N_1} + x(x-1)$ with $x(x-1)$ the maximum number of executions needed to reach a child from a first stage key. These bounds can be squeezed with the parameters related to the other stages. Table 2.1 presents the number of AES executions required to re-synchronize from k_0 to keys with increasing indexes. For comparison purpose, when the keys are updated sequentially, 10,000 invocations of the re-keying primitives are required to compute k_{10^4} from k_0. When using our construction with five stages, only $N = 20$ invocations are necessary that is five hundred times less. In the general case, one also needs to consider the generation of random values. Since the generation is also performed with the AES, the complexity of encrypting a n-block message is: $\mathcal{C} = (2N + 4n - 2)\tau_{AES}$ if we consider one invocation of G for each key

Table 2.1: Number of AES executions to derive a key from k_0 given its index

	k_{10}	k_{10^2}	k_{10^3}	k_{10^4}	k_{10^5}
#stages = 2	4	34	$3.3 \cdot 10^2$	$3.3 \cdot 10^3$	$3.3 \cdot 10^4$
#stages = 3	4	10	82	$7.7 \cdot 10^2$	$7.7 \cdot 10^3$
#stages = 4	6	8	16	$1.2 \cdot 10^2$	$1.2 \cdot 10^3$
#stages = 5	5	10	15	20	$1.4 \cdot 10^2$
sequential scheme	10	10^2	10^3	10^4	10^5

derivation and each block encryption. From [YS13], we can reduce the number of invocations of the generator until one every $\lfloor n/\log(1/\varepsilon) \rfloor$ invocations of ϕ, without loss of security:

$$\mathcal{C} = \left(N + 2n - 1 + \frac{N + 2n - 1}{\lfloor n/\log(1/\varepsilon) \rfloor} \right) \tau_{AES}.$$

2.6 Conclusion

In this chapter, we have tackled the problem of constructing an efficient and provably-secure symmetric encryption scheme based on re-keying ideas. In particular, we have first proven that a naLR naPRF combined with a block cipher, yields a non-adaptive leakage-resilient symmetric encryption scheme. Then, we have shown that such an encryption scheme does not actually require this level of security for its re-keying scheme. In fact, we have introduced a new re-keying process with relaxed security properties but which still yields a secure encryption scheme. Furthermore, it benefits from being much more efficient than a sequential scheme when both parts of the symmetric communication need to re-synchronize. We have both proven the security based on this new re-keying scheme and evaluated the global complexity.

This work shows that it is possible to use the security of the mode of operations in order to construct a leakage-resilient encryption scheme. One interesting future step to use another secure mode of operations such as OCB. Indeed, this mode is interesting since the adversary cannot know what is the real input of the block cipher and consequently classical DPA attack are thwarted. However, the security proof of this mode is a real challenge.

Chapter 3

Verifying Proofs of Higher-Order Masking

In the literature, several masking algorithms have been proposed. However, their security proofs are sometimes missing or false. Thus, in this chapter, we describe the construction of a tool which formally verifies the t-threshold security of masking schemes at a fixed order. Compared to previous formal tools, it verifies larger programs and supports higher orders.

Contents

3.1 Introduction

3.1.1 Motivation

In order to thwart DPA attacks (see Part I), the community has proposed many countermeasures but masking is probably the most widely used in the industrial context. To evaluate the security of masked implementations, the community has defined several leakage models that are discussed in Part I. In the most realistic one, the noisy leakage model, the device is assumed to leak noisy functions of the manipulated variables. However, this model is not convenient to build security proofs [PR13, DDF14]. The second very widely studied model is the t-threshold model in which the attacker is given the exact values of t intermediate variables of the implementation. This model benefits from being very convenient to establish security proofs (*e.g.,* [RP10,CPRR14]) but it is definitely less realistic. To benefit from the advantages of both models, Duc, Dziembowski and Faust proved in [DDF14] a reduction from the noisy leakage model to the t-threshold model. The theoretical bound is not tight but as shown by Duc, Faust and Standaert [DFS15a], the large factors are mostly proof artifacts and do not fit the reality of embedded devices. Hence, in the following, we will prove the security of masking schemes in the t-threshold model. Relying on the aforementioned reduction, the resulting proofs can be turned into proofs in the noisy leakage model.

In practice, most programs can easily be masked at the first order to prevent an adversary from recovering secrets using a single observation, and checking first-order masking schemes in the 1-threshold probing model is a relatively routine task since it is sufficient to check that each intermediate variable carries a distribution that is independent from the secret. However, even higher-order attacks, where t is greater than 1, have been conducted in practice [OM07,PSDQ05] and need to be protected against. Many masked implementations have been proposed to protect AES or its non-linear component, the Sbox (for example, [OMPR05, CB08, SP06, RP10, PR13]), among which some are also proved secure. Still, proving higher-order security manually in the t-threshold probing model is a much more difficult and error-prone task than it is for first-order. As a consequence, many published schemes were shown to be insecure, such as those presented by [SP06] and [RP10], broken in [CPR07] and [CPRR14]. In this chapter, we address this issue by developing automated methods to verify the security of algorithms masked at higher orders in the t-threshold probing model.

3.1.2 Related Work

Several methods which aim at proving the security of algorithms against power-analysis attacks have recently appeared. Among them, leakage detection methods seem to be strong techniques to detect potential first-order attacks. As presented and analyzed in [MOBW13], their idea is to exhibit all the points in consumption traces which may leak data-dependent information. Leakage detection thus benefits to captures any kind of first-order leakage without requiring any device profiling. In particular, while current techniques need to explicitly choose to consider value-based leakage or transition-based leakage model, leakage detection methods aim to directly analyze leakage points in consumption traces, whatever the number of intermediate variables they represent. Unfortunately, these methods also come with drawbacks. On the one hand, detecting data-dependent leakage on a single point requires important computations using a large set of consumption traces. On the other hand and as far as we know, current detection leakage techniques are dedicated to univariate scenario. That is, they can detect data-dependent leakage on one point but not on a combination of several points. Thus, while they are perfectly valid and useful to evaluate the security against first-order attacks, they cannot, at that time, evaluate the security of masking schemes against higher-order attacks.

Other tools have appeared to thwart these specific issues, at the cost of restriction on the targeted leakage model. More specifically, they aim at proving the security of masked algorithms in the t-threshold probing model [MOPT12, BRNI13, EWS14, EW14, EWTS14].

Some use type systems to propagate sensitivity marks along the programs, but such approaches are not complete [EWS14] and many programs are thus incorrectly typed as secure. Others take the underlying probability distributions into account, but can only handle low masking orders (typically orders 1 and 2), even on small programs.

3.1.3 Contributions

In this chapter, we develop automated methods to prove the security of masked implementations in the t-threshold probing model, both for value-based (*i.e.,* when a single variable leaks at a time) and transition-based leakage (*i.e.,* when two variables may leak at a time, see Part I for more details). More specifically, our theoretical contributions are three-fold: (*i*) We provide a formal characterization of security in the t-threshold probing model as a combination of variants of two well-known properties in programming languages: t-non-interference and functional equivalence; (*ii*) We provide algorithms that construct bijections between an adversary observation and a distribution that is trivially independent from the secret inputs, thereby proving that the adversary observation is independent from secret inputs; (*iii*) We provide algorithms that make use of the constructed bijections to extend sets of observations with additional observations that do not give the adversary any more information about the secrets, thereby reducing greatly the number of non-interference proofs that need to be performed in order to prove a whole program t-non-interfering.

As a practical contribution, we implement our algorithms and apply them to various masked implementations of AES, and a masked implementation of MAC-Keccak. Pleasingly, our tools are able to successfully analyze first-order masked implementations of AES (in a couple of minutes), 2 rounds of second-order masked implementations of AES (in around 22 minutes), and masked implementations of multiplication, up to order 5 (in 45 seconds). Our experiments allow us to rediscover several known attacks [CPR07, CPRR14] on flawed implementations, to check that proposed fixes, when they exist, are indeed secure, and finally to discover new attacks on flawed implementations [SP06]. We also discuss how our approach and tool can easily be adapted to deal with stronger leakage models capturing both transition-based leakage and leakage due to glitches, and illustrate it by studying the security of variants of secure field multiplication in the transition-based leakage model.

Putting our work in perspective, we deliberately focus on algorithmic methods that are able to cover large spaces of observation sets very efficiently, and without any assumption on the program. Although our results demonstrate that such methods can perform surprisingly well in practice, their inherent limitations with respect to scalability remain because of the absence of compositional properties (see Chapter 4). Therefore, our algorithmic methods can be seen as focusing primarily on proving that core functions are secure with respect to a widely-used notion of security.

3.1.4 Outline

We first review previous uses of formal methods to prove similar properties in Section 3.2. In Sections 3.3 and 3.4, we describe our algorithms. We give an example in Section 3.5. In Section 3.6, we evaluate the practicality of our approach by implementing our algorithms in the framework provided by EasyCrypt [BDG+14], and testing the performance of our implementation on representative examples from the literature.

3.2 Security in the t-Threshold Probing Model and Notion of t-non Interference

We rephrase security in the t-threshold probing model by defining the notion of *t-non interference*, which is based on the notions of probabilistic non-interference used for verifying information-flow properties in language-based security. We first define a general notion of program equivalence. Two probabilistic programs p_1 and p_2 are said to be $(\mathcal{I}, \mathcal{O})$-*equivalent*, whenever the probability distributions on $\mathcal{O}$ defined by p_1 and p_2, conditioned by the assumptions on input variables $\mathcal{I}$, are equal. This notion of equivalence subsumes two more specialized notions: *functional equivalence* and *t-non-interference*.

Definition 8. *Two programs p and $\bar{p}$ are said to be functionally equivalent when they are $(\mathcal{I}, \mathcal{Z})$-equivalent with $\mathcal{Z}$ all output variables, and $\mathcal{I}$ all input variables.*

Definition 9. *A program $\bar{p}$ is said to be t-non-interfering with respect to a set of secret input variables $\mathcal{I}_{\mathsf{sec}}$ and a set of* observations *$\mathcal{O}$ when $\bar{p}(s_0, \cdot)$ and $\bar{p}(s_1, \cdot)$ are $(\mathcal{I}_{\mathsf{pub}}, \mathcal{O})$-equivalent (with $\mathcal{I}_{\mathsf{pub}} = \mathcal{I} \setminus \mathcal{I}_{\mathsf{sec}}$ the set of non-secret input variables) for any values s_0 and s_1 of the secret input variables.*

Intuitively, a program is t-non-interfering with respect to a set of secret input variables $\mathcal{I}_{\mathsf{sec}}$ and a set of observations $\mathcal{O}$ whenever the distributions of the observations are independent from the secret inputs $\mathcal{I}_{\mathsf{sec}}$. In the following, when not explicitly mentioned, we describe our methods to evaluate the security in the value-based model, that is when an observation is exactly a single value. We discuss their adaptability in the transition-based model in Section 3.6. Furthermore, in this chapter, we mainly focus on the non-interference but the functional equivalence is addressed in [BBD+15b].

We now give an indistinguishability-based definition of the t-threshold probing model. In this model, the challenger randomly chooses two secret values s_0 and s_1 (representing for instance two different values of the secret key) and a bit b according to which the leakage will be produced: the output computation always uses secret s_0, but the adversary observations are computed using s_b. The adversary $\mathcal{A}$ is allowed to query an oracle with chosen instances of public arguments, along with a set of at most t intermediate variables (adaptively or non-adaptively chosen); such queries reveal their output and the values of the intermediate variables requested by the adversary. We say that $\mathcal{A}$ wins if she guesses b.

We now state the central theorem to our approach, heavily inspired by Duc, Dziembowski and Faust [DDF14] and Ishai, Sahai and Wagner [ISW03].

Theorem 4. *Let p and $\overline{p}$ be two programs. If p and $\overline{p}$ are functionally equivalent and $\overline{p}$ is t-non-interfering, then for every adversary $\mathcal{A}$ against $\overline{p}$ in the t-threshold probing model, there exists an adversary $\mathcal{S}$ against p in the black-box model, such that*

$$\Delta(\mathcal{S} \overset{bb}{\leftrightarrows} p, \mathcal{A} \overset{thr}{\leftrightarrows} \overline{p}) = 0$$

where $\Delta(\cdot\,;\cdot)$ denotes the statistical distance[1]*.*

Proof. Since p and $\overline{p}$ are functionally equivalent, we have $\Delta(\mathcal{S} \overset{bb}{\leftrightarrows} p, \mathcal{S} \overset{bb}{\leftrightarrows} \overline{p}) = 0$ for all black-box adversary $\mathcal{S}$, and we only have to prove that there exists an $\mathcal{S}$ such that $\Delta(\mathcal{S} \overset{bb}{\leftrightarrows} \overline{p}, \mathcal{A} \overset{thr}{\leftrightarrows} \overline{p}) = 0$.

[1] The theorem can be lifted to the noisy leakage model using Corollary 1 from [DDF14], using a small bound on the statistical distance instead.

We simply construct a simulator $\mathcal{S}'$ that simulates the leakage for $\mathcal{A}$, and build $\mathcal{S}$ by composing them. The simulator receives as inputs the public variables that are used for the execution of $\overline{p}$, and the output of $\overline{p}$, but not the t intermediate values corresponding to the observation set $\mathcal{O}$. Since $\overline{p}$ is t-non-interfering, the observations do not depend on the secret variables that are used for the execution of $\overline{p}$, and the simulator can choose arbitrary values for the secret variables, run $\overline{p}$ on these values and the public variables given as inputs, and output the requested observations. □

Following Theorem 4, we will propose algorithms to prove t-non-interference properties of probabilistic programs (functional equivalence is considered in [BBD+15c]), thereby reducing the security of masked implementations in the t-threshold probing model to the black-box security of the algorithms they implement.

Beforehand, we provide an overview of language-based techniques that could be used to verify the assumptions of Theorem 4, and to motivate the need for more efficient techniques. First, we introduce mild variants of two standard problems in programming languages, namely information-flow checking and equivalence checking, which formalize the assumptions of Theorem 4. Then, we present three prominent methods to address these problems: type systems (which are only applicable to information-flow checking), model counting, and relational logics. Finally, we discuss efficiency issues and justify the need for efficient techniques.

3.2.1 Problem Statement and Setting

Equivalence checking is a standard problem in program verification, although it is generally considered in the setting of deterministic programs, whereas we consider probabilistic programs here. Information-flow checking is a standard problem in language-based security, although it usually considers flows from secret inputs to public outputs, whereas we consider flows from secret inputs to intermediate values (*i.e.,* observations) here. Both problems can be construed as instances of relational verification. For clarity, we formalize this view in the simple case of straightline probabilistic programs. Such programs are sequences of random assignments and deterministic assignments, and have distinguished sets of input and output variables. Given a program p, we let $\mathsf{IVar}(p)$, $\mathsf{OVar}(p)$, and $\mathsf{PVar}(p)$ denote the sets of input, output, and intermediate variables of p. Without loss of generality, we assume that programs are written in single static assignment (SSA) form, and in particular, that program variables appear exactly once on the left hand side of an assignment, called their defining assignment—one can very easily transform an arbitrary straightline program into an equivalent straightline program in SSA form. Assuming that programs are in SSA form, we can partition $\mathsf{PVar}(p)$ into two sets $\mathsf{DVar}(p)$ and $\mathsf{RVar}(p)$ of deterministic and probabilistic variables, where a variable is probabilistic if it is defined by a probabilistic assignment, and is deterministic otherwise. Let $\mathcal{V}$ denote the set of program values (we ignore typing issues). Each program p can be interpreted as a function:

$$[\![p]\!] : \mathcal{D}(\mathcal{V}^{\kappa}) \to \mathcal{D}(\mathcal{V}^{\ell+\ell'})$$

where $\mathcal{D}(T)$ denotes the set of discrete distributions over a set T, and κ, ℓ and ℓ' respectively denote the sizes of $\mathsf{IVar}(p)$, $\mathsf{PVar}(p)$ and $\mathsf{OVar}(p)$. The function $[\![p]\!]$ takes as input a joint distribution on input variables and returns a joint distribution on all program variables, and is defined inductively in the expected way. Furthermore, one can define for every set of observations $\mathcal{O}$ of size m a function:

$$[\![p]\!]_{\mathcal{O}} : \mathcal{D}(\mathcal{V}^{\kappa}) \to \mathcal{D}(\mathcal{V}^{m})$$

that computes, for each $\boldsymbol{v} \in \mathcal{V}^{\kappa}$, the marginal distribution of $[\![p]\!](\boldsymbol{v})$ with respect to $\mathcal{O}$.

We can now define the information-flow checking problem formally. A program p is non-interfering with respect to a pre-condition $\Phi \subseteq \mathcal{D}(\mathcal{V}^\kappa) \times \mathcal{D}(\mathcal{V}^\kappa)$ and a set of observations $\mathcal{O}$, or $(\Phi, \mathcal{O})$-non-interfering, iff $[\![p]\!]_\mathcal{O}(\mu_1) = [\![p]\!]_\mathcal{O}(\mu_2)$ for every $\mu_1, \mu_2 \in \mathcal{D}(\mathcal{V}^\kappa)$ such that $(\mu_1, \mu_2) \in \Phi$. In our context, $(\mu_1, \mu_2) \in \Phi$ iff the marginal distributions μ_1 and μ_2 are equal with respect to public variables.

There is a direct interpretation of t-threshold probing security as a non-interference property. We say that a program p is (Φ, t)-non-interfering if it is $(\Phi, \mathcal{O})$-non-interfering for all observations sets $\mathcal{O}$ with size smaller than t (we write $\mathcal{O} \in \mathcal{P}_{\leq t}(\mathsf{PVar}(p))$ in the following). Then a program p is secure in the t-threshold probing model (with respect to a pre-condition Φ) iff it is (Φ, t)-non-interfering.

In order to capture t-threshold probing security in the transition-based leakage model, we rely on a partial function next that maps program variables to their successors. For programs that have been translated into SSA form, all program variables are of the form x_i, where x is a variable of the original program, and i is an index—typically a program line number. The successor of such a variable x_i, when it exists, is a variable of the form x_j where j is the smallest index such that $i < j$ and x_j is a program variable. Then, we say that a program p is (Φ, t)-non-interfering in the transition-based model, iff p is $(\Phi, \mathcal{O} \cup \mathsf{next}(\mathcal{O}))$-non-interfering for every set of observations $\mathcal{O}$ with size smaller than t. Then a program p is secure in the transition-based t-threshold probing model (with respect to a pre-condition Φ) iff it is (Φ, t)-non-interfering in the transition-based model.[2]

We now turn to program equivalence. For the sake of simplicity, we consider two programs p_1 and p_2 that have the same sets of input and output variables (*e.g.,* a cryptographic program and its masked version); we let $\mathcal{W}$ denote the latter. We let $[\![p]\!]_\mathcal{W}$ denote the function that computes for every initial distribution μ the marginal distribution of $[\![p]\!](\mu)$ with respect to $\mathcal{W}$. We say that p_1 and p_2 are equivalent with respect to a pre-condition Φ, iff $[\![p_1]\!]_\mathcal{W}(\mu) = [\![p_2]\!]_\mathcal{W}(\mu)$ for every distribution μ such that $(\mu, \mu) \in \Phi$.

For the sake of completeness, we point out that both notions are subsumed by the notion of $(\Phi, \mathcal{O})$-equivalence. Specifically, we say that programs p_1 and p_2 are $(\Phi, \mathcal{O})$-equivalent, iff $[\![p_1]\!]_\mathcal{O}(\mu_1) = [\![p_2]\!]_\mathcal{O}(\mu_2)$ for every two distributions μ_1 and μ_2 such that $(\mu_1, \mu_2) \in \Phi$. Therefore, both equivalence checking and information-flow checking can be implemented using as subroutine any sound algorithm for verifying that p_1 and p_2 are $(\Phi, \mathcal{O})$-equivalent.

3.2.2 Type-Based Approaches

Information-flow type systems are a class of type systems that enforce non-interference by tracking dependencies between program variables and rejecting programs containing illicit flows. There are multiple notions of non-interference (termination-sensitive, termination-insensitive, or bisimulation-based) and forms of information-flow type systems (for instance, flow-sensitive, or flow-insensitive); we refer the reader to [SM03] for a survey. For the purpose of this paper, it is sufficient to know that information-flow type systems for deterministic programs assign to all program variables a level drawn from a lattice of security levels which includes a level of public variables and secret variables. In the same vein, one can develop information-flow type systems to enforce probabilistic non-interference; broadly speaking, such type systems distinguish between public values, secret values, and uniformly distributed values. Following these

[2] Similarly, glitches could be captured by considering that each observation leaks four values: the values of the arguments, and the old and new values of the wire or register. More fine-grained leakage models depending on implementation details and combining value-based, transition-based and glitch-based leakage could also be considered.

ideas, Moss *et al.* [MOPT12] pioneer the application of information-flow type systems to masking. They use the type system as a central part in a masking compiler that transforms an input program into a functionally equivalent program that is resistant to first-order DPA. Their technique can readily be extended to prove non-interference with respect to a single observation set.

Because they are implemented with well-understood tools (such as data flow analyses) and are able to handle large programs extremely fast, information-flow type systems provide an appealing solution that one would like to use for higher-order DPA. However, the semantic information carried by types is inherently attached to individual values, rather than tuples of values, and there is no immediately obvious way to devise an information-flow type system even for second-order DPA. Notwithstanding, it is relatively easy to devise a sound method for verifying resistance to higher-order DPA using an information-flow type system in the style of [MOPT12]. The basic idea is to instrument the code of the original program with assignments $w := x_1 \parallel \ldots \parallel x_t$, where w is a fresh program variable, $x_1 \ldots x_t$ are variables of the original program, and t is the order for which resistance is sought; we let p' denote the instrumented program. Clearly, a program p is secure at order t iff for every initial values $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, $[\![p']\!]_{\{w\}}(\boldsymbol{v_1}) = [\![p']\!]_{\{w\}}(\boldsymbol{v_2})$ where w ranges over the set of fresh variables that have been introduced by the transformation. It is then possible to use an information-flow type system in the spirit of [MOPT12] to verify that c' satisfies non-interference with respect to output set $\{w\}$. However, this transformational approach suffers from two shortcomings: first, a more elaborate type system is required for handling concatenation with sufficient accuracy; second, and more critically, the transformation induces an exponential blow-up in the size of programs.

In a slightly different context, Pettai and Laud [PL14] use a type-system to prove non-interference of a limited number of adversary observations imposed by their adversary model in the multi-party computation scenario. They do so by propagating information regarding linear dependencies on random variables throughout their arithmetic circuits and progressively replacing subcircuits with random gates. Because of the limited number of possible adversary observations their model imposes, they do not run into the same scalability issues we deal with in this paper. However, their techniques for dealing with active adversaries may be useful for verifying masking-based countermeasures in the presence of fault injection attacks.

3.2.3 SMT-Based Methods

There have been a number of works that use SMT solvers to achieve more flexible analysis of masked implementations.

Bayrak *et al.* [BRNI13] develop an SMT-based method for analyzing the sensitivity of sequences of operations. Informally, the notion of sensitivity characterizes whether a variable used to store an intermediate computation in the sequence of operations depends on a secret and is statistically independent from random variables. Their approach is specialized to first-order masking, and suffers from some scalability issue, in particular, they report analysis of a single round of AES.

Eldib, Wang and Schaumont develop an alternative tool, SCSniffer [EWS14], that is able to analyze masked implementations at orders 1 and 2. Their approach is based on model counting [GSS09]: to prove that a set of probabilistic expressions is distributed independently from a set of secrets, model-counting-based tools count the number of valuations of the secrets that yield each possible value of the observed expressions and checks that that number is indeed independent from the secret. This process in itself is inherently exponential in the size of the observed expressions, even when only one such observation is considered. To overcome this issue,

SCSniffer implements an incremental approach for reducing the size of such expressions when they contain randomness that is syntactically independent from the rest of the program. This incremental approach is essential to analyzing some of their examples, but it is still insufficient for analyzing complete implementations: for instance, SCSniffer can only analyze one round of (MAC-)Keccak whereas our approach is able to analyze the full 24 rounds of the permutation. The additional power of our tool is derived from our novel technique: instead of explicitly counting solutions to large boolean systems, our tool simply constructs a bijection between two distributions, one of which is syntactically independent from the secrets. Although the complexity of this process still depends on the size of expressions (and in particular in the number of randomly sampled variables they contain), it is only polynomial in it, rather than exponential. In addition, the approach, as it is used in Sleuth and SCSniffer, is limited to the study of boolean programs or circuits, where all variables are 1 bit in size. This leads to unwieldy program descriptions and artificially increases the size of expressions, thereby also artificially increasing the complexity of the problem. Our approach bypasses this issue by considering abstract algebraic expressions rather than specific types. This is only possible because we forego explicit solution counting. Moreover, SCSniffer requires to run the tool at all orders $d \leq t$ to obtain security at level t. In contrast, we achieve the same guarantees in a single run. This is due to the fact that the exclusive-or of observed variables is used for model counting rather than their joint distribution. Our approach yields proofs of t-non-interference directly by considering the joint distribution of observed variables. Finally, we contribute a technique that helps to reduce the practical complexity of the problem by extending proofs of independence for a given observation set into a proof of independence for many observation sets at once. This process is made less costly by the fact that we can efficiently check whether a proof of independence is still valid for an extended observation set, but we believe it would apply to techniques based on model-counting given the same ability.

All of these differences lead to our techniques greatly outperforming existing approaches when it comes to practical examples. For example, even considering only masking at order 1, where it takes SCSniffer 10 minutes to prove a masked implementation of one round of Keccak (implemented bit-by-bit), it takes our tool around 7 minutes to prove the full 24 rounds of the permutation (implemented on 64-bit words as in reference implementations), and around 2 minutes to verify a full implementation of AES (including its key schedule).

3.2.4 Relational Verification

A more elaborate approach is to use program verification for proving non-interference and equivalence of programs. Because these properties are inherently relational—that is, they either consider two programs or two executions of the same program—the natural verification framework to establish such properties is relational program logic. Motivated by applications to cryptography, Barthe, Grégoire and Zanella-Béguelin [BGZB09] introduce pRHL, a probabilistic Relational Hoare Logic that is specifically tailored for the class of probabilistic programs considered in this paper. Using pRHL, $(\phi, \mathcal{O})$-non-interference a program p is captured by the pRHL judgment:

$$\{\phi\}p \sim p\{\bigwedge_{y \in \mathcal{O}} y\langle 1\rangle = y\langle 2\rangle\}$$

which informally states that the joint distributions of the variables $y \in \mathcal{O}$ coincide on any two executions (which is captured by the logical formula $y\langle 1\rangle = y\langle 2\rangle$) that start from initial memories related by ϕ.

Barthe *et al.* [BGHZ11] propose an automated method to verify the validity of such judgments. For clarity of our exposition, we consider the case where p is a straightline code program.

The approach proceeds in three steps:

1. transform the program p into a semantically equivalent program which performs a sequence of random assignments, and then a sequence of deterministic assignments. The program transformation repeatedly applies eager sampling to pull all probabilistic assignments up-front. At this stage, the judgement is of the form

$$\{\phi\}S; D \sim S; D\{\bigwedge_{y\in\mathcal{O}} y\langle 1\rangle = y\langle 2\rangle\}$$

   where S is a sequence of probabilistic assignments, and D is a sequence of deterministic assignments;

2. apply a relational weakest precondition calculus to the deterministic sequence of assignments; at this point, the judgment is of the form

$$\{\phi\}S \sim S\{\bigwedge_{y\in\mathcal{O}} e_y\langle 1\rangle = e_y\langle 2\rangle\}$$

   where e_y is an expression that depends only on the variables sampled in S and on the program inputs;

3. repeatedly apply the rule for random sampling to generate a verification condition that can be discharged by SMT solvers. Informally, the rule for random sampling requires finding a bijection between the domains of the distribution from which values are drawn, and proving that a formula derived from the post-condition is valid. We refer to [BGZB09] and [BGHZ11] for a detailed explanation of the rule for random sampling. For our purposes, it is sufficient to consider a specialized logic for reasoning about the validity of judgments of the form above. We describe such a logic in Section 3.3.1.

Note that there is a mismatch between the definition of (Φ, t)-non-interference used to model security in the t-threshold probing model, and the notion of $(\phi, \mathcal{O})$-non-interference modeled by pRHL. In the former, Φ is a relation over distributions on memories, whereas in the latter ϕ is a relation over memories. There are two possible approaches to address this problem: the first is to develop a variant of pRHL that supports a richer language of assertions; while possible, the resulting logic might not be amenable to automation. A more pragmatic solution, which we adopt in our tool, is to transform the program p into a program $i; p$, where i is some initialization step, such that p is $(\Phi, \mathcal{O})$ non-interfering iff $i; p$ is $(\phi, \mathcal{O})$ non-interfering for some pre-condition ϕ derived from Φ. In particular, i includes code marked as non-observable that *preshares* any input or state marked as secret,[3] and fully observable code that simply shares public inputs. The code for sharing and presharing, as well as a simple example of this transformation are given in Section 3.5.

Notation. *In the following, we omit Φ in the definition of t-non-interference since, in the verification of masking schemes, we only consider the equality of distributions with respect to public variables as a pre-condition.*

3.3 A Logic for Probabilistic Non-Interference

Existing verification-based techniques to prove probabilistic non-interference statements face at least efficiency issues. Thus, we propose hereafter new efficient methods. We first introduce a specialized logic to prove a vector of probabilistic expressions independent from some secret variables. Then, we describe a simple algorithm that soundly constructs derivations in our logic.

[3]This corresponds to Ishai, Sahai and Wagner's *input encoders* [ISW03].

3.3.1 Our Logic for Probabilistic Non-Interference

Our logic shares many similarities with the equational logic developed in [BDK+10] to reason about equality of distributions. In particular, it considers equational theories over multi-sorted signatures. A multi-sorted signature is defined by a set of types and a set of operators. Each operator has a signature $\sigma_1 \times \ldots \times \sigma_n \to \tau$, which determines the type of its arguments, and the type of the result. We assume that some operators are declared as invertible with respect to one or several of their arguments; informally, a k-ary operator f is *invertible with respect to its i-th argument*, or *i-invertible* for short, if, for any $(x_j)_{j\neq i}$ the function $f(x_0, \ldots, x_{i-1}, \cdot, x_{i+1}, \ldots, x_k)$ is a bijection. If f is i-invertible, we say that its i-th argument is an *invertible argument of* f. Expressions of program's intermediate variables are built inductively from two sets $\mathcal{R}$ (fresh random values) and $\mathcal{X}$ (*e.g.,* secret and public inputs) of probabilistic and deterministic variables respectively, and from operators. Expressions are (strongly) typed. The set of deterministic (resp. probabilistic) variables of a vector of expressions $\boldsymbol{e}$ is denoted as $\mathsf{dvar}(\boldsymbol{e})$ (resp. $\mathsf{rvar}(\boldsymbol{e})$). We say that an expression e is *invertible in random variable x* whenever there exist families (f_j) and (e_i^j) such that $e = f_1(\ldots, e_{i_1-1}^1, f_2(\ldots f_n(\ldots, e_{i_n-1}^n, x, \ldots)\ldots), \ldots)$, each f_j is i_j-invertible and $\forall i\ j,\ x \notin \mathsf{rvar}(e_i^j)$.

We equip expressions with an equational theory $\mathcal{E}$. An equational theory is a set of equations, where an equation is a pair of expressions of the same type. Two expressions e and e' are provably equal with respect to an equational theory $\mathcal{E}$, written $e \doteq_{\mathcal{E}} e'$, if the equation $e \doteq_{\mathcal{E}} e'$ can be derived from the standard rules of multi-sorted equational logic: reflexivity, symmetry, transitivity, congruence, and instantiation of axioms in $\mathcal{E}$. Such axioms can be used, for example, to equip types with particular algebraic structures.

We now describe the three rules we define to prove that a set of expressions $\boldsymbol{e}$ is independent from the secret inputs $\mathcal{I}_{\mathsf{sec}}$. Rule (Indep) states that $\boldsymbol{e}$ is non-interfering whenever all the deterministic variables involved in its expressions are public. Rule (Equ) states that one can replace expressions by other expressions that are provably equivalent with respect to the equational theory $\mathcal{E}$. Rule (Opt) states that, whenever the only occurrences of a random variable r in $\boldsymbol{e}$ are as the i-th argument of some fixed application of an i-invertible operator f where f's other arguments are some $(e_j)_{j\neq i}$, then it is sufficient to prove the non-interference of a set where r is substituted for $f(e_0, \ldots, e_{i-1}, r, e_{i+1}, \ldots, e_k)$ in $\boldsymbol{e}$. The soundness of rule (Opt) becomes clear by remarking that the distributions $[\![f(e_0, \ldots, e_{i-1}, r, e_{i+1}, \ldots, e_k)]\!]$ and $[\![r]\!]$ are equal, since f is i-invertible and r is uniform random and does not appear in any of the e_j. Although we could use further rules (see, for example [BDK+10]), these three rules are in fact sufficient for our purposes.

3.3.2 Our Algorithm

We now describe an algorithm that takes a set of expressions (intermediate variables) $\boldsymbol{e}$ and a boolean b as inputs and applies the three rules to determine if the set of expressions is non-interfering with respect to the fresh random variables $\mathcal{R}$ and the secret input variables $\mathcal{I}_{\mathsf{sec}}$. In the following, we make use of unspecified choose algorithms that, given a set X, return an $x \in X$ or $\perp$ if $X = \emptyset$. We discuss our chosen instantiations where valuable.

Our algorithm (Algorithm 2) works using the three rules (Indep), (Opt) and (Equ) of the logic. Until (Indep) applies, Algorithm 2 tries to find $(\boldsymbol{e}', e, r)$ such that r is a fresh random value, e is invertible in r and $\boldsymbol{e} = \boldsymbol{e}'[e/r]$; if it succeeds, it then performs a recursive call on $\boldsymbol{e}'$. When it cannot find a suitable $(\boldsymbol{e}', e, r)$, the idea is to normalize algebraic expressions as described in [ABG+14], simplifying expressions and perhaps revealing potential applications of the (Opt) rule. We use only algebraic normalization to avoid the need for user-provided hints,

and even then, only use this restricted version of the (EQU) rule as a last resort. This is for two reasons: first, ring normalization may prevent the use of some $(\boldsymbol{e}', e, r)$ triples in later recursive calls (for example, the expression $(a \oplus r) \cdot r'$ gets normalized as $a \cdot r' \oplus r \cdot r'$, which prevents the substitution of $a \oplus r$ by r); second, the normalization can be costly and negatively impact performance. The boolean b indicates wether a simplication has already been applied ($false$) or not ($true$).

Algorithm 2 Proving Probabilistic Non-Interference

```
1: function NI_{R,I_sec}(e, b)                ▷ the joint distribution of e is independent from I_sec
2:     if ∀x ∈ dvar(e). x ∉ I_sec then
3:         return INDEP
4:     (e', e, r) ← choose({(e', e, r) | e is invertible in r ∧ r ∈ R ∧ e = e'[e/r]})
5:     if (e', e, r) ≠ ⊥ then
6:         return OPT(e, r) : NI_{R,I_sec}(e', b)
7:     else if b then
8:         e ← ring_simplify(e)
9:         return EQU : NI_{R,I_sec}(e, false)
10:    return ⊥
```

In practice, we have not found any examples where Algorithm 2 fails to prove the security of a secure implementation. In the following, we use $\mathsf{NI}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(X)$ the function from Algorithm 2 with b initially true. In particular, the implementation described and evaluated in Section 3.6 relies on this algorithm.

3.4 Divide-and-Conquer Algorithms Based on Large Sets

Even with efficient algorithms to prove that a program p is t-non-interfering for an observation set $\mathcal{O}$, proving that p is t-non-interfering for every possible observation set remains a complex task. Indeed this involves proving $\mathsf{NI}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e})$ for all the sets of variables $\boldsymbol{e}$ corresponding to at most t observations. Simply enumerating all possible observation sets quickly becomes intractable as p and t grow. Our main idea to solve this problem is based on the following fact: if p is $(\mathcal{O})$-non-interfering then for every $\mathcal{O}' \subseteq \mathcal{O}$ p is $(\mathcal{O}')$-non-interfering. Therefore checking that p is $(\mathcal{O}_i)$-non-interfering for every i can be done in a single step by checking that p is $(\bigcup_i \mathcal{O}_i)$-non-interfering.

Our goal is therefore to find fewer, larger observation sets $\mathcal{O}_1, \ldots, \mathcal{O}_k$ such that for each $1 \leq i \leq k$, p is $(\mathcal{O}_i)$-non-interfering and such that for all set of at most t observations, $\mathcal{O}$ is a subset of at least one of the $\mathcal{O}_i$. Since this last condition is the contraposite of the Hitting Set problem [GJ79], which is known to be NP-hard, we do not expect to find a generally efficient solution, and focus on proposing algorithms that prove efficient in practice.

We describe and implement several algorithms based on the observation that the sequences of derivations constructed in Algorithm 2 can be used to efficiently extend the observation sets with additional observations whose joint distributions with the existing ones is still independent from the secrets. We first present algorithms that perform such extensions, and others that make use of observation sets extended in this way to find a family $\mathcal{O}_1, \ldots, \mathcal{O}_k$ of observation sets that fulfill the condition above with k as small as possible.

3.4.1 Extending Safe Observation Sets

The $\mathsf{NI}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}$ algorithm from Section 3.2 (Algorithm 2) determines, for sets X of expressions, if their joint distribution is independent from the secret inputs (*i.e.,* $\mathcal{I}_{\mathsf{sec}}$). We now want to extend such an X into a set X' that may contain more observable expressions and such that the joint distribution of X' is still independent from variables in $\mathcal{I}_{\mathsf{sec}}$.

First, we define Algorithm 3, which rechecks that a derivation $\boldsymbol{h}$ (*i.e.,* a succession of rules), returned by Algorithm 2, applies to a given set of expressions $\boldsymbol{e}$. The algorithm simply checks that the consecutive rules encoded by $\boldsymbol{h}$ can be applied on $\boldsymbol{e}$. A key observation is that if $\mathsf{NI}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e}) = \boldsymbol{h}$ then $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e},\boldsymbol{h})$. Furthermore, if $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e},\boldsymbol{h})$ and $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e}',\boldsymbol{h})$ then $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e}\cup\boldsymbol{e}',\boldsymbol{h})$.

Algorithm 3 Rechecking a derivation

function $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e},\boldsymbol{h})$ ▷ Check that the derivation represented by $\boldsymbol{h}$ can be applied to $\boldsymbol{e}$

- **if** $\boldsymbol{h} = \textsc{Indep}$ **then**
  - **return** $\forall x \in \mathsf{dvar}(\boldsymbol{e}).\ x \notin \mathcal{I}_{\mathsf{sec}}$
- **if** $\boldsymbol{h} = \textsc{Opt}(e,r) : \boldsymbol{h'}$ **then**
  - $(\boldsymbol{e'}) \leftarrow \mathsf{choose}(\{\boldsymbol{e'} \mid \boldsymbol{e} = \boldsymbol{e'}[e/r]\})$
  - **if** $\boldsymbol{e'} \neq \bot$ **then**
    - **return** $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e'},\boldsymbol{h'})$
- **if** $\boldsymbol{h} = \textsc{Equ} : \boldsymbol{h'}$ **then**
  - $\boldsymbol{e} \leftarrow \mathsf{ring\_simplify}(\boldsymbol{e})$
  - **return** $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{e},\boldsymbol{h'})$

Secondly, we consider (as Algorithm 4) an extension operation that only adds expressions on which $\boldsymbol{h}$ can safely be applied as it is.

Algorithm 4 Extending the Observation using a Fixed Derivation

function $\mathsf{extend}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{x},\boldsymbol{e},\boldsymbol{h})$ ▷ find $\boldsymbol{x'}$ such that $\boldsymbol{x} \subseteq \boldsymbol{x'} \subseteq \boldsymbol{e}$ and $\boldsymbol{h}(\boldsymbol{x'})$ is syntactically independent from $\mathcal{I}_{\mathsf{sec}}$

- $e \leftarrow \mathsf{choose}(\boldsymbol{e})$
- **if** $\mathsf{recheck}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(e,\boldsymbol{h})$ **then**
  - **return** $\mathsf{extend}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}((\boldsymbol{x},e),\boldsymbol{e}\setminus\{e\},\boldsymbol{h})$
- **else**
  - **return** $\mathsf{extend}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{x},\boldsymbol{e}\setminus\{e\},\boldsymbol{h})$

We also consider an algorithm that extends a set $\boldsymbol{x}$ with elements in $\boldsymbol{e}$ following $\boldsymbol{h}$ whilst also extending the derivation itself when needed. However, this algorithm induces a loss of performance due to the low proportion of program variables that can in fact be used to extend the observation set, wasting effort on attempting to extend the derivation when it was not in fact possible. Coming up with an efficient choose algorithm that prioritizes variables that are likely to be successfully added to the observation set is an interesting challenge that would refine this algorithm, and thus improve the performance of the space splitting algorithms we discuss next.

In the following, we use $\mathsf{extend}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{x},\boldsymbol{e},\boldsymbol{h})$ to denote the function from Algorithm 4, which is used to obtain all experimental results reported in Section 3.6.

3.4.2 Splitting the Space of Adversary Observations

Equipped with an efficient observation set extension algorithm, we can now attempt to accelerate the coverage of all possible sets of adversary observations to prove t-non-interference. The general idea of these coverage algorithms is to choose a set X of t observations and prove that the program is non-interfering with respect to X, then use the resulting derivation witness to efficiently extend X into an $\widehat{X}$ that contains (hopefully many) more variables. This $\widehat{X}$, with respect to which the program is known to be non-interfering, can then be used to split the search space recursively. We consider hereafter two splitting strategies to accelerate the enumeration: the first (Algorithm 5) simply splits the observation space into $\widehat{X}$ and its complement before covering observations that straddle the two sets. The second (Algorithm 6) splits the space many-ways, considering all possible combinations of the sub-spaces when merging the sets resulting from recursive calls.

Pairwise Space-Splitting

Our first algorithm (Algorithm 5) uses its initial tuple X to split the space into two disjoint sets of observations, recursively descending into the one that does not supersede X and calling itself recursively to merge the two sets once they are processed separately.

Algorithm 5 Pairwise Space-Splitting

```
1: function check_{R,I_sec}(x, d, e)        ▷ every x, y with y ∈ P_{≤d}(e) is independent of I_sec
2:     if d ≤ |e| then
3:         y ← choose(P_{≤d}(e))
4:         h_{x,y} ← NI_{R,I_sec}((x, y))        ▷ if NI_{R,I_sec} fails, raise error CannotProve (x, y)
5:         ŷ ← extend_{R,I_sec}(y, e \ y, h_{x,y})        ▷ if h_{x,y} = ⊤, use ŷ = y
6:         check_{R,I_sec}(x, d, e \ ŷ)
7:         for 0 < i < d do
8:             for u ∈ P_{≤i}(ŷ) do
9:                 check_{R,I_sec}((x, u), d − i, e \ ŷ)
```

Theorem 5 (Soundness of Pairwise Space-Splitting)**.** *Given a set $\mathcal{R}$ of random variables, a set $\mathcal{I}_{\mathsf{sec}}$ of secret variables, a set of expressions $\boldsymbol{e}$ and an integer $t > 0$, if $\mathsf{check}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\emptyset, t, \boldsymbol{e})$ succeeds then every combination of less than t intermediate variables is independent from $\mathcal{I}_{\mathsf{sec}}$.*

Proof. The proof is by generalizing on $\boldsymbol{x}$ and d and by strong induction on $\boldsymbol{e}$. If $|\boldsymbol{e}| < d$, the theorem is vacuously true, and this base case is eventually reached since $\widehat{\boldsymbol{y}}$ contains at least d elements. Otherwise, by induction hypothesis, the algorithm is sound for every $\boldsymbol{e}' \subsetneq \boldsymbol{e}$. After line 5, we know that all t-tuples of variables in $\widehat{\boldsymbol{y}}$ are independent, jointly with $\boldsymbol{x}$, from the secrets. By the induction hypothesis, after line 6, we know that all t-tuples of variables in $\boldsymbol{e} \setminus \widehat{\boldsymbol{y}}$ are independent, jointly with $\boldsymbol{x}$, from the secrets. It remains to prove the property for t-tuples that have some elements in $\widehat{\boldsymbol{y}}$ and some elements in $\boldsymbol{e} \setminus \widehat{\boldsymbol{y}}$. The nested for loops at lines 7-9 guarantee it using the induction hypothesis. □

Worklist-Based Space-Splitting

Our second algorithm (Algorithm 6) splits the space much more finely given an extended safe observation set. The algorithm works with a worklist of pairs $(d, \boldsymbol{e})$ (initially called with a single element $(t, \mathcal{P}_{\leq t}(\mathsf{PVar}(p)))$). Unless otherwise specified, we lift algorithms seen so far to work with vectors or sets of arguments by applying them element by element. Note in particular, that

the for loop at line 7 iterates over all vectors of n integers such that each element i_j is strictly between 0 and d_j.

Algorithm 6 Worklist-Based Space-Splitting

1: **function** $\mathsf{check}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}((d_j, \boldsymbol{e}_j)_{0\leq j<n})$ ▷ every $\boldsymbol{x} = \bigcup_{0\leq j<n} \boldsymbol{x}_j$ with $\boldsymbol{x}_j \in \mathcal{P}_{\leq d_j}(\boldsymbol{e}_j)$ is independent from $\mathcal{I}_{\mathsf{sec}}$
2: **if** $\forall j,\ d_j \leq |\boldsymbol{e}_j|$ **then**
3: $\boldsymbol{y}_j \leftarrow \mathsf{choose}(\mathcal{P}_{\leq d_j}(\boldsymbol{e}_j))$
4: $\boldsymbol{h} \leftarrow \mathsf{NI}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\bigcup_{0\leq j<n} \boldsymbol{y}_j)$ ▷ if $\mathsf{NI}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}$ fails, raise error $\mathsf{CannotProve}$ $(\bigcup \boldsymbol{y}_j)$
5: $\widehat{\boldsymbol{y}_j} \leftarrow \mathsf{extend}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(\boldsymbol{y}_j, \boldsymbol{e}_j \setminus \boldsymbol{y}_j, \boldsymbol{h})$
6: $\mathsf{check}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}((d_j, \boldsymbol{e}_j \setminus \widehat{\boldsymbol{y}_j})_{0\leq j<n})$
7: **for** j; $0 < i_j < d_j$ **do**
8: $\mathsf{check}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}(i_j, (\widehat{\boldsymbol{y}_j}, d_j - i_j, \boldsymbol{e}_j \setminus \widehat{\boldsymbol{y}_j}))$

Theorem 6 (Soundness of Worklist-Based Space-Splitting)**.** *Given a set $\mathcal{R}$ of random variables, a set $\mathcal{I}_{\mathsf{sec}}$ of secret variables, a set of expressions $\boldsymbol{e}$ and an integer $t > 0$, if $\mathsf{check}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}((t, \boldsymbol{e}))$ succeeds then every $\boldsymbol{x} \in \mathcal{P}_{\leq t}(\boldsymbol{e})$ is independent from $\mathcal{I}_{\mathsf{sec}}$.*

Proof. As in the proof of Theorem 5, we start by generalizing, and we prove that, for all vector $(d_j, \boldsymbol{e}_j)$ with $0 < d_j$ for all j, if $\mathsf{check}_{\mathcal{R},\mathcal{I}_{\mathsf{sec}}}((d_j, \boldsymbol{e}_j))$ succeeds, then every $\boldsymbol{x} = \bigcup_{0\leq j<n} \boldsymbol{x}_j$ with $\boldsymbol{x}_j \in \mathcal{P}_{\leq d_j}(\boldsymbol{e}_j)$ is independent from $\mathcal{I}_{\mathsf{sec}}$. The proof is again by strong induction on the vectors, using an element-wise lexicographic order (using size order on the $\boldsymbol{e}$) and lifting it to multisets as a bag order. If there exists an index i for which $|\boldsymbol{e}_i| < d_i$, the theorem is vacuously true. Otherwise, we unroll the algorithm in a manner similar to that in Theorem 5. After line 5, we know that, for every j, every $\boldsymbol{x} \in \mathcal{P}_{\leq d_j}(\widehat{\boldsymbol{y}_j})$ is independent from x_H. After line 6, by induction hypothesis (for all j, $\#\boldsymbol{e}_j \setminus \widehat{\boldsymbol{y}_j} < \#\boldsymbol{e}_j$ since $\widehat{\boldsymbol{y}_j}$ is of size at least d_j), we know that this is also the case for every $\boldsymbol{x} \in \mathcal{P}_{\leq d_j}(\widehat{\boldsymbol{y}_j})$. Remains to prove that every subset of $\boldsymbol{e}_j$ of size d_j that has some elements in $\widehat{\boldsymbol{y}_j}$ and some elements outside of it is also independent from $\mathcal{I}_{\mathsf{sec}}$. This is dealt with by the for loop on lines 7-8, which covers all possible combinations to recombine $\boldsymbol{y}_j$ and its complement, in parallel for all j. □

Comparison

Both algorithms lead to significant improvements in the verification time compared to the naive method which enumerates all t-tuples of observations for a given implementation. Further, our divide-and-conquer strategies make feasible the verification of some masked programs on which enumeration is simply unfeasible. To illustrate both these improvements and the differences between our algorithms, we apply the three methods to the Sbox of [CPRR14] (Algorithm 4) protected at various orders. Table 3.1 shows the results, where column *# tuples* contains the total number of tuples of program points to be considered, column *# sets* contains the number of sets used by the splitting algorithms and the *time* column shows the verification times when run on a headless VM with a dual core[4] 64-bit processor clocked at 2GHz. As can be seen, the worklist-based method is generally the most efficient one. In the following, and in particular in Section 3.6, we use the check function from Algorithm 6.

Discussion

Note that in both Algorithms 5 and 6, the worst execution time occurs when the call to extend does not in fact increase the size of the observation set under study. In the unlikely event where

[4]Only one core is used in the computation.

Table 3.1: Comparison of Algorithms 5 and 6 with naive enumeration and with each other.

Method	# tuples	Security	Complexity	
			# sets	time
First-Order Masking				
naive	63	✓	63	0.001s
pair			17	0.001s
list			17	0.001s
Second-Order Masking				
naive	12,561	✓	12,561	0.180s
pair			851	0.046s
list			619	0.029s
Third-Order Masking				
naive	4,499,950	✓	4,499,950	140.642s
pair			68,492	9.923s
list			33,075	3.894s
Fourth-Order Masking				
naive	2,277,036,685	✓	2,277,036,685	unpractical
pair			8,852,144	2959.770s
list			3,343,587	879.235s

this occurs in *all* recursive calls, both algorithms degrade into an exhaustive enumeration of all tuples, which is no worse than the naive implementation. However, this observation makes it clear that it is important for the extend function to extend observation sets as much as possible. It could be interesting, and would definitely be valuable, to find a good balance between the complexity and precision of the extend function.

3.5 Initial Transformations on Programs: An Example

To illustrate our algorithms, we consider the simple masked multiplication algorithm defined in [RP10] and relying on Algorithm 7, which is secure against 2-threshold probing adversaries. In practice, the code we consider is in 3-address form, with a single operation per line (operator application or table lookup). For brevity, we use parentheses instead, unless relevant to the discussion. In the rest of this chapter, we write Line $(n).i$ to denote the i^{th} expression computed on line n, using the convention that products are computed immediately before their use. For example, Line (5).1 is the expression $a_0 \odot b_1$, Line (5).2 is $r_{0,1} \oplus a_0 \odot b_1$ and Line (5).3 is $a_1 \odot b_0$.

When given a program whose inputs have been annotated as secret or public, we transform it as described above to add some simple initialization code that preshares secrets in a way that is not observable by the adversary, and lets the adversary observe the initial sharing of public inputs. This allows us to model, as part of the program, the assumption that shares of the secret are initially uniformly distributed and that their sum is the secret. The initialization code, as well as the transformed version of Algorithm 7 where argument a is marked as secret and b is marked as public, are shown in Algorithm 8. We use the square brackets on Line (4) of function PreShare to mean that the intermediate results obtained during the computation of the bracketed expression are not observable by the adversary: this is equivalent to the usual assumption that secret inputs and state are shared before the adversary starts performing measurements.

Once the program is in this form, it can be transformed to obtain: (i) the set of its

Algorithm 7 Secure Multiplication Algorithm ($t = 2$) from [RP10]

Input: a_0, a_1, a_2 (resp. b_0, b_1, b_2) such that $a_0 \oplus a_1 \oplus a_2 = a$ (resp. $b_0 \oplus b_1 \oplus b_2 = b$)
Output: c_0, c_1, c_2 such that $c_0 \oplus c_1 \oplus c_2 = a \odot b$

1: **function** SecMult($[\![a_0, a_1, a_2]\!], [\![b_0, b_1, b_2]\!]$)
2: $r_{0,1} \xleftarrow{\$} \mathsf{GF}(256)$
3: $r_{0,2} \xleftarrow{\$} \mathsf{GF}(256)$
4: $r_{1,2} \xleftarrow{\$} \mathsf{GF}(256)$
5: $r_{1,0} \leftarrow (r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0$
6: $r_{2,0} \leftarrow (r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0$
7: $r_{2,1} \leftarrow (r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1$
8: $c_0 \leftarrow (a_0 \odot b_0 \oplus r_{0,1}) \oplus r_{0,2}$
9: $c_1 \leftarrow (a_1 \odot b_1 \oplus r_{1,0}) \oplus r_{1,2}$
10: $c_2 \leftarrow (a_2 \odot b_2 \oplus r_{2,0}) \oplus r_{2,1}$
11: **return** $[\![c_0, c_1, c_2]\!]$

Algorithm 8 Presharing, Sharing and Preprocessed multiplication ($t = 2$, a is secret, b is public)

1: **function** PreShare(a)
2: $a_0 \xleftarrow{\$} \mathsf{GF}(256)$
3: $a_1 \xleftarrow{\$} \mathsf{GF}(256)$
4: $a_2 \leftarrow [a \oplus a_0 \oplus a_1]$
5: **return** $[\![a_0, a_1, a_2]\!]$

1: **function** Share(a)
2: $a_0 \xleftarrow{\$} \mathsf{GF}(256)$
3: $a_1 \xleftarrow{\$} \mathsf{GF}(256)$
4: $a_2 \leftarrow (a \oplus a_0) \oplus a_1$
5: **return** $[\![a_0, a_1, a_2]\!]$

1: **function** $\overline{\text{SecMult}}(a, b)$
2: $a_0 \xleftarrow{\$} \mathsf{GF}(256)$
3: $a_1 \xleftarrow{\$} \mathsf{GF}(256)$
4: $a_2 \leftarrow [a \oplus a_0 \oplus a_1]$
5: $b_0 \xleftarrow{\$} \mathsf{GF}(256)$
6: $b_1 \xleftarrow{\$} \mathsf{GF}(256)$
7: $b_2 \leftarrow (b \oplus b_0) \oplus b_1$
8: $r_{0,1} \xleftarrow{\$} \mathsf{GF}(256)$
9: $r_{0,2} \xleftarrow{\$} \mathsf{GF}(256)$
10: $r_{1,2} \xleftarrow{\$} \mathsf{GF}(256)$
11: $r_{1,0} \leftarrow (r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0$
12: $r_{2,0} \leftarrow (r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0$
13: $r_{2,1} \leftarrow (r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1$
14: $c_0 \leftarrow (a_0 \odot b_0 \oplus r_{0,1}) \oplus r_{0,2}$
15: $c_1 \leftarrow (a_1 \odot b_1 \oplus r_{1,0}) \oplus r_{1,2}$
16: $c_2 \leftarrow (a_2 \odot b_2 \oplus r_{2,0}) \oplus r_{2,1}$
17: **return** $[c_0 \oplus c_1 \oplus c_2]$

Figure 3.1: Possible wire observations for $\overline{\text{SecMult}}$. (Note that, after Lines 4 and 7, we keep a_2 and b_2 in expressions due to margin constraints.)

Line	Observed Expression	Line	Observed Expression
(2)	a_0	(12).2	$r_{0,2} \oplus a_0 \odot b_2$
(3)	a_1	(12).3	$a_2 \odot b_0$
(4)	$a_2 := (a \oplus a_0) \oplus a_1$	(12)	$(r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0$
(5)	b_0	(13).1	$a_1 \odot b_2$
(6)	b_1	(13).2	$r_{1,2} \oplus a_1 \odot b_2$
(7).1	$b \oplus b_0$	(13).3	$a_2 \odot b_1$
(7)	$b_2 := (b \oplus b_0) \oplus b_1$	(13)	$(r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1$
(8)	$r_{0,1}$	(14).1	$a_0 \odot b_0$
(9)	$r_{0,2}$	(14).2	$a_0 \odot b_0 \oplus r_{0,1}$
(10)	$r_{1,2}$	(14)	$(a_0 \odot b_0 \oplus r_{0,1}) \oplus r_{0,2}$
(11).1	$a_0 \odot b_1$	(15).1	$a_1 \odot b_1$
(11).2	$r_{0,1} \oplus a_0 \odot b_1$	(15).2	$a_1 \odot b_1 \oplus ((r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0)$
(11).3	$a_1 \odot b_0$	(15)	$(a_1 \odot b_1 \oplus ((r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0)) \oplus r_{1,2}$
(11)	$(r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0$	(16).1	$a_2 \odot b_2$
(12).1	$a_0 \odot b_2$	(16).2	$a_2 \odot b_2 \oplus ((r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0)$
		(16)	$(16).2 \oplus ((r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1)$

random variables;[5] (*ii*) the set of expressions representing all of the possible adversary observations. This final processing step on $\overline{\text{SecMult}}$ yields the set of random variables $\mathcal{R} = \{a_0, a_1, b_0, b_1, r_{0,1}, r_{0,2}, r_{1,2}\}$, and the set of expressions shown in Figure 3.1 (labelled with their extended line number). Recall that these sets were obtained with a marked as secret and b marked as public.

Figure 3.2 presents the observable transitions for Algorithm 7. It gives the old value and the new value of the register modified by each program point. This is done using a simple register allocation of Algorithm 7 (where we use the word "register" loosely, to denote program variables, plus perhaps some additional temporary registers if required) that uses a single temporary register that is never cleared, and stores intermediate computations in the variable where their end result is stored. For clarity, the register in which the intermediate result is stored is also listed in the Figure.

3.6 Experiments

In this section, we aim to show on concrete examples the efficiency of the methods we considered so far. This evaluation is performed using a prototype implementation of our algorithms that uses the EasyCrypt [BDG+14] tool's internal representations of programs and expressions, and relying on some of its low-level tactics for substitution and conversion. As such, the prototype is not designed for performance, but rather for trust, and the time measurements given below could certainly be improved. However, the numbers of sets each algorithm considers are fixed by our choice of algorithm, and by the particular choose algorithms we decided to use. We detail and discuss this particular implementation decision at the end of this section.

Our choice of examples mainly focuses on higher-order masking schemes since they are much more promising than the schemes dedicated to small orders. Aside from the masking order itself,

[5] In practice, since we consider programs in SSA form, it is not possible to assign a non-random value to a variable that was initialized with a random.

Figure 3.2: Possible transition observations for $\overline{\text{SecMult}}$. $\perp$ denotes an uninitialized register, whose content may already be known to (and perhaps chosen by) the adversary.

Line	Register	Old Contents	New Contents
(2)	a_0	$\perp$	a_0
(3)	a_1	$\perp$	a_1
(4)	a_2	$\perp$	$a \oplus a_0 \oplus a_1$
(5)	b_0	$\perp$	b_0
(6)	b_1	$\perp$	b_1
(7).1	b_2	$\perp$	$b \oplus b_0$
(7)	b_2	$b \oplus b_0$	$b \oplus b_0 \oplus a_1$
(8)	$r_{0,1}$	$\perp$	$r_{0,1}$
(9)	$r_{0,2}$	$\perp$	$r_{0,2}$
(10)	$r_{1,2}$	$\perp$	$r_{1,2}$
(11).1	$r_{1,0}$	$\perp$	$a_0 \odot b_1$
(11).2	$r_{1,0}$	$a_0 \odot b_1$	$r_{0,1} \oplus a_0 \odot b_1$
(11).3	t	$\perp$	$a_1 \odot b_0$
(11)	$r_{1,0}$	$r_{0,1} \oplus a_0 \odot b_1$	$r_{0,1} \oplus a_0 \odot b_1 \oplus a_1 \odot b_0$
(12).1	$r_{2,0}$	$\perp$	$a_0 \odot b_2$
(12).2	$r_{2,0}$	$a_0 \odot b_2$	$r_{0,2} \oplus a_0 \odot b_2$
(12).3	t	$a_1 \odot b_0$	$a_2 \odot b_0$
(12)	$r_{2,0}$	$r_{0,2} \oplus a_0 \odot b_2$	$r_{0,2} \oplus a_0 \odot b_2 \oplus a_2 \odot b_0$
(13).1	$r_{2,1}$	$\perp$	$a_1 \odot b_2$
(13).2	$r_{2,1}$	$a_1 \odot b_2$	$r_{1,2} \oplus a_1 \odot b_2$
(13).3	t	$a_2 \odot b_0$	$a_2 \odot b_1$
(13)	$r_{2,1}$	$r_{1,2} \oplus a_1 \odot b_2$	$r_{1,2} \oplus a_1 \odot b_2 \oplus a_2 \odot b_1$
(14).1	c_0	$\perp$	$a_0 \odot b_0$
(14).2	c_0	$a_0 \odot b_0$	$a_0 \odot b_0 \oplus r_{0,1}$
(14)	c_0	$a_0 \odot b_0 \oplus r_{0,1}$	$a_0 \odot b_0 \oplus r_{0,1} \oplus r_{0,2}$
(15).1	c_1	$\perp$	$a_1 \odot b_1$
(15).2	c_1	$a_1 \odot b_1$	$a_1 \odot b_1 \oplus r_{0,1} \oplus a_0 \odot b_1 \oplus a_1 \odot b_0$
(15)	c_1	$a_1 \odot b_1 \oplus r_{0,1} \oplus a_0 \odot b_1 \oplus a_1 \odot b_0$	(15).2 $\oplus r_{1,2}$
(16).1	c_2	$\perp$	$a_2 \odot b_2$
(16).2	c_2	$a_2 \odot b_2$	$a_2 \odot b_2 \oplus r_{0,2} \oplus a_0 \odot b_2 \oplus a_2 \odot b_0$
(16)	c_2	$a_2 \odot b_2 \oplus r_{0,2} \oplus a_0 \odot b_2 \oplus a_2 \odot b_0$	(16).2 $\oplus r_{1,2} \oplus a_1 \odot b_2 \oplus a_2 \odot b_1$

the most salient limiting factor for performance is the size of the program considered, which is also (more or less) the number of observations that need to be considered. Still, we analyze programs of sizes ranging from simple multiplication algorithms to either round-reduced or full AES, depending on the masking order.

We discuss our practical results depending on the leakage model considered: we first discuss our prototype's performance in the value-based leakage model, then focus on results obtained in the transition-based leakage model.

3.6.1 Value-based Model

Table 3.2 lists the performance of our prototype on multiple examples, presenting the total number of sets of observations to be considered (giving an indication of each problem's relative difficulty), as well as the number of sets used to cover all tuples of observations by our prototype. We also list the verification time, although these could certainly be improved independently of the algorithms themselves. Each of our tests is identified by a reference and a function, with additional information where relevant. The MAC-Keccak example is a simple implementation of Keccak-f on 64-bit words, masked using a variant of Ishai, Sahai and Wagner's transformation [ISW03, RP10] (noting that their $\mathsf{SecMult}$ algorithm can be used to securely compute any associative and commutative binary operation that distributes over field addition, including bitwise ANDs). The three rows without checkmarks correspond to examples on which the tool fails to prove t-non-interference. We now analyze them in more detail.

On Schramm and Paar's table-based third and fourth-order implementations of the AES Sbox, our tool finds in both cases 98,176 triples that it cannot prove independent from the secrets. In the third-order implementation, all these errors are found in 695.236s while it takes 22,119s (around thirty times more) to cover all triples in the fourth-order implementation. However, the first error is found in only 0.168s in the first case and in 0.221s in the second one. The recovered triples in fact correspond to four families of observations, which we now describe. Denoting by $X = \bigoplus_{0 \leqslant i \leqslant 3} x_i$ the Sbox input and by $Y = \bigoplus_{0 \leqslant i \leqslant 3} y_i$ its output, we can write the four sets of flawed triples as follows:

1. $\forall i, j \in \mathsf{GF}(2^8), i \neq j, \quad (x_0, \mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus i) \oplus (y_1 \oplus y_2 \oplus y_3),$
   $\mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus j) \oplus (y_1 \oplus y_2 \oplus y_3))$
2. $\forall i, j \in \mathsf{GF}(2^8), \ i \neq j, \quad (y_0, \mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus i) \oplus (y_1 \oplus y_2 \oplus y_3),$
   $\mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus j) \oplus (y_1 \oplus y_2 \oplus y_3))$
3. $\forall i, j \in \mathsf{GF}(2^8), \ i \neq j, \quad (x_0, \mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus i) \oplus (y_1 \oplus y_2),$
   $\mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus j) \oplus (y_1 \oplus y_2))$
4. $\forall i \in \mathsf{GF}(2^8), \quad (x_0, y_0, \mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus i) \oplus (y_1 \oplus y_2 \oplus y_3)).$

We recall that y_0 is read as $y_0 = \mathsf{Sbox}(x_0)$, and prove that all four families of observations in fact correspond to attacks. The first family corresponds to the attack detailed by Coron, Prouff and Rivain [CPR07]). By summing the second and third variables, the attacker obtains $\mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus i) \oplus \mathsf{Sbox}(x_1 \oplus x_2 \oplus x_3 \oplus j)$. The additional knowledge of x_0 clearly breaks the independence from X. To recover secrets from a set of observations of the second type, the attacker can sum the second and third variables to obtain $x_1 \oplus x_2 \oplus x_3$, from which he can learn $y_1 \oplus y_2 \oplus y_3$ (by combining it with the second variable) and then Y (by combining it with the first one). The third family is a variant of the first: the Sbox masks can be removed in both cases. Finally, when observing three variables in the fourth family of observations, the knowledge of both x_0 and y_0 unmasks the third observed variable, making it dependent on X.

Table 3.2: Verification of state-of-the-art higher-order masking schemes with # tuples the number t-uples of the algorithm at order t, # sets the number of sets built by our prototype and time the verification time in seconds

Reference	Target	# tuples	Result	Complexity	
				# sets	time (s)
First-Order Masking					
[RP10]	multiplication	13	✓	7	ε
[CPRR14]	Sbox (4)	63	✓	17	ε
[CPRR14]	full AES (4)	17,206	✓	3,342	128
[DPAK12]	full Keccak-f	13,466	✓	5,421	405
Second-Order Masking					
[SP06]	Sbox	1,188,111	✓	4,104	1.649
[RP10]	multiplication	435	✓	92	0.001
[RP10]	Sbox	7,140	2^{nd}-order flaws (2)	866	0.045
[CPRR14]	key schedule	23,041,866	✓	771,263	340,745
[CPRR14]	AES 2 rounds (4)	25,429,146	✓	511,865	1,295
[CPRR14]	AES 4 rounds (4)	109,571,806	✓	2,317,593	40,169
Third-Order Masking					
[RP10]	multiplication	24,804	✓	1,410	0.033
[CPRR14]	Sbox(4)	4,499,950	✓	33,075	3.894
[CPRR14]	Sbox(5)	4,499,950	✓	39,613	5.036
[SP06]	Sbox	2,057,067,320	3^{rd}-order flaws $(98, 176)$	2,013,070	695
Fourth-Order Masking					
[RP10]	multiplication	2,024,785	✓	33,322	1.138
[CPRR14]	Sbox (4)	$2,277,036,685$	✓	3,343,587	879
[SP06]	Sbox	4,874,429,560	3^{rd}-order flaws $(98, 176)$	35,895,437	22,119
Fifth-Order Masking					
[RP10]	multiplication	216,071,394	✓	856,147	45

Table 3.3: Fixing RP-CHES10 [RP10] at the second order

Reference	Sbox	# tuples	Result	Complexity	
				# sets	time
Second-Order Masking					
[RP10]	initially proposed	7,140	2^{nd}-order flaws (2)	840	0.070s
[RP10]	different refreshMasks	7,875	✓	949	0.164s
[RP10]	more refreshMasks	8,646	✓	902	0.180s
[CPRR14]	$x \cdot g(x)$ [Alg. 4]	12,561	✓	619	0.073s
[CPRR14]	tables [Alg. 5]	12,561	✓	955	0.196s

Our tool also finds two suspicious adversary observations on the Sbox algorithm proposed by Rivain and Prouff [RP10], that in fact correspond to the two flaws revealed in [CPRR14]. However, by the soundness of our algorithm, and since our implementation only reports these two flaws, we now know that these are the only two observations that reveal any information on the secrets. We consider several corrected versions of this Sbox algorithm, listed in Table 3.3. Some of these fixes focused on using a more secure mask refreshing function (borrowed from [DDF14]) or refreshing all modified variables that are reused later on (as suggested by [PR13]). Others make use of specialized versions of the multiplication algorithm [CPRR14] that allow the masked program to retain its performance whilst gaining in security.

Although it is important to note that the algorithms appear to be "precise enough" in practice, Table 3.2 also reveals that program size is not in fact the only source of complexity. Indeed, proving the full key schedule at order 2 only involves around 23 million pairs of observations, compared to the 109 million that need to be considered to prove the security of 4 rounds of AES at the same order; yet the latter takes less than 12 hours to complete compared to 4 days for the full ten rounds of key schedule. We suspect that this is due to the shapes of the two programs' dependency graphs, with each variable in the key schedule depending on a large proportion of the program's input variables, whereas the dependencies in 4 rounds of AES (including iterative key schedule) are sparser. Although properties of composition would allow us to consider large programs masked at much higher orders, we leave these investigations to further works.

Another important factor in the performance of our algorithm is the instantiation of the various choice functions. We describe them here for the sake of reproducibility. In Algorithm 2, when choosing a triple $(\boldsymbol{e}', e, r)$ to use with rule (Opt), our prototype first chooses r as the first (leftmost-first depth-first) random variable that fulfills the required conditions, then chooses e as the *largest* superterm of r that fulfills the required conditions (this fixes $\boldsymbol{e}'$). When choosing an expression to observe (in Algorithms 5 and 6) or to extend a set of observations with (in Algorithm 4), we choose first the expression that has the highest number of dependencies on random or input variables. These decisions certainly may have a significant effect on our algorithm's performance, and investigating these effects more deeply may help gather some insight on the core problems related to masking. We leave this for future work.

3.6.2 Transition-Based Model

The value-based leakage model may not always be the best fit to capture the behaviour of hardware and software. In particular, when considering software implementations, it is possible that writing a value into a register leaks both its new and old contents. To illustrate the adaptability of our algorithms, we first run some simple tests. We then illustrate another potential application of our tool, whereby masked implementations that make use of $t+1$ masks

Table 3.4: Multiplication in the transition-based leakage model

Reference	Multiplication	# tuples	Security	Complexity	
				# sets	time
[RP10]	initial scheme for order 4	3,570	order 2	161	0.008s
[RP10]	with some instructions reordering	98,770	order 3	3,488	0.179s
[RP10]	using more registers	2,024,785	order 4	17,319	1.235s

per variable can be proved secure in the transitions model at orders much higher than the generic $t/2$, simply by reordering instructions and reallocating registers.

Table 3.4 describes the result of our experiments. Our first (naive) implementation is only secure at the second order in the transition-based leakage model and uses 21 local registers (the number of registers needed for this and other implementations to be secure could also be reduced further by zeroing out registers between independent uses). Our first improved implementation achieves security at order 3 in the transition-based leakage model with only 6 local registers. Trying to provide the best possible security in this model, we also find a third implementation that achieves security at order 4. This last implementation is in fact the original implementation with additional registers. Note however, that in spite of its maximal security order, this last implementation still reuses registers (in fact, most are used at least twice).

The main point of these experiments is to show that the techniques and tools we developed, are helpful in building and verifying implementations in other models. Concretely, our tools give countermeasure designers the chance to easily check the security of their implementation in one or the other leakage model, and identify problematic observations that would prevent the countermeasure from operating properly against higher-order adversaries.

3.7 Conclusion

This work initiated the study of relational verification techniques for checking the security of masked implementations against DPA attacks at order t. Beyond demonstrating the feasibility of this approach for masking orders higher than 2, our tool benefits from having no false positive by construction. That is, no program can be typed as secure while it is not, which is mandatory for such evaluations. Furthermore, while our verifier could return false negatives, no such case happened in all the examples we evaluated. That is, all the potential attack path returned by the verifier happened to be real attack paths. As a consequence, we can be confident in the tightness in the evaluation.

The most immediate direction for further work seems to be the exhibition and the proof of compositional properties in order to achieve the verification of larger masked programs at higher orders. We tackle these issues in the next chapter.

Chapter 4

Construction of Secure Higher-Order Masking

In the previous chapter, we aimed to verify the security of higher-order masked algorithms. However, despite the improved techniques that we introduced, we are still limited in the size and the order of the programs because of the huge number of intermediate variables. Thus, the goal of this chapter's work is to exhibit compositional properties to verify but also build higher-order masking algorithms without any restriction on the program size.

Contents

4.1 Introduction

4.1.1 Motivation

There is a trade-off between the level of protection provided by masking which increases with the masking order t, and the efficiency of masked implementations. On the one hand, the number of execution traces required to mount practical attacks and the complexity to provide concrete evaluations increase exponentially with t at it is indicated in [CJRR99, SVO+10] and recalled in Part I. On the other hand, the complexity of masked implementations increases polynomially with t. Nevertheless, there is a growing interest in building implementations that are secure at high orders.

Ultimately, the validity of countermeasures such as masking is assessed by the level of protection they offer. This assessment is typically performed in two steps; first, by giving a security proof of algorithms in an appropriate model of leakage, and second, through experiments for some particular implementation on the targeted component. In this chapter, we consider the first step and leave the second step for future work. With this in mind, our first task is to choose a suitable security model. A presentation of the most fitted leakage models is given in Part I. But concretely, thanks to the last results of Duc Dziembowsky and Faust [DDF14], the t-threshold probing model can now realistically be used to prove the security of masked implementations of basic building blocks used in cryptographic algorithms, including for instance multipliers. In the following, we will refer to these masked basic building blocks, that securely implement basic operations from the original algorithm, as *gadgets*. Unfortunately, proving the security of masked implementations for complete algorithms, such as AES or Keccak, remains a challenge. Existing works apply compositional principles to derive the security of a masked implementation from the security of its gadgets, but these works have two main shortcomings. Some of them lack of rigorous justification, and as a consequence, the resulting masked algorithms are sometimes insecure; for instance, the mask refreshing operation proposed by Rivain and Prouff [RP10] leads to an insecure Sbox [CPRR14] when composed with other secure gadgets. The others overprotect the algorithms in the t-threshold probing model, which might affect their performances; for instance, the authors of [PR13, DDF14] propose to insert refresh gadgets between each operation or each time a sensitive variable is reused. Doing so, they prove the security of their algorithms when the attacker is authorized to target t intermediate variables on each gadget, but leave open the possibility of achieving t-threshold probing security with a smaller number of refresh gadgets.

4.1.2 Contributions

Our first contribution tackles the composition issues from the security properties of the atomic gadgets. We start with the observation that the common security property of *simulatability*[1] used by Rivain and Prouff [RP10] is not sufficient to guarantee that the composition of masked algorithms remains secure. We thus introduce a new and stronger security property for gadgets that we refer to as *strong simulatability*. It guarantees a form of independence between inputs and outputs which makes it very convenient to achieve composition. Intuitively, simulatability ensures that any set of $d \leq t$ observations made by the adversary can be simulated with at most d inputs. Strong simulatability is a stronger property which ensures that the number of input shares necessary to simulate the adversary observations should be independent from the number of observations made on output wires.

[1] In [RP10], Rivain and Prouff use the term *perfect simulation* to define this notion.

We validate our definition of strong simulatability through two main theoretical contributions. First, we prove that several gadgets from the literature are strongly simulatable: the multiplication of Rivain and Prouff [RP10], the (same) multiplication-based mask refreshing algorithm and the multiplication between linearly dependent inputs proposed by Coron *et al.* in [CPRR14]. The proofs of the first and second gadgets are machine-checked in EasyCrypt [BGHZ11,BDG+14], a computer-aided tool for reasoning about the security of cryptographic constructions and relational properties of probabilistic programs. The distinguishing feature of our machine-checked proofs is that they show security at arbitrary levels, whereas previous machine-checked proofs are limited to low orders: 1 or 2, and in some cases up to 5 (see Chapter 3). Second, we use the strongly simulatable multiplication-based mask refreshing algorithm to define a sound method for securely composing masked algorithms. More specifically, our main composition result shows that sensitive data can be reused as inputs to different gadgets without security flaw using judiciously placed strongly simulatable gadgets. Then, we describe a novel and efficient technique (Theorem 9) to securely compose two gadgets (without requiring that they be strongly simulatable) when the adversary can place t probes inside each of them. The characteristics of our technique make it particularly well-suited to the protection of algorithms with sensitive state in the adaptive model of Ishai, Sahai and Wagner [ISW03]. In particular, we do not require that the masking order be doubled throughout the circuit.

We also make two practical contributions to support the development of secure masking algorithms based on our notions and results. First, we define and implement an automated approach for verifying that an algorithm built by composing provably secure gadgets is itself secure. Second, and using advanced tools from programming languages, we implement an algorithm that takes as input an unprotected program P and an arbitrary order t and automatically outputs a functionally equivalent algorithm that is protected at order t, inserting mask refreshing gadgets where required.

Finally, our last contribution is experimental; using our transformation, we generate secure (for selected orders up to 10) masked algorithms for AES, Keccak, Simon, and Speck, and evaluate their running time.

4.1.3 Related Work

There has been significant work on building secure implementations of multiplications and other core functionalities; however, most of the work is based on the weaker notion of simulatability, and applies informal guidelines to obtain pen-and-paper proofs of security ([ISW03, RP10, GPQ11, CPRR14, Cor14]). Only Faust *et al.* in [FRR+14] consider formally this problem in a restricted version of the noisy leakage model. In contrast, there has been relatively little work on developing automated tools for checking that an implementation is correctly masked (see Chapter 3), or for automatically producing a masked implementation from an unprotected program. This practical line of work was first considered in the context of first-order boolean masking [MOPT12]. Subsequent works extend this approach to accommodate higher-order and arithmetic masking, using type systems and SMT solvers [BRNI13], or model counting and SMT solvers [EWS14, EW14] (see Part I, Chapter 2). The algorithmic complexity of the latter approach severely constrains its applications; in particular, tools based on model counting can only analyze first or second order masked implementations, and can only deal with round-reduced versions of the algorithms; for instance, only analyzing a single round of Keccak.

4.1.4 Outline

We first define our new security property of strong simulatability and some useful lemmas to achieve composition in Section 4.2. Then, we prove in Section 4.3 the strong simulatability of three gadgets which form the cornerstone of secure implementations: a multiplication-based refresh algorithm, the widely used secure multiplication and a multiplication between linearly dependent data. In Section 4.4, we informally describe our method to securely compose gadgets when the adversary can observe t intermediate variables in the whole circuit. We also give a complete and compositional proof of security for the well-known inversion algorithm in the Rijndael field. A formalization of this method to prove the security of large circuits is given in [BBD+15a]. In Section 4.5, we introduce a new method to securely compose two circuits when the adversary may place t probes in each of them. Finally, in Section 4.6, we evaluate the practicality of our approach by generating secure implementations from unprotected versions and measuring verification statistic and the performance of the resulting masked programs.

4.2 Composition

In this section, we review existing concepts from the literature and we display the current issues with composition. We then present an advantageous way of composing affine gadgets and introduce a new and useful notion which forms the basis of sound compositional reasoning: t-strong non-interference.

4.2.1 Gadgets

We consider programs that operate over a single finite structure $(\mathbb{K}, 0, 1, \oplus, \ominus, \odot)$; however, our techniques and tools extend smoothly to more complex scenarios. Widely used examples of such structures include binary fields, boolean rings and modular arithmetic structures. In particular cases, it may be beneficial to consider group structures rather than rings (or fields), but we do not do so here.

We define gadgets as probabilistic functions that return, in addition to their actual output, all intermediate values (including inputs) obtained during the computation. For example, the first-order mask refreshing gadget shown in Algorithm 9, can be interpreted as the following probabilistic function where r is sampled uniformly at random in $\mathbb{K}$.

$$\mathsf{Refresh}_1(a_0, a_1) = ((a_0, a_1, r), (a_0 \oplus r, a_1 \ominus r))$$

where (a_0, a_1, r) is the gadget's leakage (which depends on its implementation), and $(a_0 \oplus r, a_1 \ominus r)$ is the gadget's output.

Algorithm 9 Example of a Gadget

```
1: function Refresh_1(a)
2:     r <-$ K
3:     c_0 <- a_0 ⊕ r
4:     c_1 <- a_1 ⊖ r
5:     return c
```

Definition 10 (Gadgets)**.** *Let $m, n, \ell, o \in \mathbb{N}$. An (m, n, ℓ, o)-gadget is a probabilistic function $G : (\mathbb{K}^m)^n \to \mathbb{K}^\ell \times (\mathbb{K}^m)^o$. Parameter m denotes the gadget's* bundle size *(the number of shares over which values in $\mathbb{K}$ are shared), n is the gadget's* arity *(or number of input bundles), ℓ is*

the gadget's number of leakage wires *(or side-channel wires), and o is the number of* output bundles.

By convention, we will always include *input wires* in a gadget's leakage (and therefore always have $m{\cdot}n \leq \ell$), but will never count *output wires* in a gadget's leakage (to avoid double counting). For example, function $\mathsf{Refresh}_1$ (Algorithm 9) is a $(2, 1, 3, 1)$-gadget. In the following, we omit m when it is equal to the classical masking order t, and write $\widetilde{\mathbb{K}}$ for $\mathbb{K}^m$. We also assume that leakage wires are numbered following program order where relevant.

In practice, adversaries are never given direct access to the raw gadgets, but rather to *projections* of the gadget, that restrict the number of leaks and outputs the adversary can observe each time she queries the oracle. For any $\mathcal{O}^{(out)} \subseteq \mathbb{N}$, we define the projector $\pi_{\mathcal{O}}$ that, given a tuple, projects out the positions indicated by $\mathcal{O}$. For example, $\pi_{\{0,2\}}(x, y, z) = (x, z)$. We generalize as expected to projections on tuples of tuples.

4.2.2 t-Simulatability and t-Non-Interference

Our goal is to prove the security of a gadget in the t-threshold probing model of Ishai, Sahai and Wagner [ISW03]. To this end, we give an adapted definition of the notion of *t-non-interference* for gadgets that is sufficient to prove security in their stateless model. We treat stateful oracles later on, in Section 4.5.

A *set of observations* is represented here by a pair $(\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$ such that $\mathcal{O}^{(int)} \subseteq \mathbb{N}$ is the set of *internal observations* and $\mathcal{O}^{(out)} \subseteq \mathbb{N}^*$ is the set of *output observations*, that is, a set of sets of indexes describing which wires are observed for each output bundle; in the remainder, we often use Ω to denote observation sets. Given an (n, ℓ, o)-gadget G, we say a set of observations is admissible for G if only existing wires are observed, and is t-admissible if it is admissible and the total number of wires observed is bounded by t. We omit G when clear from context, and abuse notation, often writing $|(\mathcal{O}^{(int)}, \mathcal{O}^{(out)})|$ to denote the quantity $|\mathcal{O}^{(int)}| + \sum_{i=0}^{0-1} |\mathcal{O}^{(out)}{}_i|$.

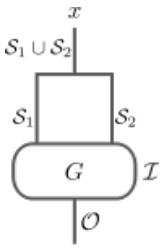
An input projection is a set $\mathcal{S} \subseteq \mathbb{N}^*$. Given an (n, ℓ, o)-gadget G, we say that an input projection $\mathcal{S}$ is *compatible for G* whenever $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_{n-1})$ and $\mathcal{S}_i \subseteq [0..m)$ for $i = 0, \ldots, n-1$. As before, we say that an input projection $\mathcal{S}$ is *t-compatible for G* if it is compatible for G and is such that $|\mathcal{S}_i| \leq t$. Intuitively, an input projection is compatible if it only projects existing input wires, and is t-compatible if it is compatible and projects at most t wires of *each* input bundle. Here too, we abuse notation and write $|\mathcal{S}|$ to denote the quantity $\sum_{i=0}^{n-1} |\mathcal{S}_i|$.

Definition 11 ($(\mathcal{S}, \Omega)$-Simulation, -Non-Interference)**.** *Let G be an (n, ℓ, o)-gadget, $\mathcal{S}$ be a compatible input projection, and Ω be an admissible observation set.*

1. *We say that G is* $(\mathcal{S}, \Omega)$-simulatable *(or $(\mathcal{S}, \Omega)$-SIM) if there exists a simulator G_Ω such that $\pi_\Omega \circ G = G_\Omega \circ \pi_\mathcal{S}$.*
2. *We say that G is* $(\mathcal{S}, \Omega)$-non-interfering *(or $(\mathcal{S}, \Omega)$-NI) if for any $\boldsymbol{s}_0$, $\boldsymbol{s}_1$ in $\widetilde{\mathbb{K}}^n$ such that $\pi_\mathcal{S}(\boldsymbol{s}_0) = \pi_\mathcal{S}(\boldsymbol{s}_1)$, we have $\pi_\Omega \circ G(\boldsymbol{s}_0) = \pi_\Omega \circ G(\boldsymbol{s}_1)$.*

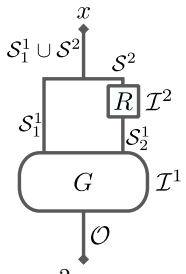
Lemma 2 (Simulation $\Leftrightarrow$ Non-Interference)**.** *For all (n, ℓ, o)-gadget G, compatible $\mathcal{S}$ and admissible Ω, G is $(\mathcal{S}, \Omega)$-SIM iff G is $(\mathcal{S}, \Omega)$-NI.*

Proof of Lemma 2. We consider a (n, ℓ, o)-gadget G, a compatible $\mathcal{S}$ and an admissible Ω. We first prove that $(\mathcal{S}, \Omega)$-SIM implies $(\mathcal{S}, \Omega)$-NI. Assume that G is $(\mathcal{S}, \Omega)$-SIM. For any $\boldsymbol{s}_0$, $\boldsymbol{s}_1$ in $\widetilde{\mathbb{K}}^n$ such that $\pi_\mathcal{S}(\boldsymbol{s}_0) = \pi_\mathcal{S}(\boldsymbol{s}_1)$, we have $\pi_\Omega \circ G(\boldsymbol{s}_0) \overset{Def.\ 11}{=} G_\Omega \circ \pi_\mathcal{S}(\boldsymbol{s}_0) = G_\Omega \circ \pi_\mathcal{S}(\boldsymbol{s}_1) \overset{Def.\ 11}{=} \pi_\Omega \circ G(\boldsymbol{s}_1)$. Thus, G is $(\mathcal{S}, \Omega)$-NI. Assume now that G is $(\mathcal{S}, \Omega)$-NI. From Definition 11, we have $\pi_\Omega \circ G(\boldsymbol{s}_0) = \pi_\Omega \circ G(\boldsymbol{s}_1)$ for any $\boldsymbol{s}_0$, $\boldsymbol{s}_1$ in $\widetilde{\mathbb{K}}^n$ such that $\pi_\mathcal{S}(\boldsymbol{s}_0) = \pi_\mathcal{S}(\boldsymbol{s}_1)$. We construct a simulator



with $|\mathcal{O}^{(int)}| + |\mathcal{O}^{(out)}| \leq t$.

Figure 4.1: Diagram 1



with $|\mathcal{O}^{(int)^2}| + |\mathcal{O}^{(int)^1}| + |\mathcal{O}^{(out)}| \leq t$.

Figure 4.2: Diagram 2

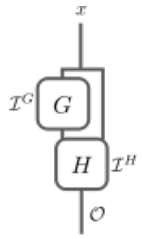


Figure 4.3: Affine-NI composition.

G_Ω by returning, given a projected input $\boldsymbol{\sigma}$, the result of $\pi_\Omega \circ G(\boldsymbol{\sigma}||0)$ for wires in Ω and random values on all other wires (where $\boldsymbol{\sigma}||0$ injects $\boldsymbol{\sigma}$ into $\widetilde{\mathbb{K}}^n$ by assigning 0 to all missing inputs). Since $\pi_{\mathcal{S}}(\pi_{\mathcal{S}}(\boldsymbol{s})||0) = \pi_S(\boldsymbol{s})$ for all full input $\boldsymbol{s}$, we have $G_\Omega \circ \pi_{\mathcal{S}}(\boldsymbol{s}) = \pi_\Omega \circ G(\pi_{\mathcal{S}}(\boldsymbol{s})||0) \overset{Def.11}{=} \pi_\Omega \circ G(\boldsymbol{s})$ for all full input $\boldsymbol{s}$. Thus, G is $(\mathcal{S}, \Omega)$-SIM. □

We now recall the notion of t-NI , defined in Chapter 3, which characterizes security in the t-threshold probing model.

Definition 12 (t-Non-Interference)**.** *An (n, ℓ, o)-gadget G is* t-non-interfering *iff, for every t-admissible Ω, there exists a t-compatible $\mathcal{S}$ such that G is $(\mathcal{S}, \Omega)$-NI. (Equivalently, G is $(\mathcal{S}, \Omega)$-SIM.)*

Inspecting proofs of security from the literature reveals that gadgets sometimes satisfy a slightly stronger property, which we find convenient to introduce as it is more clearly suited to compositional reasoning, and is critically used in the definition of robust composition (Section 4.5).

Definition 13 (t-Tight Non-Interference)**.** *We say that an (n, ℓ, o)-gadget G is* t-tightly non-interfering *(t-TNI) whenever for any t-admissible observation set Ω on G, there exists a $|\Omega|$-compatible $\mathcal{S}$ such that G is $(\mathcal{S}, \Omega)$-NI.*

4.2.3 Issues with Composition

We justify the need for intuitive and correct composition results with simple examples, that also serve to illustrate the need for stronger simulation properties and the basic principles of our verification techniques. We consider the circuits shown in Figures 4.1 and 4.2, that compute $G(x, x)$ for some shared variable x and some $(2, \ell, 1)$-gadget G. R is some $(1, \ell', 1)$-gadget implementing the identity function.

Let us first consider the circuit shown in Diagram 1 (Figure 4.1), assuming that G is t-NI (for example, G could be Rivain and Prouff's secure multiplication gadget [RP10]), and that the adversary makes a t-admissible set of observations $\Omega = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$. From the fact that G is t-NI, we know that Ω can be perfectly simulated using shares $\mathcal{S}_1$ of its first input and shares $\mathcal{S}_2$ of its second input. This means that the simulator for the entire circuit must be given shares $\mathcal{S}_1 \cup \mathcal{S}_2$ of x. The problem is of course that the number of shares of x needed to simulate the full circuit cannot be proved, in general, to be less than $t + 1$, since we only know that $|\mathcal{S}_1 \cup \mathcal{S}_2| < 2t + 1$. A similar argument applies when G is t-TNI. However, the circuit from Figure 4.1 is secure under some specific conditions on G. One such condition, which we call the affine condition, states that the gadget G can be simulated in such a way that $\mathcal{S}_1 = \mathcal{S}_2$. In this

case, the circuit is therefore t-NI. Assume now that G can be simulated using a number of shares of its inputs that is bounded by the number of observations made *in* G by the adversary, rather than by the number of observations made in the entire circuit (that is, t). In other words, we now know that $|\mathcal{S}_1|, |\mathcal{S}_2| \leq |\mathcal{O}^{(int)}| + |\mathcal{O}^{(out)}|$. This is obviously still insufficient to guarantee that the whole circuit can be simulated without knowledge of x, since the simulator could need up to $2(|\mathcal{O}^{(int)}| + |\mathcal{O}^{(out)}|)$ shares of x.

However, if we consider the circuit shown in Diagram 2 (Figure 4.2), we can prove that the circuit is secure under some conditions on R. One such condition, which we call the strong simulatability condition, states that the input of gadget R can be simulated in such a way that $|\mathcal{S}^2| \leq |{\mathcal{O}^{(int)}}^2|$ whenever $|{\mathcal{O}^{(int)}}^2 \cup \mathcal{S}_2^1| \leq t$. Under this assumption, we can prove that the whole circuit is t-NI. Indeed, in such a setting, the entire circuit can be simulated using at most $|\mathcal{S}_1^1 \cup \mathcal{S}^2|$ shares of x. Since $|\mathcal{S}_1^1| \leq |{\mathcal{O}^{(int)}}^1| + |\mathcal{O}^{(out)}|$, $|\mathcal{S}^2| \leq |{\mathcal{O}^{(int)}}^2|$ and $|{\mathcal{O}^{(int)}}^2| + |{\mathcal{O}^{(int)}}^1| + |\mathcal{O}^{(out)}| \leq t$, then $|\mathcal{S}_1^1 \cup \mathcal{S}^2| \leq t$.

The next two paragraphs formalize these conditions. For clarity, we only consider gadgets that output a single wire bundle and omit the value of o in descriptions. Our reasoning to arbitrary gadgets is generalized in [BBD+15a].

4.2.4 Affine Non-Interference

We define the class of affine gadgets and show that they satisfy useful security properties which can be used to build efficient compositions. Informally, a gadget is affine if it performs sharewise computations. By extension, the class of affine functions is the class of functions that can be computed using affine gadgets. For example, using boolean masking in $\mathbb{K} = \mathsf{GF}(2^n)$, affine functions include linear operators (field addition, shifts and rotations), bitwise negation and scalar multiplication.

Definition 14 (Affine Gadgets)**.** *A (n, ℓ)-gadget G is said to be* affine *whenever there exists a family $(G_i)_{0 \leq i < m}$ of probabilistic functions $G_i \in \mathbb{K}^n \to \mathbb{K}^{\ell_i} \times \mathbb{K}$ such that $\sum_{i=0}^{m-1} \ell_i = \ell$, and $G = \prod_{i=0}^{m-1} G_i$ (where the product on functions reorders outputs so that side-channels appear first and in the right order).*

Intuitively, an affine gadget G can be seen as the product (or parallel composition) of m gadgets $(G_i)_{0 \leq i < m}$ such that G_i takes as input the i^{th} share of each of G's n inputs and outputs the i^{th} share of G's output. It is important to note that the leakage computation should also be distributed. Note that the composition of affine gadgets is affine.

Affine gadgets fulfill a more precise property that partially specifies the set of shares of each input required to simulate a given set of observations. We define this more precise property as *affine non-interference*.

Definition 15 (Affine-Non-Interference)**.** *An (n, ℓ)-gadget G is affine-NI iff for every admissible set of observations $\Omega = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$, there exists a $|\mathcal{O}^{(int)}|$-compatible $\widehat{\mathcal{S}}$ such that G is $(\widehat{\mathcal{S}} \cup \mathcal{O}^{(out)}, \Omega)$-NI.*

We now show that every affine gadget is affine-NI, and illustrate how this precise property can be used to prove t-NI in contexts where other gadget-level properties may be insufficient.

Theorem 7. *Every affine (n, ℓ)-gadget G is affine-NI.*

Proof. We prove that G is $(\widehat{\mathcal{S}} \cup \mathcal{O}^{(out)}, \Omega)$-SIM instead. The simulator G_Ω can simply be constructed as the product of all the G_i components of G in which adversary observations occur. This requires the $\mathcal{O}^{(out)}$ shares, plus at most $|\mathcal{O}^{(int)}|$ additional ones due to observations on

side-channels. Intuitively, internal observations are simulated using shares indexed by $\widehat{\mathcal{S}}$, and output observations are simulated using shares indexed by $\mathcal{O}^{(out)}$. □

We now consider a simple example which allows us to see how the details of affine-NI allow fine-grained compositional security proofs.

Lemma 3 (Composition of affine-NI gadgets)**.** *Assuming both G and H are affine-NI, the circuit shown in Figure 4.3, is affine-NI.*

Proof. Let $\Omega = ((\mathcal{O}^{(int)^G}, \mathcal{O}^{(int)^H}), \mathcal{O}^{(out)})$ be a t-admissible observation set for the circuit shown in Figure 4.3. By affine-NI of H, we know that there exists $\widehat{\mathcal{S}}^H$ such that $|\widehat{\mathcal{S}}^H| \leq |\mathcal{O}^{(int)^H}|$ and H is $((\widehat{\mathcal{S}}^H \cup \mathcal{O}^{(out)}, \widehat{\mathcal{S}}^H \cup \mathcal{O}^{(out)}), (\mathcal{O}^{(int)^H}, \mathcal{O}^{(out)}))$-NI. It is therefore sufficient for us to simulate shares $\widehat{\mathcal{S}}^H \cup \mathcal{O}^{(out)}$ of each of H's inputs. One of them is given by G's outputs. By affine-NI of G, we know that there exists $\widehat{\mathcal{S}}^G$ such that $|\widehat{\mathcal{S}}^G| \leq |\mathcal{O}^{(int)^G}|$ and G is $(\widehat{\mathcal{S}}^G \cup (\widehat{\mathcal{S}}^G \cup \mathcal{O}^{(out)}), (\mathcal{O}^{(int)^G}, \widehat{\mathcal{S}}^G \cup \mathcal{O}^{(out)}))$-NI. It is therefore sufficient for us to simulate shares $\widehat{\mathcal{S}}^G \cup \widehat{\mathcal{S}}^H \cup \mathcal{O}^{(out)}$ of G's input. Note that G's input is also H's second input. We therefore need shares $(\widehat{\mathcal{S}}^G \cup (\widehat{\mathcal{S}}^H \cup \mathcal{O}^{(out)})) \cup (\widehat{\mathcal{S}}^H \cup \mathcal{O}^{(out)})$ to simulate the whole circuit. Simplifying, we obtain that we need $\widehat{\mathcal{S}}^G \cup \widehat{\mathcal{S}}^H \cup \mathcal{O}^{(out)}$ shares of the circuit's inputs to simulate the whole circuit. By the constraint on the sizes of $\widehat{\mathcal{S}}^G$ and $\widehat{\mathcal{S}}^H$ and the t-admissibility constraint on Ω, we deduce that the circuit can be simulated using at most t shares of its input and conclude. □

Note that the precise expression for the set of input shares used to simulate leakage given by the notion of affine-NI is key in finishing this proof. In particular, this fine-grained example of composition would be impossible without knowing that the sets of shares of each of H's inputs required to simulate H are equal, and without knowing, in addition, that the set of shares of G's inputs required to simulate G is an extension of the set of shares of its outputs we need to simulate.

4.2.5 t-Strong Non-Interference

We now introduce our main notion for which we will derive sound and secure composition principles: *t-strong non-interference*.

Definition 16 (t-Strong Non-Interference)**.** *A (n, ℓ)-gadget G is said to be* t-strongly non-interfering *(or t-SNI) whenever, for every t-admissible $\Omega = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$, there exists a $|\mathcal{O}^{(int)}|$-compatible $\mathcal{S}$ such that G is $(\mathcal{S}, \Omega)$-NI.*

Essentially, a t-SNI gadget can be simulated using a number of each of its input shares that is only bounded by the number of observations made by the adversary on inner leakage wires, and is independent from the number of observations made on output wires, as long as the total number of observations does not exceed t. This independence with the output observations is critical in securely composing gadgets with related inputs. More specifically, the following lemma illustrates how t-SNI supports compositional reasoning.

Lemma 4 (Composition of t-SNI)**.** *If G and R are t-SNI, the circuit shown in Figure 4.2 is t-SNI.*

Proof. Let $\Omega = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$ be a t-admissible set of observations on the circuit, where $\mathcal{O}^{(int)} = (\mathcal{O}^{(int)^1}, \mathcal{O}^{(int)^2})$ is partitioned depending on the sub-gadget in which observations occur. The set $\Omega^1 = (\mathcal{O}^{(int)^1}, \mathcal{O}^{(out)})$ of observations made on G is t-admissible, and we therefore know, by t-SNI, that there exists an $|\mathcal{O}^{(int)^1}|$-compatible input projection $\mathcal{S}^1 = (\mathcal{S}_1^1, \mathcal{S}_2^1)$ for G such

that G is $(\mathcal{S}, \Omega^1)$-NI. Simulating G and considering that simulator part of the adversary, the set of adversary observations made on R is now $\Omega^2 = ({\mathcal{O}^{(int)}}^2, \mathcal{S}_2^1)$. Since $|\mathcal{S}_2^1| \leq |{\mathcal{O}^{(int)}}^1|$, we know that Ω^2 is t-admissible and can make use of the fact that R is t-SNI. This yields the existence of a $|{\mathcal{O}^{(int)}}^2|$-compatible input projection $\mathcal{S}^2$ for R such that R is $(\mathcal{S}^2, \Omega^2)$-NI. We now have a simulator for the whole circuit that makes use of the set of shares $\mathcal{S}_1^1 \cup \mathcal{S}^2$ of x. Since $|\mathcal{S}_1^1| \leq |{\mathcal{O}^{(int)}}^1|$, $|\mathcal{S}^2| \leq |{\mathcal{O}^{(int)}}^2|$, and $|{\mathcal{O}^{(int)}}^1| + |{\mathcal{O}^{(int)}}^2| \leq t$, we can conclude. Note that this proof method also constructs the simulator for the circuit by composing the simulators for the core gadgets after checking that they exist. □

An interesting example, based on Rivain and Prouff's algorithm to compute an inverse in $\mathsf{GF}(2^8)$ is discussed in Section 4.3 (Figure 4.4 and Theorem 8).

A remark on efficiency. An unexpected benefit of strong non-interference is that it leads to significant efficiency gains: specifically, one can safely dispense from refreshing the output of a strongly non-interfering gadget. We exploit this insight to improve the efficiency of algorithms transformed by our compiler.

4.3 Some Useful SNI Gadgets

In this section, we show that the mask refreshing gadget from Rivain and Prouff [RP10] does not satisfy t-SNI, and hence should be used very carefully (or not at all), whereas the refreshing gadget from Duc, Dziembowski and Faust [DDF14] satisfies t-SNI and thus can be used to compositionally define masked algorithms. Then, we show that the multiplication gadget provided by Rivain and Prouff in [RP10] is also t-SNI. Finally, we also show that the gadget proposed by Coron *et al.* [CPRR14] to compute $h : x \mapsto x \otimes g(x)$ for some linear function g and some internal, associative $\otimes$ that distributes over addition in $\mathbb{K}$ is also t-SNI.

The proofs for the mask refreshing and multiplication gadgets have been verified formally in EasyCrypt; for convenience, we also provide pen-and-paper proof sketches for these proofs and that of the combined gadget from [CPRR14].

4.3.1 Mask Refreshing Algorithms

We focus on two mask refreshing algorithms: the RefreshMasks algorithm introduced in [RP10], that consists in adding a uniform sharing of 0, and an algorithm based on multiplying by a trivial sharing of 1 using secure multiplication algorithm SecMult [RP10]. Below, we show that the former is not t-SNI, whilst the latter is t-SNI and can therefore be used to compositionally build secure implementations.

Addition-Based Mask Refreshing Algorithm

Algorithm 10 is the addition-based refreshing algorithm introduced by Rivain and Prouff [RP10]. Since it only samples a number of random masks linear in the masking order, this algorithm is very efficient, and is in fact t-NI (as proved by its authors [RP10]). However, it is not t-SNI. Indeed, for any order $t \geq 2$, observing the intermediate variable $a_0 \oplus u$ and the output $a_1 \oplus u$ (in the first loop iteration) lets the adversary learn the sum $a_0 \oplus a_1$, which cannot be perfectly simulated from less than two of a's shares. Since it is not t-SNI, the RefreshMasks algorithm cannot be used as illustrated in Section 4.2 to compositionally guarantee the security of masked algorithms. This explains, in particular, the flaw it induces [CPRR14] when used in Rivain and

Prouff's secure Sbox algorithm [RP10]. In fact, the counterexample we gave to t-SNI is central to the flaw exhibited by Coron, Prouff, Rivain, and Roche.

Algorithm 10 Addition-Based Mask Refreshing Algorithm

```
1: function RefreshMasks(a)
2:     for i = 1 to t do
3:         u ←$ 𝕂
4:         a_0 ← a_0 ⊕ u
5:         a_i ← a_i ⊕ u
6:     return a
```

Multiplication-Based Mask Refreshing Algorithm

Algorithm 11 presents the mask refreshing algorithm by Duc *et al.* [DDF14], based on applying the secure multiplication of Rivain and Prouff [RP10] to a trivial sharing of 1 as $(1, 0, \ldots, 0)$.

Algorithm 11 Multiplication-Based Mask Refreshing Algorithm

```
1: function RefreshMult(a)
2:     for i = 0 to t do
3:         c_i ← a_i
4:     for i = 0 to t do
5:         for j = i + 1 to t do
6:             r ←$ 𝕂                    /* this random value is referred to as r_{i,j} */
7:             c_i ← c_i ⊕ r             /* the result is referred to as c_{i,j} */
8:             c_j ← c_j ⊖ r             /* the result is referred to as c_{j,i} */
9:     return c
```

Proposition 2. *Algorithm 11 is t-SNI.*

Proof of Proposition 2. We provide a pen-and-paper proof of Proposition 2; the proof matches closely its formalization in EasyCrypt. The definition of correctness is standard and can be stated formally using the notion of unmasking, where for any n and $x \in \mathbb{K}^n$, the *unmasking of* x is defined as the sum $[\![x]\!] = \bigoplus_{i=0}^{n-1} x_i$. Thus, we have to prove that the algorithm implements the identity function: $[\![\mathsf{RefreshMult}(a)]\!] = [\![a]\!]$. This can be seen by expanding its results and simplifying the sums.

To prove t-SNI, we construct a simulator similar to those previously used to prove the t-NI of several masking transformations ([ISW03, RP10, CPRR14]). Let $\Omega = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$ be a t-admissible set of observations, and let $d_1 = |\mathcal{O}^{(int)}|$ and $d_2 = |\mathcal{O}^{(out)}|$. Note that $d_1 + d_2 \leq t$. Our goals are:

1. to find a d_1-compatible set $\mathcal{S}$,
2. to construct a perfect simulator that uses only shares $\mathcal{S}$ of the inputs.

First, we identify which variables are internals and which are outputs. Internals are the a_i, the $r_{i,j}$ (the value of r at iteration i, j), and the $c_{i,j}$ (resp. $c_{j,i}$) which correspond to the value of the variable c_i (resp. c_j) at iteration i, j. Outputs are the final values of c_i (i.e. $c_{i,t}$).

We define $\mathcal{S}$ as follows: for each observation among a_i, $r_{i,j}$ and $c_{i,j}$ (with $j < t$) we add the index i to $\mathcal{S}$. It is clear that $\mathcal{S}$ contains at most d_1 indexes. We now construct the simulator. For clarity, observe that the RefreshMult algorithm can be equivalently represented using the following matrix:

$$\begin{pmatrix} a_0 & 0 & r_{0,1} & r_{0,2} & \cdots & r_{0,t} \\ a_1 & \ominus r_{0,1} & 0 & r_{1,2} & \cdots & r_{1,t} \\ a_2 & \ominus r_{0,2} & \ominus r_{1,2} & 0 & \cdots & r_{2,t} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_d & \ominus r_{0,t} & \ominus r_{1,t} & \ominus r_{2,t} & \cdots & 0 \end{pmatrix}.$$

In this setting, $c_{i,j}$ corresponds to the partial sum of the $j+2$ first elements of line i. For each $i \in \mathcal{S}$ (that is, for each line i that contains at least one observed internal value), a_i is provided to the simulator (by definition of $\mathcal{S}$). Thus, the simulator can sample all $r_{i,j}$ and compute all partial sums $c_{i,j}$ and the ith output normally. At this point, all values on line $i \in \mathcal{S}$ (internal and output) are perfectly simulated.

We still have to simulate the observed output values for rows on which no internal values are observed. Remark that simulating the ith line also necessarily fixed the value of all random variables appearing in the ith column (so that dependencies between variables are preserved). After internal observations are simulated, at most d_1 lines of the matrix are fully filled. Therefore, at least $t - d_1 \geq d_2$ random values are not simulated on lines on which no internal observations are made. For each output observation made on one such line (say i), we can therefore pick a different $r_{i,j}$ that we fix so that output i can be simulated using a freshly sampled uniform value. □

In order to formally verify this proof with EasyCrypt, we need to prove the equivalence between two programs which share the same inputs $\{a_i\}_{i \in I}$. That is, we show that whatever the observations made by adversary, as soon as they are upper bounded by t, he cannot distinguish between both programs. To proceed, we need to write different derivations of the original program in order to simplify the proof of equivalence for Easycrypt. Thus, we organize the security proof of Proposition 2 as a sequence of games and a sequence of codes.

Proof of Proposition 2 with Easycrypt. We use different colors in the codes to underline the differences with the previous game.

Game 0

The first game represents the original refreshing function RefreshMult which computes the $t+1$ shares c_i without any observation.

```
function RefreshMult(a) :
for i = 0 to t do
  c_i ← a_i;
for i = 0 to t do
  for j = i + 1 to t
    r_{i,j} ← $;
    c_i ← c_i ⊕ r_{i,j};
    c_j ← c_j ⊖ r_{i,j};
return c
```

Game 1

The second game also represents the original refreshing function but on which the attacker can make up to t observations whose t_1 on the internal variables and $t_2 = t - t_1$ on the outputs. All the observations are given in the argument of the function and for each intermediate variable, if it belongs to the observations, it is stored in a table. We refer to this function as R_0.
As explained in Section 4.3, there are three kinds of internal observations a_i, $r_{i,j}$, $c_{i,j}$ and the final c_i. With this first function, we can build the subscript I. If we define by $\mathsf{ind_k}$ the k^{th} index of a variable, $I = \mathsf{ind_1}(a_i) \cup \mathsf{ind_1}(r_{i,j}) \cup \mathsf{ind_1}(c_{i,j})$. In this step, we formally verify with Easycrypt that the cardinal of I is at most t_1 and we show that RefreshMult and R_0 computes the same outputs.

Function $\mathsf{R_0}(\boldsymbol{a}, \boldsymbol{O})$:
for $i = 0$ **to** d **do**
  $c_i \leftarrow a_i$;
  if $(a_i \in O)$ **then** $\overline{a_i} \leftarrow a_i$;
for $i = 0$ **to** d **do**
  for $j = i + 1$ **to** d **do**
    $r_{i,j} \leftarrow \$$;
    if $(r_{i,j} \in O)$ **then** $\overline{r_{i,j}} \leftarrow r_{i,j}$;
    $c_i \leftarrow c_i \oplus r_{i,j}$;
    if $(c_{i,j} \in O)$ **then** $\overline{c_{i,j}} \leftarrow c_i$;
    $c_j \leftarrow c_j \ominus r_{i,j}$;
    if $(c_{j,i} \in O)$ **then** $\overline{c_{j,i}} \leftarrow c_j$;
for $i = 0$ **to** d **do**
    if $(c_i \in O)$ **then** $\overline{c_i} \leftarrow c_i$;
return $\boldsymbol{c}$

Game 2

We make a few changes on R_0:

- all the fresh random values are generated at the beginning of the function with the random oracle-like function Sample and stored them in a matrix,
- all the values corresponding to the internal observations are computed,
- the outputs $(c_i)_{i \in I}$ are computed,
- the outputs $(c_i)_{i \notin I}$ are computed.

We refer to this new function as R_1 and we use EasyCrypt to prove that R_0 and R_1 are equivalent if they share the same inputs $(a_i)_{i \in I}$.

Function $\mathsf{R_1}(\boldsymbol{a}, \boldsymbol{O})$:
for $i = 0$ **to** t **do**
  for $j = i + 1$ **to** t **do**
    $\overline{r_{i,j}} \leftarrow \mathsf{Sample}(i, j)$;
for $i = 0$ **to** t **do**
  if $(a_i \in O)$ **then** $\overline{a_i} \leftarrow a_i$;
for $i = 0$ **to** t **do**
  if $(i \in I)$ **then**
    for $j = 0$ **to** t **do**
      if $(r_{i,j} \in O)$ **then**
        $r_{i,j} \leftarrow \overline{r_{i,j}}$;
      if $(c_{i,j} \in O)$ **then**
        $c_{i,j} \leftarrow \mathsf{SumCij}(i, j)$;
    if $(c_i \in O)$ **then**
      $c_i \leftarrow \mathsf{SumCij}(i, t)$;
for $i = 0$ **to** d **do**
  if $(i \notin I)$ **then**
    if $(c_i \in O)$ **then**
      $c_i \leftarrow \mathsf{SumCij}(i, t)$;
return $\boldsymbol{c}$

Function $\mathsf{SumCij}(i, j)$:
$s \leftarrow a_i$;
for $k = 0$ **to** j **do**
  if $(i < k)$ **then** $s \leftarrow s \oplus \overline{r_{i,k}}$;
  elseif $(i > k)$ **then** $s \leftarrow s \ominus \overline{r_{k,i}}$;
return s;

Game 3

We now make a conceptual change to Game 2. The generation of the random values is delayed as late as possible, i.e., that is just before their first use. We refer to this new function as R_2. We prove the equivalence between Games 2 and 3 if they share the same inputs $\{a_i\}_{i\in I}$ with a generic proof made in Eager-Lazy.

Function $\mathsf{SumCij}(i,j)$:
$s \leftarrow a_i$;
for $k = 0$ **to** j **do**
  if $(i < k)$ **then** $s \leftarrow s \oplus \mathsf{Sample}(i,k)$;
  elseif $(i > k)$ **then** $s \leftarrow s \ominus \mathsf{Sample}(k,i)$;
return s;

Function $\mathsf{R}_2(\boldsymbol{a}, \boldsymbol{O})$:
for $i = 0$ **to** t **do**
  if $(a_i \in O)$ **then** $\overline{a_i} \leftarrow a_i$;
for $i = 0$ **to** t **do**
  if $(i \in I)$ **then**
    for $j = 0$ **to** d **do**
      if $(r_{i,j} \in O)$ **then**
        $r_{i,j} \leftarrow \mathsf{Sample}(i,j)$;
      if $(c_{i,j} \in O)$ **then**
        $c_{i,j} \leftarrow \mathsf{SumCij}(i,j)$;
    if $(c_i \in O)$ **then**
      $c_i \leftarrow \mathsf{SumCij}(i,t)$;
for $i = 0$ **to** d **do**
  if $(i \notin I)$ **then**
    if $(c_i \in O)$ **then**
      $c_i \leftarrow \mathsf{SumCij}(i,t)$;
return $\boldsymbol{c}$;

Game 4

In this game, we make a significant change in the computation of the c_i for all the i which are not in I. Concretely, we show that there exist a non empty set of indexes L such that $\forall \ell \in L$, $r_{i,\ell}$ is not assigned yet. Then, instead of computing the c_i (for $i \notin I$) as follows:

$$\forall \ell \in L,\ r_{i,\ell} \leftarrow \$,$$
$$c_i \leftarrow a_i \oplus \bigoplus_{j=0}^{i-1} r_{i,j} \ominus \bigoplus_{j=i+1}^{t} r_{i,j},$$

we make the following change:

$$c_i \leftarrow \$,$$
$$\forall \ell \in L\backslash\{k\},\ r_{i,\ell} \leftarrow \$,$$
$$\text{if } (i < k),\quad r_{i,k} \leftarrow c_i \ominus a_i \ominus \bigoplus_{j=0, j\neq k}^{i-1} r_{i,j} \oplus \bigoplus_{j=i+1, j\neq k}^{t} r_{i,j},$$
$$\text{if } (i > k),\quad r_{i,k} \leftarrow \ominus c_i \oplus a_i \oplus \bigoplus_{j=0, j\neq k}^{i-1} r_{i,j} \ominus \bigoplus_{j=i+1, j\neq k}^{t} r_{i,j}.$$

We prove the equivalence between Game 3 and Game 4 with EasyCrypt when functions R_2 and R_3 share the same inputs $\{a_i\}_{i\in I}$. The most critical part of this step is undoubtedly to ensure that the subscript J contains at least one index. To do so, we need to show that the elements $r_{i,\ell}$ with $\ell \in L$ were not already used and won't be reused anywhere. Eventually, we formally prove that the results of R_3, which represents the final simulator, only depends on the inputs $\{a_i\}_{i\in I}$.

Function $\mathsf{SumCij}(i, j)$:
$s \leftarrow a_i$;
for $k = 0$ **to** j **do**
  if $(i < k)$ **then**
    $s \leftarrow s \oplus \mathsf{Sample}(i, k)$;
  else if $(i > k)$ **then**
    $s \leftarrow s \ominus \mathsf{Sample}(k, i)$;
return s;

Function $\mathsf{SetCi}(i)$;
$s \leftarrow a_i$; $k \leftarrow 0$;
while $((k \leqslant t) \wedge ((i == k) \vee$
  $(k < i \wedge ((i, k) \in \mathrm{dom}\ r)) \vee$
  $(i < k \wedge ((k, i) \in \mathrm{dom}\ r)))$ **do**
  if $(i < k)$ **then** $s \leftarrow s \oplus r_{i,k}$;
  else then $s \leftarrow s \ominus r_{i,k}$;
  $k \leftarrow k + 1$;
$k' \leftarrow k$; $r' \leftarrow \$$;
for $k = k'$ **to** t **do**
  $s \leftarrow s \oplus r_{i,k}$;
if $(i < k')$ **then** $r_{i,k'} \leftarrow s \oplus r'$;
else $r_{k',i} \leftarrow s \ominus r'$;
return r';

Function $\mathsf{R}_3(\boldsymbol{a}, \boldsymbol{O})$:
for $i = 0$ **to** t **do**
  if $(a_i \in O)$ **then**
    $\overline{a_i} \leftarrow a_i$;
for $i = 0$ **to** t **do**
  if $(i \in I)$ **then**
    for $j = 0$ **to** t **do**
      if $(r_{i,j} \in O)$ **then**
        $r_{i,j} \leftarrow \mathsf{Sample}(i, j)$;
      if $(c_{i,j} \in O)$ **then**
        $c_{i,j} \leftarrow \mathsf{SumCij}(i, j)$;
    if $(c_i \in O)$ **then**
      $c_i \leftarrow \mathsf{SetCi}(i)$;
for $i = 0$ **to** t **do**
  if $(i \notin I)$ **then**
    if $(c_i \in O)$ **then**
      $c_i \leftarrow \mathsf{SetCi}(i)$;
return $\boldsymbol{c}$;

□

Finally, we have formally proved that an adversary could not distinguish between two programs which share the same inputs $\{a_i\}_{i \in I}$ with at most t observations and that the cardinal of I was upper bounded by the number of internal observations d_1.

In the following, we call $\mathsf{Refresh}_t$ (omitting the index when clear from context) the core gadget implemented using Algorithm 11.

4.3.2 Secure Multiplication Algorithms

We focus here on two multiplication algorithms: the SecMult algorithm introduced in [RP10] and Algorithm 4 from [CPRR14], which computes function $h : x \mapsto x \odot g(x)$ for some linear function g.

SecMult Algorithm

We show Rivain and Prouff's multiplication algorithm SecMult in Algorithm 12 [RP10]. Note that this algorithm is correct and secure for the computation of any internal, associative and commutative operation $\otimes$ that distributes over addition in $\mathbb{K}$ $(\oplus)$. This includes, for example, field multiplication $\odot$ in $\mathsf{GF}(2^n)$, and multiplication & in the boolean ring $\mathbb{B}^n$. In addition to being t-NI as claimed by Rivain and Prouff [RP10], we show that it is also t-SNI (Proposition 3). This stronger security property makes it valuable for performance, since it may reduce the number of required mask refreshing gadgets required to compositionally prove security.

Proposition 3. *Algorithm 12 is t-SNI.*

We now prove Proposition 3 informally. A formal proof has also been done in EasyCrypt on the same model than for the multiplication-based refresh algorithm.

Algorithm 12 Secure Multiplication Algorithm [RP10]

1: **function** SecMult($\boldsymbol{a}, \boldsymbol{b}$)
2: **for** $i = 0$ to t **do**
3: $c_i \leftarrow a_i \otimes b_i$
4: **for** $i = 0$ to t **do**
5: **for** $j = i + 1$ to t **do**
6: $r \xleftarrow{\$} \mathbb{K}$ /* this random value is referred to as $r_{i,j}$ */
7: $c_i \leftarrow c_i \oplus r$ /* referred by $c_{i,j}$ */
8: $r \leftarrow a_i \otimes b_j \ominus r \oplus a_j \otimes b_i$ /* referred by $r_{j,i}$ */
9: $c_j \leftarrow c_j \oplus r$ /* referred by $c_{j,i}$ */
10: **return** $\boldsymbol{c}$

Proof. As for the multiplication-based mask refreshing algorithm, we first focus on the functional correctness by proving that the algorithm implements the field multiplication function: $[\![\mathsf{SecMult}(a, b)]\!] = [\![a \otimes b]\!]$. Similarly, this can be easily verified by simplifying the sums and by the ring axioms (in particular distributivity of $\otimes$ over $\oplus$).

To prove t-SNI, we build a simulator. Let $\Omega = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$ be a t-admissible set of observations, and let $d_1 = |\mathcal{O}^{(int)}|$ and $d_2 = |\mathcal{O}^{(out)}|$. Note that $d_1 + d_2 \leq t$. Our goals are to find two d_1-compatible sets $\mathcal{S}$ and $\mathcal{S}'$, and to construct a perfect simulator that uses only shares $\mathcal{S}$ of $\boldsymbol{a}$ and shares $\mathcal{S}'$ of $\boldsymbol{b}$.

First, we identify which variables are internals and which are outputs. We directly split the internals in four groups for the needs of the proof:

- *Group 1:* the a_i, the b_i, and the $a_i \otimes b_i$,
- *Group 2:* the $c_{i,j}$ (resp. $c_{j,i}$) which corresponds to the value of the variable c_i (resp. c_j) at iteration i, j,
- *Group 3:* the $r_{i,j}$ (the first value of r at iteration i, j), and the $r_{j,i}$ (the second value of r at iteration i, j),
- *Group 4:* the $a_i \otimes b_j$ and the $a_i \otimes b_j \ominus r_{i,j}$.

The output variables are the final values of c_i (*i.e.*, $c_{i,t}$).

As the algorithm takes two inputs a and b, we define two subscripts $\mathcal{S}_a$ and $\mathcal{S}_b$ which will contain the indexes of each input's shares that will be further used for the simulation of the observations. For each observation in the first or the second group, we add the index i to $\mathcal{S}_a$ and to $\mathcal{S}_b$. For each observation in the third or the fourth group: if the index i is already in $\mathcal{S}_a$, we add the index j to $\mathcal{S}_a$, otherwise we add the index i to $\mathcal{S}_a$ and if the index i is already in $\mathcal{S}_b$, we add the index j to $\mathcal{S}_b$, otherwise we add the index i to $\mathcal{S}_b$. It is clear that the final sets $\mathcal{S}_a$ and $\mathcal{S}_b$ contain each one at most d_1 indexes with $d_1 \leq t$.

We now construct the simulator. For clarity, observe that the SecMult algorithm can be equivalently represented using the following matrix:

$$\begin{pmatrix} a_0 \otimes b_0 & 0 & r_{0,1} & r_{0,2} & \cdots & r_{0,t} \\ a_1 \otimes b_1 & r_{1,0} & 0 & r_{1,2} & \cdots & r_{1,t} \\ a_2 \otimes b_2 & r_{2,0} & r_{2,1} & 0 & \cdots & r_{2,t} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_t \otimes b_t & r_{t,0} & r_{t,1} & r_{t,2} & \cdots & 0 \end{pmatrix}.$$

In this setting, $c_{i,j}$ corresponds to the partial sum of the $j+2$ first elements of line i. For each variable $r_{i,j}$ $(i < j)$ entering in the computation of an observation, we assign it a fresh random value. Then, for each observation in the first group, a_i and b_i are provided to the simulator (by definition of $\mathcal{S}_a$ and $\mathcal{S}_b$) thus the observation is perfectly simulated. For an observation in the third group, we distinguish two cases. If $i < j$, $r_{i,j}$ is already assigned to a fresh random value. If $i > j$, either $(i,j) \in \mathcal{S}_a \wedge \mathcal{S}_b$ and the observation is perfectly simulated from $r_{j,i}, a_i, b_i, a_j$ and b_j or $r_{j,i}$ does not enter in the computation of any internal variable that was observed and $r_{i,j}$ (line 8) is assigned to a fresh random value. Each observation made in the fourth group is perfectly simulated using $r_{i,j}$, a_i and b_j. As for an observation in the second group, the corresponding variable is a partial sum composed of a product $a_i \otimes b_i$ and of variables $r_{i,j}$. Since a_i and b_i are provided to the simulator in this case, we focus on each remaining $r_{i,j}$. Each one of them such that $i < j$ is already assigned to a fresh random value. For the others, if $r_{j,i}$ enters in the computation of any other internal observation, then $(i,j) \in \mathcal{S}_a \wedge \mathcal{S}_b$ and $r_{i,j}$ is simulated with $r_{j,i}, a_i, b_i, a_j$ and b_j. Otherwise, $r_{i,j}$ is assigned to a fresh random value.

We still have to simulate the observations on output variables. We start with the ones whose intermediate sums (group 2) are also observed. For each such variable c_i, the biggest partial sum which is observed is already simulated. Thus, we consider the remaining terms $r_{i,j}$. Each one of them such that $i < j$ is already assigned to a fresh random value. For the others, either $(i,j) \in (\mathcal{S}_a \cap \mathcal{S}_b)$ and c_i is perfectly simulated from $r_{j,i}$, a_i, b_i, a_j and b_j or $r_{j,i}$ does not enter in the computation of any internal variable observed and c_i is assigned to a fresh random value. We now consider output observations whose partial sums are not observed. Each of them is composed of t $r_{i,j}$. And at most one of them can enter in the computation of each other variable c_i. Since, we already considered (without this one) at most $t-1$ observations, at least one $r_{i,j}$ does not enter in the computation of any other observed variable. Thus, c_i is assigned to a fresh random value. □

MultLinear Algorithm to Compute $x \otimes g(x)$

Coron *et al.* [CPRR14] introduce an extended multiplication algorithm which we recall as Algorithm 13, and prove that it is t-NI. In fact, their proof even shows that the gadget is t-TNI, but the authors do not identify this stronger property. We show here that this algorithm is in fact t-SNI and therefore dispenses the user from having to refresh its output's masks.

Proposition 4. *Algorithm 13 is t-SNI.*

We provide a pen-and-paper proof of Proposition 4. This proof has not yet been formalized in EasyCrypt, and we leave this as future work if a fully certified compiler or verifier is desired.

Proposition 4. For functional correctness, we have to prove that the algorithm implements the function: $[\![h(x)]\!] = [\![x \otimes g(x)]\!]$. This can be seen by expanding its results and simplifying the field expressions.

Let $\Omega = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$ be a t-admissible set of observations, and let $d_1 = |\mathcal{O}^{(int)}|$ and $d_2 = |\mathcal{O}^{(out)}|$. Note that $d_1 + d_2 \leq t$. To prove t-SNI, we need to: (i) find a d_1-compatible set $\mathcal{S}$, (ii) construct a perfect simulator that uses only shares $\mathcal{S}$ of the input.

First, we identify which variables are internals and which are outputs. For the sake of clarity, we denote by $r_{i,j}$ and $r'_{i,j}$ the random variables for which $i < j$. Internals are:

1. the a_i, the $g(a_i)$ and the $a_i \otimes g(a_i)$ that only depend on a_i,
2. the $r'_{i,j}$, the $g(r'_{i,j})$, the $a_i \otimes g(r'_{i,j})$ and the $r'_{i,j} \otimes g(a_i)$ that depend on both a_i and $r'_{i,j}$,

Algorithm 13 $h : x \mapsto x \otimes g(x)$ [CPRR14, Algorithm 4]

function MultLinear($\boldsymbol{a}$)
  for $i = 0$ **to** t **do**
    for $j = i + 1$ **to** t **do**
      $r_{i,j} \xleftarrow{\$} \mathbb{K}$
      $r'_{i,j} \xleftarrow{\$} \mathbb{K}$
      $t \leftarrow a_i \otimes g(r'_{i,j}) \ominus r_{i,j}$
      $t \leftarrow t \oplus (r'_{i,j} \otimes g(a_i))$
      $t \leftarrow t \oplus (a_i \otimes g(a_j \ominus r'_{i,j})$
      $t \leftarrow t \oplus ((a_j \ominus r'_{i,j}) \otimes g(a_i))$
      $r_{j,i} \leftarrow t$ /* $r_{j,i} = r_{i,j} \oplus a_i \otimes g(a_j) \oplus a_j \otimes g(a_i)$ */
  for $i = 0$ **to** t **do**
    $c_i \leftarrow a_i \otimes g(a_i)$
    for $j = 0$ **to** t, $j \neq i$ **do**
      $c_i \leftarrow c_i \oplus r_{i,j}$ /* referred to as $c_{i,j}$ */
  return $\boldsymbol{c}$

3. the $a_j \ominus r'_{i,j}$, the $g(a_j \ominus r'_{i,j})$, the $a_i \otimes g(a_j \ominus r'_{i,j})$ and the $(a_j \ominus r'_{i,j}) \otimes g(a_i)$ that depend on a_i, a_j and $r'_{i,j}$,
4. the $r_{i,j}$, the $a_i \otimes g(r'_{i,j}) \ominus r_{i,j}$, the $a_i \otimes g(r'_{i,j}) \oplus r'_{i,j} \otimes g(a_i) \ominus r_{i,j}$, the $a_i \otimes g(r'_{i,j}) \oplus r'_{i,j} \otimes g(a_i) \oplus a_i \otimes g(a_j \ominus r'_{i,j}) \ominus r_{i,j}$ and the $a_i \otimes g(a_j) \oplus a_j \otimes g(a_i) \ominus r_{i,j}$ that are invertible in $r_{i,j}$,
5. the $a_i \otimes g(a_i) \oplus \bigoplus_{j=0}^{j_0}(a_i \otimes g(a_j) \oplus a_j \otimes g(a_i) \oplus r_{j,i})$ with $1 \leqslant j_0 \leqslant i-1$ and the $a_i \otimes g(a_i) \oplus \bigoplus_{j=0}^{i-1}(a_i \otimes g(a_j) \oplus a_j \otimes g(a_i) \ominus r_{j,i}) \oplus \bigoplus_{j=i+1}^{j_0} r_{i,j}$ with $i < j_0 < d$.

Outputs are the final values of c_i (i.e. $c_{i,t}$).

We define $\mathcal{S}$ as follows. For each observation among the first or the fifth group, we add the index i to $\mathcal{S}$. For each observation in groups 2, 3 or 4, we add the index j to $\mathcal{S}$ if $i \in \mathcal{S}$, otherwise we add i to $\mathcal{S}$. Since we only consider internal observations when constructing $\mathcal{S}$, it is clear that $\mathcal{S}$ contains at most d_1 indexes. We now construct the simulator. For clarity, observe that the MultLinear algorithm can be equivalently represented using the following matrix:

$$\begin{pmatrix} a_0 \otimes g(a_0) & 0 & r_{0,1} & r_{0,2} & \cdots & r_{0,t} \\ a_1 \otimes g(a_1) & f(a_0,a_1) \ominus r_{0,1} & 0 & r_{1,2} & \cdots & r_{1,t} \\ a_2 \otimes g(a_2) & f(a_0,a_2) \ominus r_{0,2} & f(a_1,a_2) \ominus r_{1,2} & 0 & \cdots & r_{2,t} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_d \otimes g(a_d) & f(a_0,a_t) \ominus r_{0,t} & f(a_1,a_t) \ominus r_{1,t} & f(a_2,a_t) \ominus r_{2,t} & \cdots & 0 \end{pmatrix}$$

with $f(x,y) = x \otimes g(y) \oplus y \otimes g(x)$. In this setting, $c_{i,j}$ corresponds to the partial sum of the $j+2$ first elements of line i. For each variable $r_{i,j}$ or $r'_{i,j}$ entering in the computation of an observation, we assign it a fresh random value. Then, for each observation in the first group, a_i is provided to the simulator (by definition of $\mathcal{S}$) thus the observation is perfectly simulated. For each observation in the second group, a_i is provided to the simulator and $r'_{i,j}$ is already assigned to a random value thus the observation is perfectly simulated. For each observation in the third group, we consider two cases. If $j \in \mathcal{S}$, then a_j is also provided to the simulator and the observation can be perfectly simulated with a_i, a_j and $r'_{i,j}$. If $j \notin \mathcal{S}$, then $r'_{i,j}$ does not enter in the computation of any other observation and $a_j \ominus r'_{i,j}$ can be assigned to a fresh random value. The observation is thus perfectly simulated with a_i. For each observation in the fourth

group, we also consider two cases. If $j \in \mathcal{S}$, then a_j is also provided to the simulator and the observation is perfectly simulated using a_i, a_j, $r_{i,j}$ and $r'_{i,j}$. If $j \notin \mathcal{S}$, then $r_{i,j}$ does not enter in the computation of any other observation. Thus, since this observation is invertible with respect to $r_{i,j}$ it can be perfectly simulated by a fresh random value. Finally, for each observation in the fifth group, we consider the different terms. The first product $a_i \otimes g(a_i)$ can be perfectly simulated with a_i. Then, the sum of $r_{i,j}$ can be perfectly simulated with the corresponding random values. As for the sum of $(a_i \otimes g(a_j) \oplus a_j \otimes g(a_i) \ominus r_{j,i})$ we consider two cases. If $j \in \mathcal{S}$, this sum can be perfectly simulated with a_i, a_j and $r_{j,i}$. Otherwise, $r_{j,i}$ does not enter in the computation of any other observation and we can simulate the entire term using a fresh random value.

We still have to simulate the observed output values for rows on which no internal values are observed. Remark that simulating the ith line also necessarily fixed the value of all random variables appearing in the ith column (so that dependencies between variables are preserved). After internal observations are simulated, at most d_1 lines of the matrix are fully filled. Therefore, at least $t - d_1 \geqslant d_2$ (with $d_2 > 0$ if the adversary makes an output observation) random values are not yet simulated on lines on which no internal observations are made. For each output observation made on one such line (say i), we can therefore pick a different $r_{i,j}$ that we fix so that output i can be simulated using a freshly sampled uniform value. □

Remark 6. *Note that the first part of the proof, involving the simulation of internal observations only, was initially made in [CPRR14] to prove the gadget t-NI. However, the authors omitted one internal variable: $a_i \otimes g(r'_{i,j}) \ominus r_{i,j}$. We thus fix the proof of t-NI and further extend it to t-SNI by simulating outputs without any additional input shares.*

Remark 7. *In the compiler, we use the second algorithm provided by Coron* et al. *(Algorithm 5 in [CPRR14]) to compute the multiplication $x \otimes g(x)$ using a table. The algorithm is not exactly the same but the security proof is* a priori *similar. Namely, even if the intermediate variables are quite different, they can be classified like the ones of Algorithm 4 with the same dependencies (group 5 is exactly the same for both algorithms). Since we use these same dependencies in the aforementioned proof to explain the simulation of all possible sets of observations independently from the secret, we can reasonably claim that Algorithm 5 from [CPRR14] is also t-SNI.*

4.4 Simple Compositional Proofs of Security

In this section, we show how the various notions of non-interference, and more specifically the notion of t-SNI can be used to obtain compositional proofs of security for large circuits. We start by abstractly describing a generic proof method for compositionally proving a circuit t-NI based only on affine, t-SNI, and t-TNI properties of core gadgets and checking simple arithmetic side-conditions. We then illustrate it by detailing a compositional proof of security for a masked version of an inversion algorithm in $\mathsf{GF}(2^8)$ [RP10].

4.4.1 Securely Composing Secure Gadgets

We consider a masked circuit P constructed by composition of n chosen core gadgets (affine gadgets or those t-SNI gadgets discussed in Section 4.3), and a topological ordering on P's gadgets (using 1, rather than n, to denote the last gadget according to that ordering). Let Ω be an arbitrary t-admissible observation set on P. We split Ω according to whether observations occur on P's output bundles (we name those $\mathcal{O}^{(out)^i}$, one for each of P's o outputs) or are internal to a gadget in P's (we name those $\mathcal{O}^{(int)^i}$, one for each core gadget in P). The t-admissibility

constraint on Ω implies the following *global constraint* on its components:

$$|\Omega| = \sum_{1 \leq i \leq o} |\mathcal{O}^{(out)^i}| + \sum_{1 \leq i \leq n} |\mathcal{O}^{(int)^i}| \leq t$$

The process starts with:

- an initial set of constraint C that only contains the global constraint;
- for each wire that serves as a connection between core gadgets in P, a set ${\mathcal{O}^{(out)^i}}_j$, that intuitively corresponds to gadget i's jth output bundle; all ${\mathcal{O}^{(out)^i}}_j$ are initially empty except for those that correspond to P's output bundles, to which the corresponding $\mathcal{O}^{(out)^k}$ (or union of $\mathcal{O}^{(out)^k}$s, if the same bundle is used multiple times as output) is assigned.

Starting from Gadget 1 (the last gadget according to the chosen topological ordering) and the initial state described above, and for each gadget G^i (progressing back through the chosen ordering), the following operations are performed:

- check that the side condition for gadget G^i's non-interference property follows from the constraints in C:
  - if G^i is t-TNI or t-SNI, check that $C \Rightarrow |\mathcal{O}^{(int)^i}| + \sum_{1 \leq j \leq o_i} |{\mathcal{O}^{(out)^i}}_j| \leq t$;
  - if G^i is affine, the side-condition is trivial.
- using the corresponding non-interference property, derive a set of shares for each of G^i's inputs that suffice to simulate the internal and output leakage in G^i, and add the corresponding constraints to C:
  - if G^i is t-TNI, a fresh set of indexes $\mathcal{S}^i_j$ is introduced for each input bundle j, and the constraint $|\mathcal{S}^i_j| \leq |\mathcal{O}^{(int)^i}| + \sum_{1 \leq k \leq o_i} |{\mathcal{O}^{(out)^i}}_k|$ is added to C for all j;
  - if G^i is t-SNI, a fresh set of indexes $\mathcal{S}^i_j$ is introduced for each input bundle j, and the constraint $|\mathcal{S}^i_j| \leq |\mathcal{O}^{(int)^i}|$ is added to C for all j;
  - if G^i is affine, a fresh set of indexes $\widehat{\mathcal{S}}^i$ is introduced, and the constraint $|\widehat{\mathcal{S}}^i| \leq |\mathcal{O}^{(int)^i}|$ is added to C.
- the newly computed sets of shares on G^i's inputs are propagated to become output observations on the gadget from which they come (except if they correspond to C's inputs):
  - if G^i is t-TNI or t-SNI, for all j, if G^i's jth input bundle is connected to another gadget $G^{i'}$'s kth output bundle, set ${\mathcal{O}^{(out)^{i'}}}_k \leftarrow {\mathcal{O}^{(out)^{i'}}}_k \cup \mathcal{S}^i_j$;
  - if G^i is affine, if any of G^i's input bundles is connected to another gadget $G^{i'}$'s kth output bundle, set ${\mathcal{O}^{(out)^{i'}}}_k \leftarrow {\mathcal{O}^{(out)^{i'}}}_k \cup \widehat{\mathcal{S}}^i \cup \mathcal{O}^{(out)^i}$.

If, at any point in this process, a side-condition fails to check, the circuit is not compositionally secure (although it may still be t-SNI). However, the only case in which such a failure could occur is if a gadget's output serves as input to multiple gadgets. In this case, the failure in checking the side-condition can in fact be used to automatically insert a mask refreshing gadget as needed, and resume the proof from that point on.

Once all gadgets in the circuit are simulated as described, one can then easily check whether the accumulated constraints in C are sufficient to guarantee that the set of shares associated with each one of P's input bundles is of a size smaller than or equal to t.

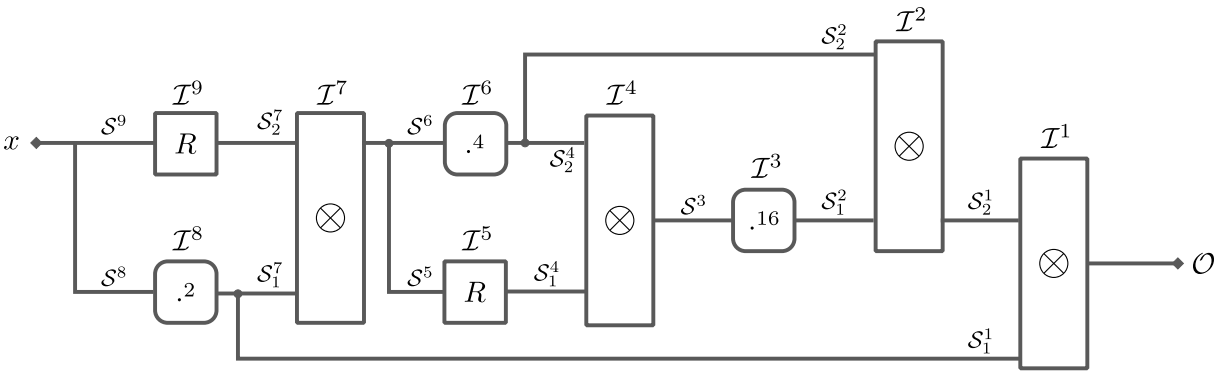


Figure 4.4: Gadget $\cdot^{254}$

4.4.2 An Example: AES Inversion Algorithm by Rivain and Prouff

We now illustrate this process on Rivain and Prouff's algorithm for computing inversion in $\mathsf{GF}(2^8)$ [RP10, CPRR14] when implemented over $t+1$ shares. A circuit implementing this operation securely is shown in Figure 4.4. We use a simple composition argument to prove that this inversion is t-SNI, relying on the fact that the multiplication gadget $\otimes$ and the refreshing gadget R (that is, Refresh) are both t-SNI. We recall that the function $x \mapsto x^{2^n}$ for any n is linear in binary fields and rely on affine gadgets $\cdot^2$, $\cdot^4$ and $\cdot^{16}$ to compute the corresponding (linear) functionalities.

Theorem 8. *Gadget $\cdot^{254}$, shown in Figure 4.4, is t-SNI.*

Proof. The proof follows the process described above. We detail it here to illustrate the compositional proof process on a practical example.

Let $\Omega = (\bigcup_{1\leq i\leq 9} \mathcal{O}^{(int)^i}, \mathcal{O}^{(out)})$ be a t-admissible observation set. In particular, we know that the *global constraint* $|\mathcal{O}^{(out)}| + \sum_{1\leq i\leq 9} |\mathcal{O}^{(int)^i}| \leq t$ holds. The proof constructs the simulator by simulating each gadget in turn, starting from the final multiplication (Gadget 1) and progressing from right to left and upward.

Gadget 1 - since $\otimes$ is t-SNI and $|\mathcal{O}^{(int)^1} \cup \mathcal{O}^{(out)}| \leq t$ (by the global constraint), we know that there exist observation sets $\mathcal{S}_1^1$, $\mathcal{S}_2^1$ such that $|\mathcal{S}_1^1| \leq |\mathcal{O}^{(int)^1}|$, $|\mathcal{S}_2^1| \leq |\mathcal{O}^{(int)^1}|$ and Gadget 1 is $((\mathcal{S}_1^1, \mathcal{S}_2^1), (\mathcal{O}^{(int)^1}, \mathcal{O}^{(out)}))$-NI; (note that $\mathcal{S}_1^1$ and $\mathcal{S}_2^1$ become output observations on Gadgets 8 and 2, respectively)

Gadget 2 - since $\otimes$ is t-SNI and $|\mathcal{O}^{(int)^2} \cup \mathcal{S}_2^1| \leq t$ (by the simulation of Gadget 1 and the global constraint, we know that there exist observations sets $\mathcal{S}_1^2$, $\mathcal{S}_2^2$ such that $|\mathcal{S}_1^2| \leq |\mathcal{O}^{(int)^2}|$, $|\mathcal{S}_2^2| \leq |\mathcal{O}^{(int)^2}|$, and Gadget 2 is $((\mathcal{S}_1^2, \mathcal{S}_2^2), (\mathcal{O}^{(int)^2}, \mathcal{S}_2^1))$-NI;

Gadget 3 - since $\cdot^{16}$ is affine, we know that there exists an observation set $\mathcal{S}^3$ such that $|\mathcal{S}^3| \leq |\mathcal{O}^{(int)^3}| + |\mathcal{S}_1^2| \leq |\mathcal{O}^{(int)^3}| + |\mathcal{O}^{(int)^2}|$ (by the simulation of Gadget 2) and Gadget 3 is $(\mathcal{S}^3, (\mathcal{O}^{(int)^3}, \mathcal{S}_1^2))$-NI;

Gadget 4 - since $\otimes$ is t-SNI and $|\mathcal{O}^{(int)^4} \cup \mathcal{S}^3| \leq t$ (by the simulation of Gadget 3 and the global constraint), we know that there exist observation sets $\mathcal{S}_1^4$, $\mathcal{S}_2^4$ such that $|\mathcal{S}_1^4| \leq |\mathcal{O}^{(int)^4}|$, $|\mathcal{S}_2^4| \leq |\mathcal{O}^{(int)^4}|$, and Gadget 4 is $((\mathcal{S}_1^4, \mathcal{S}_2^4), (\mathcal{O}^{(int)^4}, \mathcal{S}^3))$-NI;

Gadget 5 - since RefreshMult is t-SNI and $|\mathcal{O}^{(int)^5} \cup \mathcal{S}_1^4| \leq t$ (by the simulation of Gadget 4 and the global constraint), we know that there exist observation sets $\mathcal{S}_1^5$, $\mathcal{S}_2^5$ such that $|\mathcal{S}_1^5| \leq |\mathcal{O}^{(int)^5}|$, $|\mathcal{S}_2^5| \leq |\mathcal{O}^{(int)^5}|$, and Gadget 5 is $((\mathcal{S}_1^5, \mathcal{S}_2^5), (\mathcal{O}^{(int)^5}, \mathcal{S}_1^4))$-NI;

Gadget 6 - since $\cdot^4$ is affine, we know that there exists an observation set $\mathcal{S}^6$ such that $|\mathcal{S}^6| \leq |\mathcal{O}^{(int)^6}| + |\mathcal{S}_2^2 \cup \mathcal{S}_2^4| \leq |\mathcal{O}^{(int)^6}| + |\mathcal{O}^{(int)^2}| + |\mathcal{O}^{(int)^4}|$ (by the simulation of Gadgets 2 and 4) and Gadget 6 is $(\mathcal{S}^6, (\mathcal{O}^{(int)^6}, \mathcal{S}_2^2 \cup \mathcal{S}_2^4))$-NI;

Gadget 7 - since $\otimes$ is t-SNI and $|\mathcal{O}^{(int)^7} \cup \mathcal{S}^5 \cup \mathcal{S}^6| \leq t$ (by the simulation of Gadgets 5 and 6 and the global constraint), we know that there exist observation sets $\mathcal{S}_1^7$, $\mathcal{S}_2^7$ such that $|\mathcal{S}_1^7| \leq |\mathcal{O}^{(int)^7}|$, $|\mathcal{S}_2^7| \leq |\mathcal{O}^{(int)^7}|$, and Gadget 7 is $((\mathcal{S}_1^7, \mathcal{S}_2^7), (\mathcal{O}^{(int)^7}, \mathcal{S}^5 \cup \mathcal{S}^6))$-NI;

Gadget 8 - since $\cdot^4$ is affine, we know that there exists an observation set $\mathcal{S}^8$ such that $|\mathcal{S}^8| \leq |\mathcal{O}^{(int)^8}| + |\mathcal{S}_1^1 \cup \mathcal{S}_1^7| \leq |\mathcal{O}^{(int)^8}| + |\mathcal{O}^{(int)^1}| + |\mathcal{O}^{(int)^7}|$ (by the simulation of Gadgets 1 and 7) and Gadget 8 is $(\mathcal{S}^8, (\mathcal{O}^{(int)^8}, \mathcal{S}_1^1 \cup \mathcal{S}_1^7))$-NI;

Gadget 9 - since $\otimes$ is t-SNI and $|\mathcal{O}^{(int)^9} \cup \mathcal{S}_2^7| \leq t$ (by the simulation of Gadget 7 and the global constraint), we know that there exists an observation set $\mathcal{S}^9$ such that $|\mathcal{S}^9| \leq |\mathcal{O}^{(int)^9}|$, and Gadget 9 is $(\mathcal{S}^9, (\mathcal{O}^{(int)^9}, \mathcal{S}^7))$-NI;

Each of these steps gives us the existence of a simulator for the relevant gadget. Composing them together constructs a simulator for the whole circuit that expects $|\mathcal{S}^8 \cup \mathcal{S}^9|$ shares of x. Since we have $|\mathcal{S}^8| \leq |\mathcal{O}^{(int)^7}| + |\mathcal{O}^{(int)^1}|$ and $|\mathcal{S}^9| \leq |\mathcal{O}^{(int)^9}|$, we can conclude that $|\mathcal{S}^8 \cup \mathcal{S}^9| \leq \sum_{1 \leq i \leq 9} |\mathcal{O}^{(int)^i}|$ and therefore that gadget $\cdot^{254}$ is t-SNI. □

Remark 8. *In the proof of Theorem 8, we do not precisely keep track of the simulation sets for affine gadgets as the more general bounds on their size are sufficient to conclude. In practice, when considering large affine sub-circuits, it is often important to keep track of the precise composition of the simulation sets rather than just their size in order to avoid false negatives.*

4.5 Stronger Composition Results

So far, we have shown that a simple compositional proof system and some machine-checked proofs are sufficient to prove an algorithm secure in Ishai, Sahai and Wagner's *stateless* t-threshold probing model [ISW03]. However, this is not enough to guarantee security against an adversary that may move probes between oracle queries when these queries make use of some shared secret state. For instance, if an adversary can move his probes between two executions of an AES (with related inputs/outputs), then the aforementioned compositional properties do not hold anymore. To protect against such an adversary, the literature [ISW03, CPRR14, DDF14] recommends to protect the circuit using $2t+1$ shares and refresh the entirety of the secret state between oracle queries.

We propose a novel method to protect an algorithm against adaptive probing that does not require doubling the number of shares in the entire circuit. Rather, we only double the number of shares on the state when it is stored between oracle queries. More clearly, the state is stored as $2m = 2t+2$ shares, but computation is performed over $m = t+1$ shares only. To enable this, we rely on algorithms Double and Half (Algorithm 14), that double the number of shares and divide it by 2, respectively, in combination with a mask refreshing gadget over $2m$ shares. The

security proof for this mechanism is made feasible by the compositional proof system presented in this chapter.

Algorithm 14 Robust Mask Refreshing: Double and Half

```
1: function Double(a)
2:     for i = 0 to t do
3:         c_2i ←$ 𝕂
4:         c_2i+1 ← a_i ⊖ c_2i
5:     return c
```

```
1: function Half(a)
2:     for i = 0 to t do
3:         c_i ← a_2i ⊕ a_2i+1
4:     return c
```

For simplicity, we express our composition theorem (Theorem 9) on gadgets that have a single input and a single output that we use to model the state. It is easy to generalize to arbitrary scenarios, taking care to use Double, Refresh and Half as specified on *all* the variables that encode the shared state in both gadgets.

Theorem 9 (Robust Composition)**.** *Given two t-TNI gadgets F and G, for any t-admissible observations set $\Omega^{F'}$ on F' and any t-admissible observation set $\Omega^{G'}$ on G', there exists a $|\Omega^{F'}|$-compatible $\mathcal{S}$ such that the composition $F';G'$ described below is $(\mathcal{S}, (\Omega^{F'}, \Omega^{G'}))$-NI.*

F':
$y \leftarrow F(x)$;
$\bar{y} \leftarrow \mathsf{Double}(y)$;
$\bar{y} \leftarrow \mathsf{Refresh}_{2t+2}(\bar{y})$;

G':
$\bar{y} \leftarrow \mathsf{Refresh}_{2t+2}(\bar{y})$;
$y \leftarrow \mathsf{Half}(\bar{y})$;
$z \leftarrow G(y)$;

Intuitively, any set made of $t_1 \leqslant t$ observations on F' and $t_2 \leqslant t$ observations on G' can be perfectly simulated by only t_1 shares of each F''s input.

Corollary 1. *Any number n of t-TNI gadgets $(G_i)_{0 \leq i < n}$ can be composed in a robust way as outlined in Theorem 9.*

The proof of Theorem 9 makes use of the following facts, that can be used to construct a simulator for $F';G'$ and bound the size of the sets of shares of the inputs it requires. We easily extend the notions of gadget and (S/T)NI to include algorithms such as Half and Double that do not use the same m on inputs and outputs.

- For any observation set Ω^{H} on Half, there exists a $(2|\Omega^{\mathsf{H}}|)$-compatible input projection $\mathcal{S}^{\mathsf{H}}$ such that Half is $(\mathcal{S}^{\mathsf{H}}, \Omega^{\mathsf{H}})$-NI. Indeed, whenever share i of the output is observed, we can give the simulator both shares $2i$ and $2i+1$ of the input.
- For an observation set $\Omega^{\mathsf{D}} = (\mathcal{O}^{(int)}, \mathcal{O}^{(out)})$ on Double, there exists a $(|\mathcal{O}^{(int)}| + |\mathcal{O}^{(out)}|)$-compatible input projection $\mathcal{S}^{\mathsf{D}}$ such that Double is $(\mathcal{S}^{\mathsf{D}}, \Omega^{\mathsf{D}})$-NI. Indeed, whenever both output shares $2i$ and $2i+1$ are observed, the simulator is given input share i, otherwise, no input share is needed; internal observations are exactly the inputs and are trivially simulated.

Proof of Theorem 9. Let $\Omega^{G'} = ((\mathcal{O}^{(int)^{\mathsf{R}}}_{G}, \mathcal{O}^{(int)^{\mathsf{H}}}, \mathcal{O}^{(int)^{G}}), \mathcal{O}^{(out)^{G'}})$ be a t-admissible observation set on G', and $\Omega^{F'} = ((\mathcal{O}^{(int)^{F}}, \mathcal{O}^{(int)^{\mathsf{D}}}, \mathcal{O}^{(int)^{\mathsf{R}}}_{F}), \mathcal{O}^{(out)^{F'}})$ be a t-admissible observation set on F'. $\Omega^{G} = (\mathcal{O}^{(int)^{G}}, \mathcal{O}^{(out)^{G'}})$ is t-admissible, therefore there exists a $(|\mathcal{O}^{(int)^{G}}| + |\mathcal{O}^{(out)^{G'}}|)$-compatible $\mathcal{S}^{G}$ such that G is $(\mathcal{S}^{G}, \Omega^{G})$-NI. By the property on Half, there exists a $(2(|\mathcal{O}^{(int)^{\mathsf{H}}}| + |\mathcal{S}^{G}|))$-compatible $\mathcal{S}^{\mathsf{H}}$ such that Half is $(\mathcal{S}^{\mathsf{H}}, (\mathcal{O}^{(int)^{\mathsf{H}}}, \mathcal{S}^{G}))$-NI. Now, the observation set $\Omega^{\mathsf{R}}_{G} = (\mathcal{O}^{(int)^{\mathsf{R}}}, \mathcal{S}^{\mathsf{H}})$ on the Refresh is $2t$-admissible. Therefore, there exists a $|\mathcal{O}^{(int)^{\mathsf{R}}}_{G}|$-compatible $\mathcal{S}^{G'}$ such that this instance of Refresh is $(\mathcal{S}^{G'}, \Omega^{\mathsf{R}}_{G})$-NI. It is important to note here

that $|\mathcal{S}^{G'}| \leq |\mathcal{O}^{(int)^{\mathsf{R}}}_{G}| \leq t$. We can now move on to simulating F', augmenting the observation set $\Omega^{F'}$ with the observations $\mathcal{S}^{G'}$ on its output necessary to perfectly simulate G'. The observation set $\Omega^{\mathsf{R}}_{F} = (\mathcal{O}^{(int)^{\mathsf{R}}}_{F}, \mathcal{O}^{(out)^{\mathsf{R}}}_{F} \cup \mathcal{S}^{G'})$ is $2t$-admissible. Therefore, there exists a $|\mathcal{O}^{(int)^{\mathsf{R}}}_{F}|$-compatible input projection $\mathcal{S}^{\mathsf{R}}_{F}$ such that this instance of the Refresh gadget is $(\mathcal{S}^{\mathsf{R}}_{F}, \Omega^{\mathsf{R}}_{F})$-NI. Note again that $|\mathcal{S}^{\mathsf{R}}_{F}| \leq t$, here. Considering the observation set $\Omega^{\mathsf{D}} = (\mathcal{O}^{(int)^{\mathsf{D}}}, \mathcal{S}^{\mathsf{R}})$ on the Double procedure, we deduce the existence of a $(|\mathcal{O}^{(int)^{\mathsf{D}}}| + |\mathcal{S}^{\mathsf{R}}_{F}|)$-compatible input projection $\mathcal{S}^{\mathsf{D}}$ (in fact, the bound is tighter than this, but we only need this one). Finally, we use the fact that F is t-TNI to finish constructing the simulator and bounding the size of the final input projection $\mathcal{S}$. □

4.6 Implementation and Evaluation

As a proof of concept, we implement our compiler to read and produce programs in a reasonable subset of C (including basic operators, constant for loops, table lookups at *public* indexes in public tables, and mutable secret state), equipped with libraries implementing core and extended operations for some choices of $\mathbb{K}$. The techniques and results we present in this chapter are in no way restricted to this language and could be adapted to many other settings (ASM or VHDL, for example) given a concrete target. We see such an adaptation purely as a programming language challenge, as it requires properly formalizing the semantics and side-channels of such low-level platforms. As is standard in compilation, our compiler performs several passes over the code in order to produce its final result.

4.6.1 Compiler Implementation

Parsing and Pre-Typing

This pass parses C code into our internal representation, checks that the program is within the supported subset of C, performs C type-checking and checks that variables marked as sensitive (variables given type $\mathbb{K}$) are never implicitly cast to public types. Implicit casts from public types to $\mathbb{K}$ (when compatible, for example, when casting a public `uint8_t` to a protected variable in $\mathsf{GF}(2^8)$) are replaced with public sharing gadgets.

Gadget Selection and Generic Optimizations

This pass heuristically selects optimal gadgets depending on their usage. For example, multiplication of a secret by a public value can be computed by an affine gadget that multiplies each share of the secret, whereas the multiplication of two secrets must be performed using the SecMult gadget. Further efforts in formally proving precise types for specialized core gadgets may also improve this optimization step. This pass also transforms the C code to clarify calling conventions (ensuring that arguments are passed by value when necessary), to make it follow a simpler form that makes it easier to type-check, and to optimize the use of intermediate registers.

Type-Checking and Refresh Insertion

This is the core of our compiler. We note that the verification explained in Section 4.4 fails exactly when a mask refreshing operation is needed. At the cost of tracking some more information and reinforcing the typing constraint on sub-gadgets, we use this observation to automatically insert Refresh gadgets where required. When type-checking fails, the variable whose masks need to be refreshed is duplicated and one of its uses is replaced with the refreshed duplicate. To avoid having to re-type the entire program after insertion of a refresh gadget, our compiler keeps

track of typing information for each program point already traversed and simply rewinds the typing to the program point immediately after the last modification.

The source program can also be annotated with explicit refresh operations that may help the compiler in its type-checking operations. The compiler itself can also be run in a mode that only performs type-checking and reports failures without attempting to correct the program, allowing it to be used for the direct verification of implementations clearly structured as compositions of gadgets.

Code-Generation

Finally, once all necessary mask refreshing operations have been inserted and the program has been type-checked, we produce a masked C program. This transformation is almost a one-to-one mapping from the instructions in the type-checked programs to calls to a library of verified core gadgets or to newly defined gadgets. Some cleanup is performed on loops to clarify the final code whenever possible.

4.6.2 Practical Evaluation

To test the effectiveness of our compiler, we apply it to implementations of different algorithms, generating masked implementations at various orders. We apply our compiler to the following programs:

- AES ($\odot$): a full block of AES-128 masked using the standard multiplication gadget, and implemented in $\mathsf{GF}(2^8)$ (see below for input and output Sbox algorithms);
- AES ($x \odot g(x)$): a full block of AES-128 masked using Coron *et al.*'s gadget for computing $x \odot g(x)$, and implemented in $\mathsf{GF}(2^8)$;
- Keccak: a full computation (24 rounds) of Keccak-f[1600], implemented in $\mathbb{B}^{64}$;
- Simon: a block of Simon(128,128), implemented in $\mathsf{GF}(2^{64})$;
- Speck: a block of Speck(128,128), implemented in $\mathbb{B}^{64}$.

The following codes show the input code sent to the compiler (AES S-Box) and the resulting generated code. Notice that the compiler automatically introduces the needed refresh (in function `pow254`) and uses the affine gadget `and_pub` instead of the one based on SecMult.

AES S-Box input code

```
bint8_t pow254(bint8_t x){
  bint8_t r, w, z;

  z  = gf256_pow2(x);

  r  = gf256_mul(x,z);
  w  = gf256_pow4(r);

  r  = gf256_mul(r,w);
  r  = gf256_pow16(r);
  r  = gf256_mul(r,w);
  r  = gf256_mul(r,z);
  return r;
}

bint8_t affine (bint8_t a){
  uint8_t j;
  bint8_t res = 0x00;
  bint8_t t;

  for (j = 0; j < 8; j++){
    t    = a » j;
    t   ^= a » ((j + 4)%8);
    t   ^= a » ((j + 5)%8);
    t   ^= a » ((j + 6)%8);
    t   ^= a » ((j + 7)%8);
    t   &= 0x01;
    res ^= t « j;
  }

  res = res ^ 0x63;
  return res;
}

bint8_t sbox(bint8_t a){
  bint8_t r;

  r = pow254(a);
  r = affine(r);
  return r;
}
```

Generated code

```
void pow254 (bint8_t r, bint8_t x){
  bint8_t x0;
  bint8_t w;
  bint8_t z;
  bint8_t aux;
  bint8_copy(x0, x);
  gf256_pow2(z, x0);
  bint8_refresh(aux, z);
  gf256_mul(r, x0, aux);
  gf256_pow4(w, r);
  bint8_refresh(aux, w);
  gf256_mul(r, r, aux);
  gf256_pow16(r, r);
  gf256_mul(r, r, w);
  gf256_mul(r, r, z);
  return;
}

void affine (bint8_t res, bint8_t a){
  bint8_t aux;
  bint8_t a0;
  uint8_t j;
  bint8_t t;
  bint8_copy(a0, a);
  bint8_to_base(res, 0);
  for(j = 0; j < 8; j++) {
    bint8_rshift(t, a0, j);
    bint8_rshift(aux, a0, (j+4)%8);
    bint8_xor(t, t, aux);
    bint8_rshift(aux, a0, (j+5)%8);
    bint8_xor(t, t, aux);
    bint8_rshift(aux, a0, (j+6)%8);
    bint8_xor(t, t, aux);
    bint8_rshift(aux, a0, (j+7)%8);
    bint8_xor(t, t, aux);
    bint8_and_pub(t, t, 1);
    bint8_lshift(aux, t, j);
    bint8_xor(res, res, aux);
  }
  bint8_xor_pub(res, res, 99);
  return;
}

void sbox (bint8_t r, bint8_t a){
  bint8_t a0;
  bint8_copy(a0, a);
  pow254(r, a0);
  affine(r, r);
  return;
}
```

Speck makes use of both bitwise operations (that are difficult to perform on additively shared variables) and modular addition (which is costly to perform on boolean shared variable). We choose to mask Speck using a Boolean secret-sharing scheme, implementing the algorithm in $\mathbb{B}^{64}$ and using Coron, Großschädl and Vadnala's algorithm to compute modular addition directly on boolean-shared variables [CGV14]. Since this is a defined gadget, it is compiled as part of the program and the security of its masked version is proved by typing during compilation.

Table 4.1 shows resource usage statistics for generating the masked algorithms (at any order) from unmasked implementations of each algorithm. The table shows the total number of mask refreshing operations inserted in the program[2], the compilation time, and the memory consumption. The first Keccak line refers to an implementation of Keccak where the mask refreshing

[2]Note that the number of mask refreshing operations executed during an execution of the program may be much greater, since the sub-procedure in which the insertion occurs may be called multiple times.

Table 4.1: Time taken to generate masked implementation at any order

Scheme	# Refresh	Time	Memory
AES ($\odot$)	2	0.09s	4Mo
AES ($x \odot g(x)$)	0	0.05s	4Mo
Keccak	0	121.20	456Mo
Keccak(2)	600	2728.00s	22870Mo
Simon	67	0.38s	15Mo
Speck	61	6.22s	38Mo

Table 4.2: Time taken by 10,000 executions of each program at various masking orders

Scheme	unmasked	Order 1	Order 2	Order 3	Order 5	Order 10
AES ($\odot$)	0.078s	2.697s	3.326s	4.516s	8.161s	21.318s
AES ($x \otimes g(x)$)	0.078s	2.278s	3.209s	4.368s	7.707s	17.875s
Keccak	0.238s	1.572s	3.057s	5.801s	13.505s	42.764s
Simon	0.053s	0.279s	0.526s	0.873s	1.782s	6.136s
Speck	0.022s	4.361s	10.281s	20.053s	47.389s	231.423s

operation is already inserted in the unmasked algorithm, and the compiler is run in pure verification mode. The second line refers to the compiler being run on a purely unmasked algorithm. The difficulty here comes from the large number of mask refreshing gadgets that need to be inserted, requiring the type-checker to backtrack and start over multiple times. However, first running the type-checker on a round reduced version of Keccak and identifying the problematic program point allows the user to manually insert the mask refreshing operation and simply use the compiler as a verifier to check that the resulting algorithm is indeed secure. Apart from this extreme example, which is due to the particular shape of the Keccak χ function and the way it is used in Keccak-f's round function, all compilations are rather cheap. Also note that even costly compilations only have to be performed *once* to transform an unmasked algorithm into a masked algorithm that is secure at any order.

Remark 9. *The compiler reports that the modular addition gadget used in Speck is indeed secure without inserting any refresh gadgets. This serves as a compositional and machine-checked proof of security for this algorithm at any order t.*

Table 4.2 reports the time taken to execute the resulting programs 10,000 times at various orders.[3] For AES and Speck, the figures shown in the "unmasked" column are execution times for the input to our compiler: a table-based implementation of AES or an implementation of Speck that uses machine arithmetic, rather than Coron, Großschädl and Vadnala's algorithm would be much faster, but cannot be masked directly. As an additional test to assess the performance of generated algorithms at very high order, we ran 10,000 instances of AES masked at order 100; it took less than 18 minutes to complete (so 0.108 seconds per instance).

Modifying our compiler to make use of the addition-based mask refreshing gadget shown in Algorithm 10 (and therefore produce insecure programs) yields virtually the same figures as those shown for AES, highlighting the fact that the cost of using a secure mask refreshing gadget is negligible. However, it is important to keep in mind that fresh randomness is much less costly

[3]On a Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz with 64Go of memory running Linux (Fedora)

in our setting than, say, on an embedded device, and that the use of a secure refresh gadget may cause a more important performance degradation in such a setting.

4.7 Conclusion

We have addressed the theoretical problem of secure composition in higher-order masking implementations, by introducing the notion of strong simulatability, and by showing that it supports provably correct compositional security analyses. We have displayed a general method to verify composition or to build a higher-order secure algorithm by properly positioning strong simulatable refresh gadgets. Moreover, we have observed that other gadgets from the literature are strongly simulatable, thereby reducing the needs in refresh gadgets instances, and leading to more efficient design. To exploit these new results on large circuits, we have constructed a concrete and efficient compiler, which allows us to obtain secure implementations for masking at any higher-order of several cryptographic algorithms. Finally, we have extended our compilation results to protect protocols during which the adversary is able to regularly move his probes. There are many avenues to extend our theoretical results and program transformation. Given that the masked algorithms generated by our tool are provably secure in the t-threshold probing model, and that the relationship of this model with the noisy leakage model, we expect that our algorithms will resist practical attacks, but it would also be comforting to carry out a practical security evaluation of our algorithms and validate their security experimentally.

Chapter 5

Masking and Leakage-Resilient Primitives: One, the Other(s) or Both?

In this chapter, we aim to help industrial experts to choose the best protection against power-analysis attacks. To achieve a certain security level, the community has proposed two algorithmic countermeasures: masking and the use of leakage-resilient primitives. In the following, we build a framework to compare these countermeasures in terms of efficiency and security depending on the function to protect. In particular, we realized the hardware and software simulations to evaluate the security of pseudo-random functions and generators against power-analysis attacks according to the number of measurements or the data complexity.

Contents

5.1 Introduction

5.1.1 Motivation

As mentioned all along this thesis, masking countermeasure and the use of constructions proven secure in the leakage-resilient cryptography model are frequently considered solutions to improve

security of implementations against side-channel attacks. One limitation of masking is that (as most countermeasures against DPA [MOP07]) it 'only' reduces the amount of information leakage, at the cost of sometimes strong performance overheads [GSF13] (*e.g.,* complexity in $O(t^2)$ for the t^{th}-order masked multiplications with Boolean masking). In parallel, the topic of leakage resilience has given rise to quite animated debates in the cryptographic community (see Part I, Section 2.4). The quest for models that adequately capture physical reality is still ongoing. For instance, Standaert, Pereira and Yu proposed in [SPY13] a new security notion involving simulators. The idea is to give the adversary either the real leakage or a simulated leakage independent from the secret key. If the adversary cannot distinguish between these two leakage with a significant advantage, then the algorithm must be secure against power-analysis attacks. While this new notion is a very promising step to gather practical and theoretical people, the simulator appeared to be distinguishable by Galea *et al.* [GMO+14]. Fortunately, Galea *et al.* also show how to fix the problem so that the security notion still holds. Yet, and independent of the relevance of the proofs obtained within these models, a more pragmatic problem is to find out the security levels of leakage-resilient constructions in front of standard side-channel adversaries (*i.e.,* the same as the ones considered in security evaluations for masking). That is, are these primitives useful to help cryptographic designers to pass current certification procedures (e.g. EMVco [Eur] or Common Criteria [Com])?

Unfortunately, claims in one or the other direction remained vague so far. The main reason is that, as hinted by Bernstein in a CHES 2012 rump session talk, substantiated answers require to consider both security and performances [Ber12], *i.e.,* two qualities that are generally hard to quantify.

5.1.2 Contributions

In this chapter, we aim to contribute to this issue and provide tools allowing to determine the best way to reach a given security level in different (software and hardware) scenarios, within the limits of what empirical evaluations can provide. For this purpose, we will consider the AES-based PRG and PRF illustrated in Figures 5.1 and 5.2, respectively.

For every key k_i and set of N public plaintexts $\{p_j^{i+1}\}_{1 \leqslant j \leqslant N}$, the PRG produces N outputs: a new secret key k_{i+1} and $N-1$ pseudo-random strings $\{y_j^{i+1}\}_{1 \leqslant j \leqslant N-1}$. Basically, each output is obtained by encrypting one plaintext p_j^i with the secret key k_i. Figure 5.1 illustrates the procedure with $N = 2$ (left) and $N = 256$ (right).

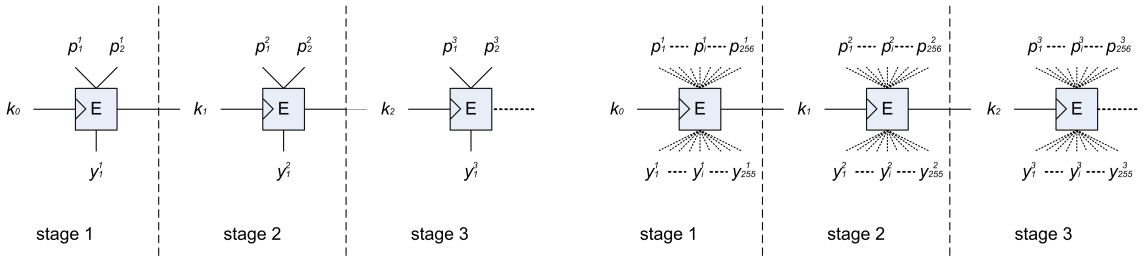


Figure 5.1: Stateful leakage-resilient PRG with $N = 2$ (left) and $N = 256$ (right).

Concerning the PRF, we use the tree-based construction from Goldreich, Goldwasser and Micali [GGM84]. The number of stages and encryptions is fixed by the ratio between the output bit size 128 and $\log_2(N)$. At each stage i, the uniformly distributed public value p_j^i, where j is given by the value of the i^{th} $\log_2(N)$ bits of the plaintext x, is encrypted using the current key

k_{i-1}. The encryption result is then used as a key for the next stage, except for the last stage where it is the PRF output y. For large values of N, an attacker could mount a DPA on this last stage, with the knowledge of the PRF output, by asking for the encryption of N different public values using the last key. In order to counteract such an attack, the last stage can be completed by a whitening step as proposed by Medwed, Standaert and Joux in [MSJ12].

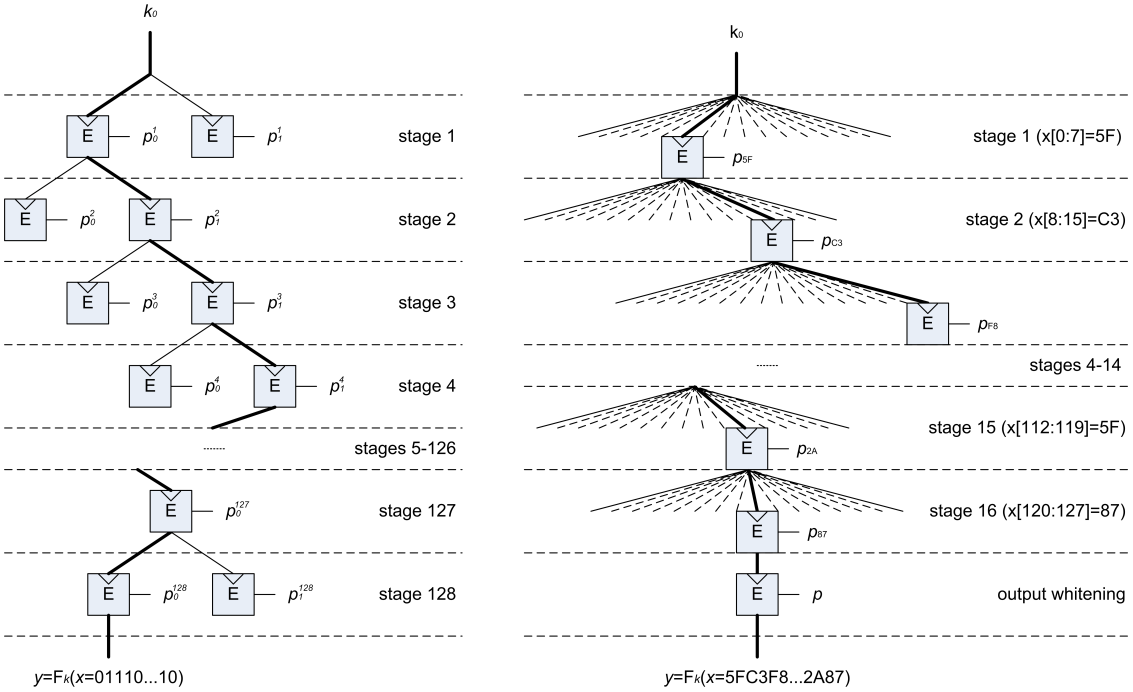


Figure 5.2: Stateless leakage-resilient PRF with $N = 2$ (left) and $N = 256$ (right).

Quite naturally, there is a simple efficiency versus security tradeoff for both types of constructions. In terms of efficiency, we can first evaluate the number of output bits per AES encryption for both primitives. In the first (PRG) case, we produce $(N - 1)$ 128-bit outputs every N AES encryptions, that is, a 128-bit output every $\frac{N}{N-1}$ AES encryptions. In the second (PRF) case, we produce a 128-bit output every $\frac{128}{\log(N)}$ AES encryptions (+1 if output whitening is used). In terms of security, the important feature in our discussions is that the PRG construction is stateful (*i.e.,* maintains a state – here the key k_i – in memory between consecutive invocations) while the PRF one is stateless (*i.e.,* produces the output y only based on the current input x). As a result, the PRG limits the *number of measurements* that a side-channel adversary can perform with the same key, since each key k_i is only going to be observed once as long as no cycles occur. By contrast, the PRF only limits his *data complexity* (*i.e.,* the number of plaintexts that can be observed). In practice, it means that in this latter case, the same measurement can be repeated multiple times, e.g. in order to get rid of the physical noise through averaging. As already discussed by Medwed *et al.* in [MSJ12], Section 3, this may lead to significant difference in terms of security against DPA.

In order to compare and combine these two primitives with masking, we investigate whether they can lead to *security-bounded implementations*, *i.e.,* implementations such that the time complexity of the best side-channel attack remains bounded independent of the number of measurements performed by the adversary. Doing so, we first show that the stateful leakage-resilient PRG in Figure 5.1 naturally leads to such implementations since it limits the number

of measurements by construction. By contrast, this guarantee is harder to reach with (stateless) leakage-resilient PRFs such as in Figure 5.2 since the attacker can still replay the same measurements until she recovers noiseless information. The tweaked construction proposed in [MSJ12] (that takes advantage of hardware parallelism) is in fact the only security-bounded PRF we found in our experiments. Next, we put forward that better security at lower cost is obtained by using the leakage-resilient PRG alone (*i.e.,* without masking), while masking alone is the most efficient solution for improving the security of stateless primitives whenever the implementations cannot be security-bounded. Therefore, our results underline that both masking and leakage-resilient primitives can be useful ingredients in the design of physically secure designs. But they also lead to the counterintuitive observation that sometimes (in fact, frequently), these solutions are better used separately, hence contradicting the usual intuition that security against side-channel attacks is best obtained via a combination of countermeasures.

Admittedly, these results are only obtained for a set of side-channel attacks that are currently representative of the state-of-the-art and we do not provide formal security proofs. In the same lines, the differences between leakage-resilient PRGs and PRFs do not contradict their own proofs: they only indicate that the (crucial) assumption of bounded leakage can imply different challenges for hardware designers. Hence, instantiating these primitives with the same AES implementation can lead to different security levels (even if the same N value is used in both cases).

5.1.3 Outline

The main goal of this chapter is to provide sound techniques to evaluate how leakage-resilient PRGs/PRFs and masking combine. In the next section, we provide a brief description of the methodology we will use for this purpose, and underline its limitations. The two main components, namely performance and security evaluations, are detailed in Sections 5.3 and 5.4, and then combined in Section 5.5.

5.2 Methodology and Limitations

Our proposal essentially holds in five steps that we detail below:

1. *Fix the target security level.* In the following, we will take the AES Rijndael with 128-bit key as case study. Since a small security degradation due to side-channel attacks is unavoidable, we will consider 120-bit, 100-bit and 80-bit target security levels for illustration. We do not go below 80-bit keys since it typically corresponds to current short-term security levels [Cry].

2. *Choose an implementation.* Given a cryptographic algorithm, this essentially corresponds to the selection of a technology and possibly a set of countermeasures to incorporate in the designs to evaluate. In the following, we will consider both software and hardware implementations for illustration, since they lead to significantly different performance and security levels. As for countermeasures, different types of masking schemes will be considered.

3. *Evaluate performances / extract a cost function.* Given an implementation, different metrics can be selected for this purpose (such as code size, RAM, or cycle count in software and area, frequency, throughput or power consumption in hardware). Both for software and hardware implementations, we will use combined functions, namely the “code size × cycle count" product and the “area / throughput" ratio. While our methodology would be

perfectly applicable to other choices of metrics, we believe they are an interesting starting point to capture the efficiency of our different implementations. In particular for the hardware cases, such metrics are less dependent on the serial vs. parallel nature of the target architectures.

4. *Evaluate security / extract the maximum number of measurements.* This central part of our analysis first requires to select the attacks from which we will evaluate security. As mentioned in introduction, we choose to focus on power-analysis attacks. We believe that it is a first step to evaluate the security of implementations against side-channel attacks. However, as a consequence, our results will be only valid in this context and may not be extended to other side-channel attacks without a deeper analysis. We discuss such issues in Section 5.6 with electromagnetic attacks. In the following, we will thus consider the very powerful standard DPA attacks described in [MOS11] for this purpose. Furthermore, we will investigate them in the profiled setting of template attacks (*i.e.,* assuming that the adversary can build a precise model for the leakage function) [CRR03]. This choice is motivated by the goal of approaching worst-case evaluations [SMY09]. Based on these attacks, we will estimate the security graphs introduced in [VGS13] and described in Part I, Chapter 2. From a given security level (e.g. 120-bit time complexity), we will finally extract the maximum number of measurements per key tolerated, as can be bounded by the PRG construction.

5. *Compute a global cost metric (possibly with an application constraint).* In case of security-bounded implementations, the previous security evaluation can be used to estimate how frequently one has to refresh the key within a leakage-resilient construction. From this estimate, we derive the average number of AES encryptions to execute per 128-bit output. By multiplying this number with the cost function of our performance evaluations, we obtain a global metric for the implementation of an AES-based design ensuring a given security level. In case of security-unbounded implementations, re-keying is not sufficient to maintain the target security level independent of the number of measurements performed by the adversary. So the cost functions have to be combined with an application constraint, stating the maximum number of measurements that can be tolerated to maintain this security level.

Quite naturally, such a methodology is limited in the same way as any performance and security evaluation. From the performance point-of-view, our investigations only apply to a representative subset of the (large) set of AES designs published in the literature. We first paid attention to state-of-the-art implementations and countermeasures, but applying our methodology to more examples is naturally feasible (and desirable). A very similar statement holds for security evaluations. Namely, we considered standard DPA attacks as a starting point, and because they typically correspond to the state-of-the-art in research and evaluation laboratories. Yet, cryptanalytic progresses can always appear. For example, the algebraic side-channel attacks introduced in [RS09, RSV09], while somewhat unrealistic for now, would certainly lead to different security levels. Besides, countermeasures such as masking may rely on physical assumptions that are difficult to compare rigorously (since highly technology-dependent), as will be detailed next with the case of glitches.

Note that these limitations are to a large extent inherent to the problem we tackle, and our results also correspond to the best we can hope in this respect. Hence, more than the practical conclusions that we draw in the following sections (that are of course important for current engineers willing to implement physically secure designs), it is the fact that we are able to compare the performance vs. security tradeoffs corresponding to the combination of

leakage-resilient constructions with masking that is the most important contribution of this work. Indeed, these comparisons are dependent on the state-of-the-art implementations and attacks that are considered to be relevant for the selected algorithm.

5.3 Performance Evaluations

In this section, we provide our performance evaluations for unprotected and masked AES designs. As previously mentioned, we will consider both software and hardware examples for this purpose. In this context, the main challenge is to find implementations that are (reasonably) comparable. This turned out to be relatively easy in the software case, for which we selected a couple of implementations in 8-bit microcontrollers, *i.e.,* typical targets for side-channel analysis. By contrast, finding implementations in the same technology turns out to be more challenging in hardware: transistor sizes have evolved from (more than) 130μm to (less than) 65ηm over the last 15 years (*i.e.,* the period over which most countermeasures against side-channel attacks have been proposed). Hence, published performance evaluations for side-channel protected designs are rarely comparable. Yet, we could find several designs in a recent FPGA technology, namely the Xilinx Virtex-5 devices (that are based on a 65ηm process).

The performances of the implementations we will analyze are summarized in Table 5.1. As previously mentioned, our software cost function is the frequently considered "code size $\times$ cycle count" metric, while we use the "area / throughput" ratio in the hardware (FPGA) case. As for the countermeasures evaluated, we first focused on the higher-order masking scheme pro-

Table 5.1: Performance of some illustrative AES implementations.

Software (8-bit) Implementations	*code size* (*bytes*)	*cycle count*	*cost function*	*physical assumptions*
Unprotected [EGG$^+$12]	1659	4557	7.560	-
1-mask Boolean [RP10]	3153	$129 \cdot 10^3$	406.7	glitch-sensitive
1-mask polynomial [GSF13, PR11]	20 682	$1064 \cdot 10^3$	22 000	glitch-resistant
2-mask Boolean [RP10]	3845	$271 \cdot 10^3$	1042	glitch-sensitive
FPGA (Virtex-5) Implementations	*area* (*slices*)	*throughput* (*enc/sec*)	*cost function*	*physical assumptions*
Unprotected (128-bit) [RYS]	478	$\frac{245 \cdot 10^6}{11}$	21.46	-
1-mask Boolean (128-bit) [RYS]	1462	$\frac{100 \cdot 10^6}{11}$	160.8	glitch-sensitive
Threshold (8-bit) [MPL$^+$11]	958	$\frac{170 \cdot 10^6}{266}$	1499	glitch-resistant

posed by Rivain and Prouff at CHES 2010, which can be considered as the state-of-the-art in software [RP10][1]. We then added the CHES 2011 polynomial masking scheme of Prouff and Roche [PR11] (and its implementation in [GSF13]), as a typical example of "glitch-resistant" solution relying on secret sharing and multiparty computation (see the discussion in the next paragraph). A similar variety of countermeasures is proposed in hardware, where we also consider an efficient but glitch-sensitive implementation proposed in [RYS], and a threshold AES implementation that is one of the most promising solutions to deal with glitches in this case [MPL$^+$11]. Note that this latter implementation is based on an 8-bit architecture (rather than a 128-bit one for the others). So although our cost function is aimed at making comparisons between different

[1] A higher-order attack against the S-box [RP10] has been exhibited in [CPRR14] but is still hard to mount in practice.

architectures more reflective of the algorithms' and countermeasures' performances, more serial implementations as this one generally pay a small overhead due to their more complex control logic.

Physical assumptions and glitches. As explicited in Table 5.1, countermeasures against side-channel attacks always rely on a number of physical assumptions. In the case of masking, a central one is that the leakage of the shares manipulated by the target implementation should be independent of each other [ISW03]. Glitches, that are transient signals appearing during the computations in certain (e.g. CMOS) implementations, are a typical physical default that can cause this assumption to fail, as first put forward by Mangard *et al.* in [MPG05]. There are two possible solutions to deal with such physical defaults: either by making explicit to cryptographic engineers that they have to prevent glitches at the physical level, or by designing countermeasures that can cope with glitches.

Interestingly, the first solution is one aspect where hardware and software implementations significantly differ. Namely, while it is usually possible to ensure independent leakages in masked software, by ensuring a sufficient time separation between the manipulation of the shares, it is extremely difficult to avoid glitches in hardware [MPO05]. Yet, even in hardware it is generally expected that the glitch signal will be more difficult to exploit by adversaries, especially if designers pay attention to this issue [MM12]. In this context, the main question is to determine the amplitude of this signal: if sufficiently reduced in front of the measurement noise, it may turn out that a glitch-sensitive masked implementation leads to improved security levels (compared to an unprotected one). Since this amplitude is highly technology-dependent, we will use it as a parameter to analyze the security of our hardware implementations in the next sections. Yet, we recall that it is a safe practice to focus on glitch-resistant implementations when it comes to hardware.

5.4 Security Evaluations

We now move to the core of our analysis, namely the security evaluation of different implementations. For this purpose, we first need to discuss the type of security evaluation we will conduct, which can be viewed as a tradeoff between generality and informativeness. That is, one ideally wants to reach general conclusions in the sense that they are independent of the underlying device technology. A typical solution for this purpose is to evaluate the "security order" of a countermeasure, as defined by Coron *et al.* [CPR07]. Informally, the security order corresponds to the largest moment in the leakage probability distributions that is key-independent (hence from which no information can be extracted). For example, an unprotected implementation can be attacked by computing mean values (*i.e.,* first-order moments) [KJJ99]. By contrast, the hope of masking is to ensure that adversaries will have to estimate higher-order moments, which is expected to increase the data complexity required to extract information, as first shown by Chari *et al.* [CJRR99]. Evaluating the order is interesting because under the independent leakage assumption mentioned in the last section, it can be done based on the mathematical description of a countermeasure only. Of course, the informativeness of such an abstract evaluation is limited since (1) it indeed does not allow testing whether the independent leakage assumption is fulfilled, and (2) even if this assumption is fulfilled, there is no strict correspondance between the security order and the security level of an implementation (e.g. measured with a probability of success corresponding to some bounded complexities). This is because already for masking (*i.e.,* the countermeasure that aims at increasing the security order), and even if independent leakages are observed in practice, the actual complexity of a side-channel attack highly depends

on the amount of noise in the measurements. And of course, there are also countermeasures that simply do not aim at increasing the security order, e.g. shuffling [HOM06].

One appealing way to mitigate the second issue is to perform so-called "simulated attacks". This essentially requires to model the leakage corresponding to different sensitive operations in an idealized implementation. For example, a usual approximation is to consider that all the intermediate values during a cryptographic computation (such as the S-boxes inputs and outputs for a block cipher) leak the sum of their Hamming weight and a Gaussian distributed noise [MOP07]. It is then possible to accurately estimate the evaluation metrics proposed in [SMY09] (see Part I, Chapter 2), *i.e.,* mutual information, success rate, guessing entropy, from these mathematically generated leakages. Furthermore, one can use the noise variance as a security parameter and analyze its impact on the time and data complexity of successful attacks. Quite naturally, such an alternative still does not solve the first issue (*i.e.,* the independent leakage assumption), for which the only possibility is to evaluate the real measurements of an actual implementation, in a given technology. This latter solution is admittedly the most informative, but also the least general, and is quite intensive for comparison purposes (since it requires to have access to source codes, target devices and measurement setups for all the designs to evaluate). Interestingly, it has been shown that simulated attacks can be quite close to real ones in the context of standard DPA and masking [SVO$^+$10]. So since our goal is to show that there exist realistic scenarios where leakage-resilient PRGs/PRFs and masking are useful ingredients to reach a given security level at the lowest cost, we will use this type of evaluations in the following.

Note finally that computing explicit formulae for the success rate according to the level of noise and the number of measurements (see *e.g.,* [Riv09, FLD12, LPR$^+$14]) can not replace the simulation of attacks. Indeed, these formulae only predict subkey (typically key bytes) recoveries while we consider security graphs for full 128-bit master keys. Besides, they are only applicable to unprotected devices so far, and hardly capture masked implementations and the effect of key-dependent algorithmic noise as we will consider next.

5.4.1 Evaluation Setups

We will consider two types of setups in our evaluations: one for software and one for hardware. As illustrated in Figure 5.3 in the case of a Boolean-masked S-box implementation with two shares, the main difference is that the software performs all the operations sequentially, while the hardware performs them in parallel. We will further assume that the leakage of parallel operations is summed [PSDQ05]. As previously mentioned, we will illustrate our analyses with a Hamming weight leakage function. Additionally, we will consider a noise variance of 10, corresponding to a signal-to-noise ratio of 0.2 (as defined in Part I, Section 2.2.2). This is a typical value, both for software implementations [DRS$^+$12] and FPGA measurement boards [KSK$^+$10].

Let us denote the AES S-box as S, a byte of plaintext and key as x_i and k_i (respectively), the random shares used in masking as r_i^j (before the S-box) and m_i^j (after the S-box), the Hamming weight function as HW, the bitwise XOR as $\oplus$, the field multiplication used in polynomial masking as $\otimes$, and Gaussian-distributed noise random variables N_i^j. From these notations, we can specify the list of all our target implementations as summarized in Tables 5.2 and 5.3 with *plain* either the known plaintexts or the chosen plaintexts construction.

A couple of observations are worth being underlined as we now discuss.

First, and as already mentioned, the main difference between software and hardware implementations is the number of exploitable leakage samples: there is a single such sample per

Table 5.2: List of our target software implementations.

Ref.: 8-bit software	*leakage function* ($\forall\ i \in [1;16]$)	*glitches*	*plain*
KSU: Unprotected [EGG+12]	$L_i = \mathsf{HW}(\mathsf{S}(x_i \oplus k_i)) + N_i$	no	KP
KSB_1: 1-mask Boolean [RP10]	$L_i^1 = \mathsf{HW}(\mathsf{S}(x_i \oplus k_i) \oplus M_i) + N_i^1$ $L_i^2 = \mathsf{HW}(M_i) + N_i^2$	no	KP
KSP_1: 1-mask polynomial [GSF13, PR11]	$L_i^1 = \mathsf{HW}(\mathsf{S}(x_i \oplus k_i) \oplus M_i \otimes P_0) + N_i^1$, $L_i^2 = \mathsf{HW}(\mathsf{S}(x_i \oplus k_i) \oplus M_i \otimes P_1) + N_i^2$	no	KP
KSB_2: 2-mask Boolean [RP10]	$L_i^1 = \mathsf{HW}(\mathsf{S}(x_i \oplus k_i) \oplus M_i^1 \oplus M_i^2) + N_i^1$, $L_i^2 = \mathsf{HW}(M_i^1) + N_i^2$, $L_i^3 = \mathsf{HW}(M_i^2) + N_i^3$	no	KP

Table 5.3: List of our target hardware implementations.

Virtex-5 FPGA	*leakage function* (sum over $1 \leq i \leq 16$)	*glitches*	*plain*
KHU: Unprotected (128-bit) [RYS]	$L = \sum[\mathsf{HW}(\mathsf{S}(x_i \oplus k_i))] + N$	no	KP
CHU: Unprotected (128-bit) [RYS]	$L = \sum[\mathsf{HW}(\mathsf{S}(x \oplus k_i))] + N$	no	CP
KHB_1: 1-mask Boolean (128-bit) [RYS]	$L = \sum[\mathsf{HW}(\mathsf{S}(x_i \oplus k_i) \oplus M_i) + \mathsf{HW}(M_i)] + N$	no	KP
KHB_1^*: 1-mask Boolean (128-bit) [RYS]	$L = \sum[\frac{\mathsf{HW}(\mathsf{S}(x_i \oplus k_i))}{f} + \mathsf{HW}(M_i)] + N$	1^{st}-order	KP
CHB_1^*: 1-mask Boolean (128-bit) [RYS]	$L = \sum[\frac{\mathsf{HW}(\mathsf{S}(x \oplus k_i))}{f} + \mathsf{HW}(M_i)] + N$	1^{st}-order	CP
KHT_2: Threshold (8-bit) [MPL+11]	$L = \sum[\mathsf{HW}(\mathsf{S}(x_i \oplus k_i) \oplus M_i^1 \oplus M_i^2)$ $+\mathsf{HW}(M_i^1) + \mathsf{HW}(M_i^2)] + N$	no	KP
KHT_2^*: Threshold (8-bit) [MPL+11]	$L = \sum[\frac{\mathsf{HW}(\mathsf{S}(x_i \oplus k_i) \oplus M_i^1) + \mathsf{HW}(M_i^1)}{f} + \mathsf{HW}(M_i^2)]$ $+N$	2^{nd}-order	KP

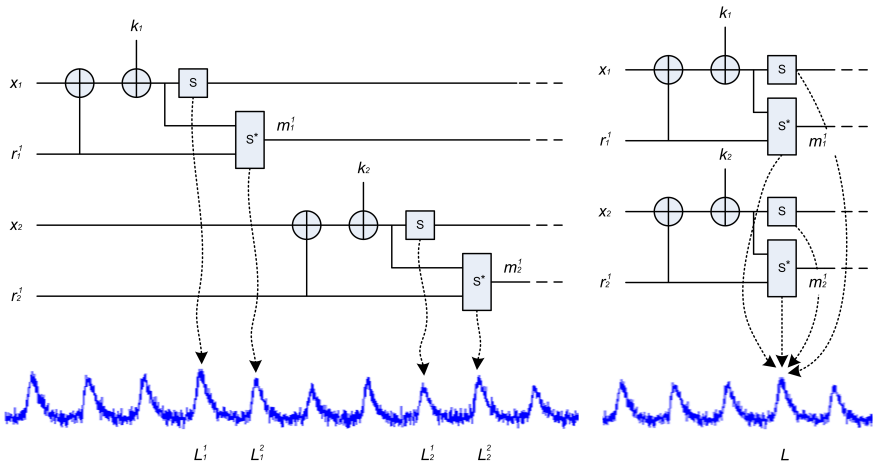


Figure 5.3: Simulated leaking implementations. Left: software, right: hardware.

plaintext in hardware while there are $16 \times (N_m + 1)$ ones in software (with N_m the number of masks). Next, we only considered glitches in hardware (since it is generally possible to ensure independent leakage in software, by ensuring a sufficient time separation between the manipulation of the shares). We assumed that "first-order glitches" can appear in our Boolean-masked FPGA implementation, and modeled the impact of the mask as an additive binomial noise in this case. We further assumed that the amplitude of this first-order signal was reduced according to a factor f. This factor corresponds to the parameter used to quantify the amplitude of the glitches mentioned in the previous section. Note that this modeling is sound because the complexity of a first-order DPA only depends on the value of its SNR (which is equivalent to correlation and information theoretic metrics in this case, as proven in [MOS11]). So even leakage functions deviating from the Hamming weight abstraction would lead to similar trends. Since the threshold implementation in [MPL+11] guarantees the absence of first-order glitches, we only analyzed the possibility of second-order glitches for this one, and modeled them in the same way as just described (*i.e.,* by considering the second mask M_i^2 as an additive binomial noise, and reducing the amplitude of the second-order signal by a factor f). Third, the chosen-plaintext construction of [MSJ12] is only applicable in hardware. We only evaluated its impact for the unprotected implementation, and the 1-mask Boolean one with glitches. As will become clear in the next section, this is because the data complexity bound to 256 (that is the maximum tolerated by design in this case) is only relevant when successful side-channel attacks occur for such small complexities (which was only observed for implementations with first-order signal).

For convenience, we denoted each implementation in our experiments with three letters. The first one corresponds to the type of scenario considered, *i.e.,* with Known (K) or carefully Chosen (C) plaintexts. The second one indicates whether we are in a Software (S) or Hardware (H) case study. The third one corresponds to the type of countermeasure selected, *i.e.,* Unprotected (U), 1- or 2-mask Boolean (B_1, B_2), 1-mask Polynomial (P_1) and 2-mask threshold (T_2). The additional star signals finally reflect the presence of (first-order or second-order) glitches. For example, KHB_1^* is an AES design protected with a 1-mask Boolean scheme, implemented in an imperfect hardware leading to first-order glitches, and analyzed in the context of known (uniform) plaintexts.

5.4.2 Template Attacks and Security Graphs

Given the leakage functions defined in Tables 5.2 and 5.3 and the description provided in Part I Section 2.2, in the following, and for each byte of the AES master key, we will consider Gaussian templates for unprotected implementations $(\mu_{(k_i,x_i)}, \sigma_{(k_i,x_i)})$, and Gaussian mixtures for masked implementations $(\boldsymbol{\mu}_{(k_i,x_i)}, \Sigma_{(k_i,x_i)})$. This leads to models of the form:

$$\mathbb{P}_{\text{model}}[k_i|l_i, x_i] = \frac{\phi_{\mu_{(k_i,x_i)},\sigma_{(k_i,x_i)}}(l_i)}{\sum_{k_i^* \in \mathcal{K}} \phi_{\mu_{(k_i^*,x_i)},\sigma_{(k_i^*,x_i)}}(l_i))}, \tag{5.1}$$

$$\mathbb{P}_{\text{model}}[k_i|l, x_i] = \frac{\phi_{\mu_{(k_i,x_i)},\sigma_{(k_i,x_i)}}(l)}{\sum_{k_i^* \in \mathcal{K}} \phi_{\mu_{(k_i^*,x_i)},\sigma_{(k_i^*,x_i)}}(l))}, \tag{5.2}$$

for (software and hardware) unprotected implementations and:

$$\mathbb{P}_{\text{model}}[k_i|l_i^1, l_i^2, x_i] = \frac{\sum_{m_i^* \in M} \phi_{\boldsymbol{\mu}_{(k_i,x_i,m_i^*)},\Sigma_{(k_i,x_i,m_i^*)}}(l_i^1, l_i^2)}{\sum_{k_i^* \in \mathcal{K}} \sum_{m_i^* \in M} \phi_{\boldsymbol{\mu}_{(k_i^*,x_i,m_i^*)},\Sigma_{(k_i^*,x_i,m_i^*)}}(l_i^1, l_i^2)}, \tag{5.3}$$

$$\mathbb{P}_{\text{model}}[k_i|l, x_i] = \frac{\sum_{m_i^* \in M} \phi_{\mu_{(k_i,x_i,m_i^*)},\sigma_{(k_i,x_i,m_i^*)}}(l)}{\sum_{k_i^* \in \mathcal{K}} \sum_{m_i^* \in M} \phi_{\mu_{(k_i^*,x_i,m_i^*)},\sigma_{(k_i^*,x_i,m_i^*)}}(l)}, \tag{5.4}$$

for (software and hardware) masked implementations with two shares. The formula naturally extends to more shares by just adding more sums over the masks. Note that in these models, all the noise (including the algorithmic one in hardware implementations) is captured by the Gaussian distribution[2]. Given these models, the template adversary will accumulate information on the key bytes k_i, by computing products of probabilities corresponding to multiple plaintexts. Doing so and for each key byte, she will produce lists of 256 probabilities corresponding each possible candidate $\tilde{k_i}$, defined as follows:

$$p_{\tilde{k_i}} = \prod_{j=1}^{q} \mathbb{P}_{\text{model}}[\tilde{k_i}|\mathbf{L}^{(j)}, x_i^{(j)}], \tag{5.5}$$

with the leakage vector $\mathbf{L}^{(j)}$ respectively corresponding to $l_i^{(j)}$ (resp. $l^{(j)}$) in the context of Equ. 5.1 (resp. Equ. 5.2) and $l_i^{1,(j)}, l_i^{2,(j)}$ (resp. $l^{(j)}$) in the context of Equ. 5.3 (resp. Equ. 5.4). The number of measurements is given by q in Equ. 5.5. Next and for each target implementation, we will repeat 100 experiments. For each value of q in these experiments, use the rank estimation algorithm provided in [VGS13] to evaluate the time complexity needed to recover the full AES master key. Eventually, we will build security graphs, as described in Part I Section 2.2, where the attack probability of success is provided in function of a time complexity and a number of measurements.

Iterative DPA against constructions with carefully chosen plaintexts. Note that while standard DPA attacks are adequate to analyze the security of unprotected and masked implementations in a known-plaintext scenario, their divide-and-conquer strategy hardly applies to the PRF in [MSJ12], with carefully-chosen plaintexts leading to key-dependent algorithmic noise. This is because the (maximum 256) constants c_j used in this proposal are such that all 16 bytes

[2]While algorithmic noise is generated with a binomial distribution in our experiments (as mentioned in the previous subsections), it is closely approximated by a normal one, since combined with enough (simulated) physical noise that is Gaussian.

are always identical. Hence, a standard DPA will provide a single list of probabilities, containing information about the 16 AES key bytes at once. In this case, we additionally considered the iterative DPA described in this previous reference, which essentially works by successively removing the algorithmic noise generated by the best-rated key bytes. While such an attack can only work under the assumption that the adversary has a very precise leakage model in hand, we use it as a representative of worst-case attack against such a construction.

5.4.3 Experimental Results

The security graphs of all the implementations listed in Tables 5.2 and 5.3, have been similarly evaluated and are given in Figures 5.4 to 5.9, where we additionally provide the maximum number of measurements tolerated to maintain security levels corresponding to 2^{120}, 2^{100} and 2^{80} time complexity. The black curve represents the minimal observed rank for the key, the white one the maximal observed rank and the red one the mean rank.

The first four figures 5.4 and 5.5 represent the security graphs for known plaintexts in software scenario with or without masking. While the unprotected implementation is very sensitive to side-channel attacks (*i.e.,* security below 2^{20} with 70 measurements), the masked implementations significantly raise the security levels. The most secure implementation is achieved with polynomial masking where the attacker must acquire more than 150,000 measurements to decrease the time complexity below 2^{80}.

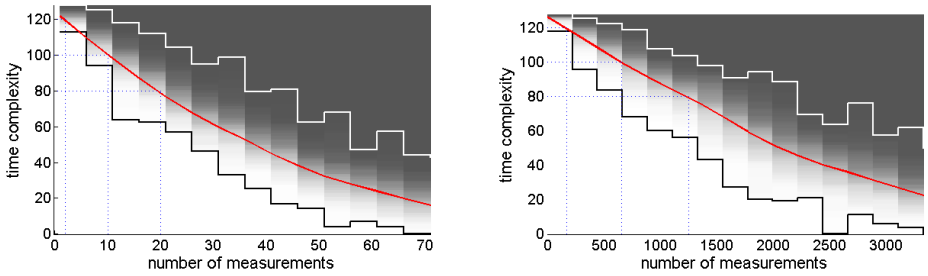


Figure 5.4: DPA-based security graphs for KSU (left) and KSB_1 (right).

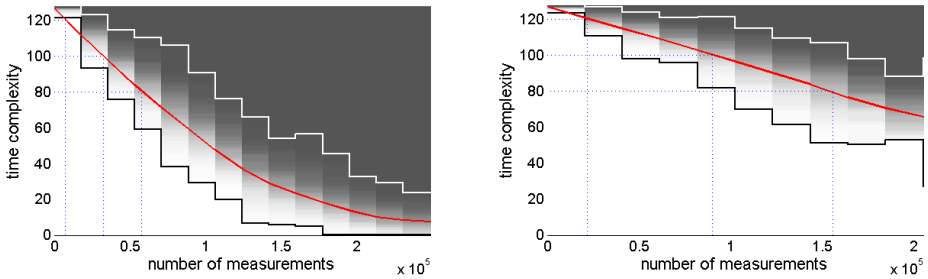


Figure 5.5: DPA-based security graphs for KSB_2 (left) and KSP_1 (right).

The seven figures from 5.6 to 5.9 describe the security evolution in case of hardware implementations in different scenarios. Not surprisingly, the unprotected hardware case provide higher security than in the software case since the Sboxes are performed in parallel. The chosen-plaintext scenario provides better security (time complexity still above 2^{80} after 250 measure-

ments) but is limited in data complexity. As we can see, the different masked implementations improve the security level, and particularly the threshold countermeasures.

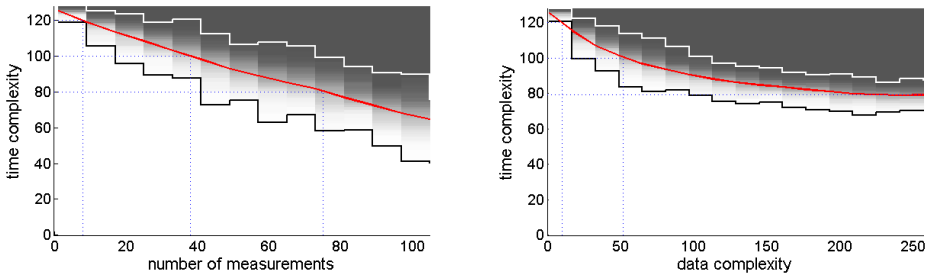


Figure 5.6: DPA-based security graphs for KHU (left) and CHU (right).

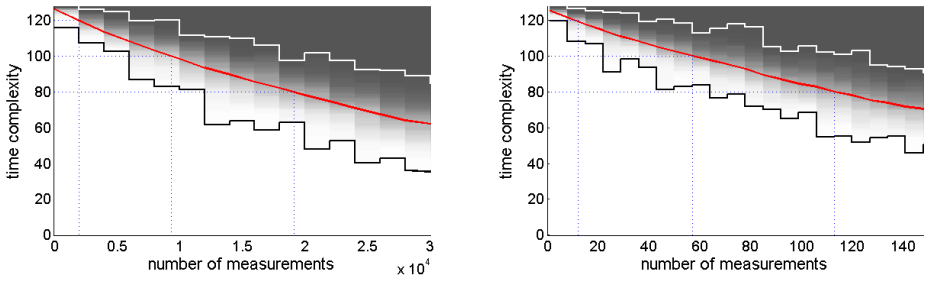


Figure 5.7: DPA-based security graphs for KHB_1 (left) and $\mathrm{KHB}_1^*/f = 1$ (right).

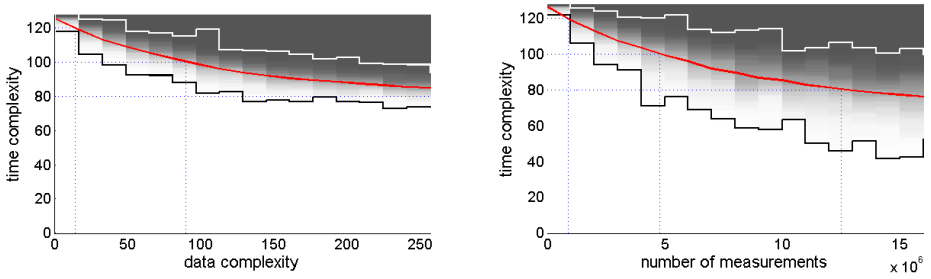


Figure 5.8: DPA-based security graphs for CHB_1^*/f=1 (left) and KHT_2 (right).

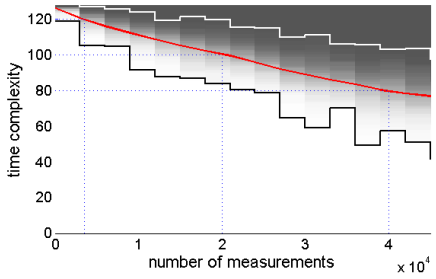


Figure 5.9: DPA-based security graphs for $\text{KHT}_2^*/f{=}1$.

In the aforementioned case of iterative DPA (Figure 5.10), the adversary recovers the AES key bytes but still has to find their position within the AES state, which (roughly) corresponds to $16! \approx 2^{44}$ possibilities [BDH+13].

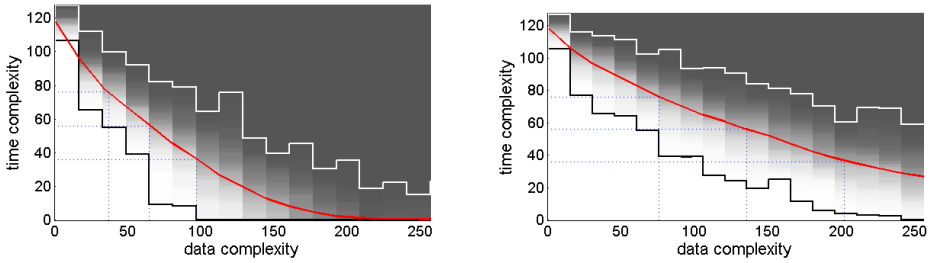


Figure 5.10: Iterative DPA-based security graphs for CHU (left) and $\text{CHB}_1^*/f = 1$ (right).

5.5 Security against Performance Tradeoffs

We now combine the results in the previous sections to answer our main question. Namely, what is the best way to exploit masking and/or leakage-resilient primitives to resist standard DPA in hardware and software implementations?

5.5.1 Leakage-resilient PRGs

Let M be the maximum number of measurements tolerated to maintain a given security level for one of the implementations in section 5.4. The re-keying in leakage-resilient PRGs is such that it is exactly this number M that is limited by design (*i.e.,* the value N in Figure 5.1 bounds M for the adversary), hence directly leading to security-bounded implementations. The global cost metric we use in this case can be written as $\frac{M}{M-1} \times$ *cost function*, where the first factor corresponds to the average number of AES encryptions that are used to produce each 128-bit output string, and the second one is the cost function of Table 5.1.

A comparison of different leakage-resilient PRG implementations in software (*i.e.,* based on different unprotected and protected AES implementations) is given in Figure 5.11 for 80-bit, 100-bit and 120-bit security levels with the corresponding global cost metrics. The main observation in this context is that the straightforward implementation of the PRG with an unprotected AES design is the most efficient solution (its global cost metric is the lower one in the three security scenarios). This is mainly because moving from the smallest M value (*i.e.,* $M = 2$, as imposed

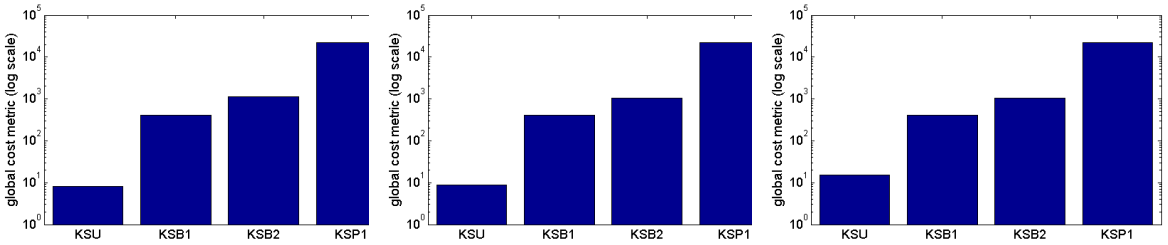


Figure 5.11: LR-PRGs in software. 80-bit (left), 100-bit (middle) and 120-bit (right) security.

by the 120-bit security level in the unprotected case - see Figure 5.4-left) to large ones (e.g. $M > 1000$ for masked implementations) can only lead to a gain factor of 2 for the global cost metric, which is not justified in view of the performance overheads due to the masking. For a similar reason (*i.e.,* the limited interest of increasing M), the global cost metric is essentially independent of the target security level in the figure. In other words, there is little interest in decreasing this security level since it leads to poor performance improvements. The hardware implementations in Figure 5.12 lead to essentially similar intuitions as also witnessed by the limited impact of decreasing the amplitude of the glitch signal with the f factor (see the KHB_1^* and KHT_2^* implementations for which $f = 10$ in the latter figures)

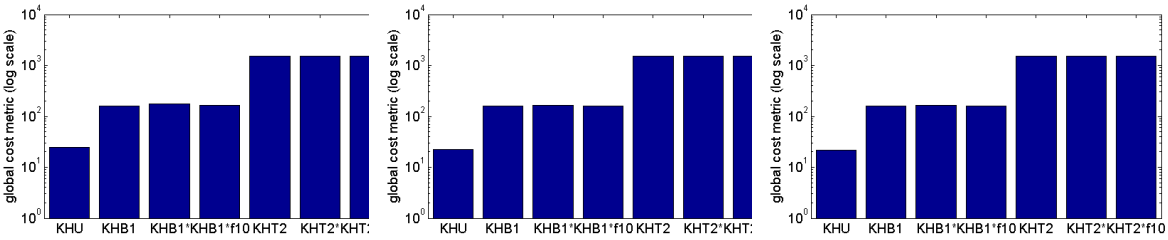


Figure 5.12: LR-PRGs in hardware. 80-bit (left), 100-bit (middle) and 120-bit (right) security.

5.5.2 Leakage-Resilient PRFs

Security-unbounded implementations.

Let us now consider (stateless) leakage-resilient PRFs. As already mentioned, those constructions only bound the adversary's data complexity. The main observation in this case is that if random plaintexts are considered, such implementations can only be security-unbounded (with the slight cautionary note that we give below). This fact can be easily explained when the PRF is instantiated with an unprotected software implementation of the AES. What happens then is that the adversary can repeat his measurements to get rid of the physical noise, and consequently move from the security graph of Figure 5.13-left to the one of Figure 5.13-right. Such a "repeating" attack is exactly the one already mentioned in [MSJ12] to argue that bounded data complexity is not enough to bound computational security. In fact, it similarly applies to masked implementations. The only difference is that the adversary will not average his measurements, but rather combine them as in Equation 5.5. This is because given a leakage function, e.g. the Hamming weight one that leads to 9 distinguishable events, the distribution of the measurements in a masked implementation will lead to the same number of distinguishable events: the only difference is that more sampling will be necessary to distinguish them (see the

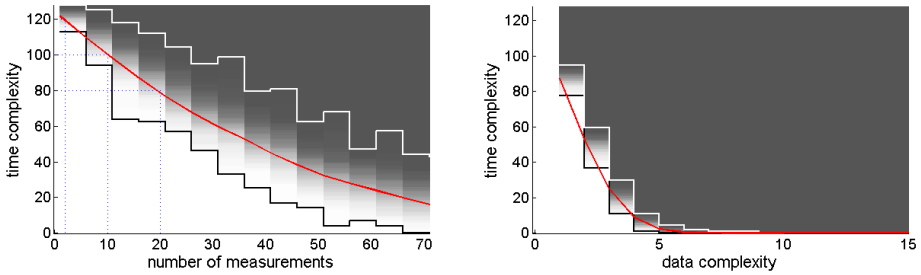


Figure 5.13: DPA-based security graphs for KSU (left) and repeating attack (right).

appendices in [SVO+10] for a plot of these distributions). So, if the number of measurements is not bounded, attacks with low time complexities, as in Figure 5.4 right, will always exist.

One important consequence is that using the PRF construction in this context is essentially useless for all the AES implementations we consider here. The only way to maintain a target security level for such stateless primitives is to limit the number of measurements by putting a constraint on the lifetime of the system. And this lifetime will be selected according to the maximum number of measurements tolerated that can be extracted from our security graphs, which now highly depends on the countermeasure selected. In other words, we can only evaluate the cost function and the security level attained independently in this case, as illustrated in Figure 5.14 for our software instances. Here, we naturally come back to the standard result that

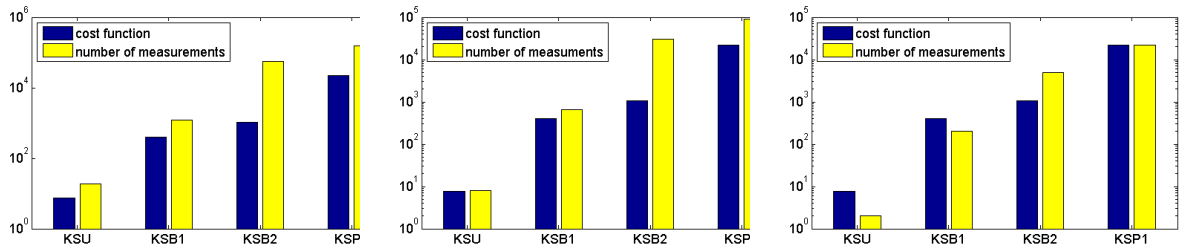


Figure 5.14: LR-PRFs in software with KP. 80-bit (left), 100-bit (middle) and 120-bit (right) security.

Boolean (resp. polynomial) masking increases security at the cost of performance overheads that are roughly quadratic (resp. cubic) in the number of shares. Note that the security level of the 1-mask polynomial scheme is higher than the 2-mask Boolean one for the noise variance we consider, which is consistent with the previous work of Roche and Prouff [RP11]. Similar conclusions are obtained with hardware implementations (Figure 5.15), for which the impact of glitches is now clearly visible. For example, a factor $f = 10$ essentially multiplies the number of measurements by f for the Boolean masking with first-order glitches, and f^2 for the threshold implementation with second-order glitches.

Cautionary note. The statement that stateless leakage-resilient PRFs can only be security unbounded if known plaintexts are considered essentially relates to the fact that repeated measurements allow removing the effect of the noise and the masks in a leaking implementation. Yet, this claim should be slightly mitigated in the case of algorithmic noise in hardware implementations. Indeed, this part of the noise can only be averaged up to the data complexity bound

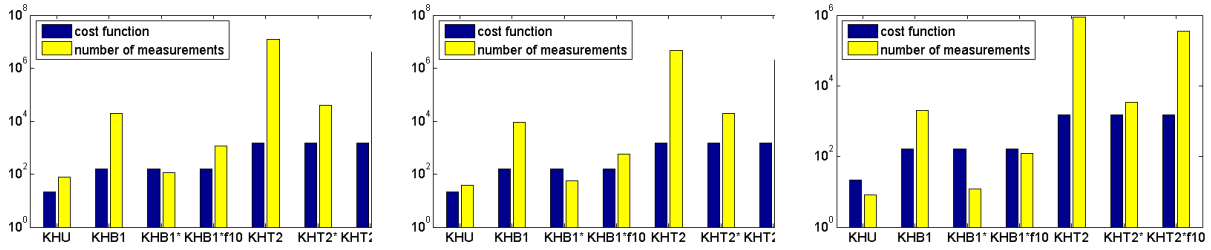


Figure 5.15: LR-PRFs in hardware with KP. 80-bit (left) and 120-bit (right) security.

that is imposed by the PRF design. Taking the example of our hardware implementations where all 16 S-boxes are manipulated in parallel, the SNR corresponding to algorithmic noise can be computed as the ratio between the variance of a uniformly distributed 8-bit values's Hamming weight (*i.e.,* 2) and the variance of 15 such values (*i.e.,* 30). Averaging this noise over M plaintexts will lead to SNRs of $\frac{1}{15/M}$, which is already larger than 17 if $M = 256$ (*i.e.,* a noise level for which the security graph will be extremely close to the worst case one of Figure 5.13-right). So although there is a "gray area" where a leakage-resilient PRF implemented in hardware can be (weakly) security-bounded, these contexts are of quite limited interest because they will imply bounds on the data complexity that are below 256, *i.e.,* they anyway lead to less efficient solutions than the tweaked construction that we investigate in the next subsection.

Security-bounded implementations. As just discussed, stateless primitives hardly lead to security bounded implementations if physical and algorithmic noise can be averaged - which is straightforwardly feasible in a known plaintext scenario. The tweaked construction in [MSJ12] aims at avoiding such a weakness by preventing the averaging of the algorithmic noise, thanks to the combined effect of hardware parallelism and carefully chosen plaintexts leading to key-dependencies in this noise. Since only the physical noise can be averaged in this case, the bounded data complexity that the leakage-resilient PRF guarantees leads to security-bounded implementations again, when thwarting power-analysis attacks. This is illustrated both by the standard DPAs (such as in Figures 5.6-right and 5.8-left) and the iterative attacks (such as in Figure 5.10) that can be performed against this PRF[3]. As in Section 5.5.1, we extracted the maximum data complexity D from these graphs, and produced as global cost metric:

$$\left\lceil \frac{128}{\lfloor \log_2(D) \rfloor} \right\rceil \times \textit{cost function},$$

where the first factor corresponds to the (rounded) average number of AES encryptions needed to produce a 128-bit output, and the second one is the cost function of Table 5.1. A comparison of our different leakage-resilient PRFs instantiated with a hardware implementation of the AES and chosen plaintexts is given in Figure 5.16. Here again, we observe that the most efficient solution is to consider an unprotected design. Interestingly, we also observe that for the unprotected AES, the iterative attack is the worst case for the 80-bit security level (where it forces the re-keying after 97 plaintexts vs. 256 for the standard DPA), while the standard DPA is the worst-case for the 120-bit security level (where it forces the re-keying after 10 plaintexts vs. 37 for the iterative attack). This nicely fits the intuition that iterative attacks become more powerful as the data complexity increases, *i.e.,* when the additional time complexity corresponding to the

[3]As previously mentioned, there is an additional $16! \approx 2^{44}$ time complexity implied in the iterative DPA attacks, corresponding to the enumeration of a permutation over the 16 AES key bytes that is necessary to test each key candidate.

enumeration of a permutation over 16 bytes becomes small compared to the time complexity required to recover the 16 AES key bytes (unordered).

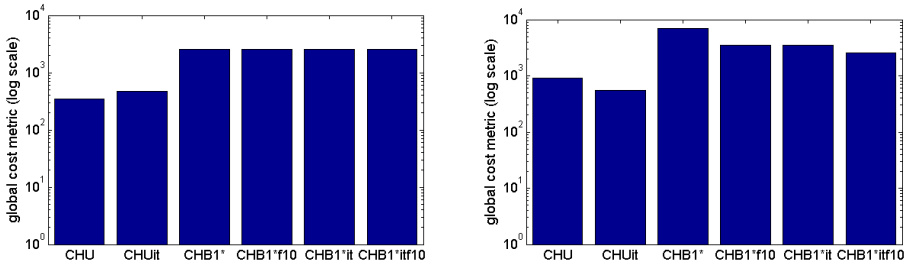


Figure 5.16: LR-PRFs in hardware with CP. 80-bit (left) and 120-bit (right) security.

It is worth noticing that the aforementioned conclusions are valid when we aim to protect a cryptosystem against power-analysis attacks. However, they should be mitigated when considering other kinds of side-channel attacks. For instance, Longo, De Mulder, Page, and Tunstall show in [GMPT15] that targeting such parallelized implementations with electromagnetic (EM) attacks may be devastating. Indeed, contrary to power consumption, EM emanations can be collected from very localized regions using micro antennas. In particular, as shown in [GMPT15], these antennas can isolate the signal corresponding to a specific AES S-box when it is fully parallelized in hardware. This is because, in practice, S-boxes can leak at different frequencies, what cannot be captured through power analysis. Even if there exists some countermeasures against EM attacks, such as space randomization [PBB+10], the security impact of parallelization should be nuanced somehow. Nevertheless, as far as we know, the EM attacks as well as the other side-channel attacks should not change our conclusions on the best choice of countermeasures between masking and leakage-resilient primitives which still mainly rely on the characteristics of the inherent primitive to protect.

5.6 Conclusion

Our results essentially show that masking and leakage-resilient constructions hardly combined constructively. For (stateful) PRGs, our experiments indicate that both for software and hardware implementations, a leakage-resilient design instantiated with an unprotected AES is the most efficient solution to reach any given security level. For stateless PRFs, they rather show that a bounded data complexity guarantee is (mostly) ineffective in bounding the (computational) complexity of the best attacks. So implementing masking and limiting the lifetime of the cryptographic implementation seems to be the best solution in this case.

Chapter 6

Conclusion and Perspectives

6.1 Conclusion

In this part, we tackled the three issues mentioned in the introduction in Chapter 1. We compared the two main countermeasures against side-channel attacks and we studied separately each of them. We proposed a leakage-resilient encryption scheme which seems to be efficient enough to be used in practice and we proposed automatic tools to verify and generate masked implementations.

6.2 Perspectives

While most of the DPA attacks are based on the divide-and-conquer strategy, we showed in this part that the key could be recovered even when it is not split for the computation. Thus, an interesting line of research is to analyze all such functions which are generally left unprotected in real implementations. They both need to be studied to evaluate the impact of side-channel attacks and they need to receive suited protections.

Regarding the countermeasures, one important issue that still needs to be addressed is to build protections which are well fitted to the underlying component. One important line of research is to exhibit a leakage model both close to the reality and convenient for the security proofs. This work has already started with the introduction of the noisy leakage model and its reduction to the t-threshold probing model. However, the bound is still not tight enough to define practical and secure masking schemes. Another line of research is to parametrize the countermeasures with the characterization of each component. Depending on the leakage features, the countermeasures could fix for instance the number of shares or the use of registers. To achieve these goals, we first need to be able to characterize efficiently the components and to define countermeasures that deal with the results.

Part IV

Conclusion of This Thesis

Chapter 1

Conclusion

In this thesis, we focused on several current issues related to power-analysis attacks, developing new attacks against existing schemes as well as proposing new countermeasures for existing schemes with provable security.

Contents

1.1 Power-Analysis Attacks on Multiplications

As presented in the second part of this thesis, we studied in particular the attacks on the field/ring multiplications between a 128-bit known variable and a 128-bit secret key. While side-channel analysis generally relies on divide-and-conquer strategies, in this context, each bit of the leaking values depends on all the 128 bits of the secret key. Thus, we exhibited new methods in order to deal with the large secret diffusion. Our first trial showed that despite the important dependency with the secret, the secret key could be recovered from the observation of several leaking multiplication outputs under small levels of noise. In our second trial, we showed that the same key recovery could be achieved under much more reasonable (*i.e.,* higher) levels of noise, such as those observed on the FPGA implementation that we attacked.

1.2 Countermeasures against Power-Analysis Attacks

As presented in the third part of this thesis, we also had the opportunity to focus on two main algorithmic side-channel countermeasures: the use of leakage-resilient primitives and masking.

First, in Chapter 2, we built a leakage-resilient encryption scheme, which seems to be efficient enough to be implemented in practice. In this context, we provided security proofs to guarantee its security in the specific model that we followed.

Then, in Chapters 3 and 4, we deeply looked at the masking schemes. First, we made an effort to verify the existing algorithms masked at high orders. For this purpose, we built an automatic verifier able to check the security of algorithms in the t-threshold probing model up to order 5 for limited program sizes (*e.g.,* multiplication). Thanks to this tool, we were able to confirm the presumed security of some algorithms, to recover existing flaws for others, and to discover new attack paths on flawed algorithms. In order to concentrate on larger programs, we then investigated the compositional properties of cryptographic algorithms. So far, several

works in the literature assumed the security of composition of small secure functions as long as the manipulated shares were refreshed before being reused. Unfortunately, this is not always correct. Thus, we used formal methods to automatically verify the security of the small blocks and then we exhibited new security properties and compositional theorems to securely combine them using an existing refreshing algorithm. As a result, we are now able to generate a masked algorithm at an arbitrary order from its non protected version.

Finally, to conclude the study of these two countermeasures, we compared their use in an industrial context according to the function being protected and the constraints in terms of efficiency and security. In particular, we showed that the best countermeasure depends highly on the primitive we are trying to protect. The use of leakage-resilient primitives is generally more suited for PRGs while masking usually performs better with PRFs.

Chapter 2

Perspectives

The subjects on which we focused in this thesis still leave several questions opened.

Contents

2.1 Power-Analysis Attacks

The attacks on the multiplication algorithms, described in Part II, introduced new paths to target different operations on which the divide-and-conquer strategy does not apply. Therefore, it is mandatory to identify algorithms whose intermediate variables does not depend on small parts of the secret but whose leakage can be expressed by linear (noisy) equations of the key bits. While most of them are left unmasked in current implementations, it is very important to mask them as well as the known sensitive functions (*e.g.,* AES Sboxes).

Furthermore, a different direction concerns the use of learning with errors algorithms. The investigation that we made in Part II showed that we can improve such existing methods for side-channel purposes. Namely, the main point is *a priori* to limit the number of measurements which corresponds, in such algorithms, to the number of queries. Although we made a few steps in this direction, such algorithms could likely be further improved and turned into a systematic tool for side-channel cryptanalysis.

2.2 Countermeasures against Power-Analysis Attacks

2.2.1 Leakage-Resilient Cryptography

The use of leakage-resilient primitives aims to provide security with respect to arbitrary leakage functions and does not take into account the particular device that we are trying to protect. However, in reality, it often fails to meet the performances and security requirements needed for practice. Concerning the security, the leakage-resilient cryptography model assumes that a quantity, upper bounded to λ bits, of useful leakage can be given to the attacker at each execution. Although most papers define the advantage of the final attacker according to this

quantity λ, we can notice that in most cases, it has to be very small for the security to be reasonable. However, in practice, the attacker generally gets much more useful information by observing all the points of each consumption trace. Nevertheless, this observation does not reveal security breaches in the leakage-resilient constructions, since many factors in security bounds are due to either the proof artifacts or the use of unrealistic security models. While there were a few improvements in terms of efficiency in the past few years, a proper evaluation of the corresponding security is still needed.

2.2.2 Leakage Models for Masking

The growing computation capabilities force cryptographers to continuously increase the security level of their algorithms and implementations. While first-order masking schemes were sufficient to protect most of the implementations a few years ago, they are becoming more and more obsolete. Therefore, the algorithm that we designed to verify the security of masking schemes still needs to be improved to target implementations at higher orders. Moreover, such an algorithm should also be combined with our compiler in order to allow a more efficient verification of the security of large higher-order programs in the t-threshold probing model, through the use of composition. Finally, significant improvements have been left as future work, such as the portability of these programs, the optimization of choice functions, the adaptation of the tools to any masking scheme (*e.g.,* multiplicative, additive), and the verification of all the existing higher-order masking schemes.

2.3 Evaluation of Attacks and Countermeasures

Finally, a mandatory point that was only briefly investigated in this thesis is the evaluation of both attacks and countermeasures on real devices. With respect to attacks, since the models that we used do not perfectly match the reality, it is fundamental to determine if the key can still be recovered in practice. Even if we cannot guarantee the success of these attacks on every device, concretely demonstrating a key recovery on at least one of them would already be quite significant. Regarding the countermeasures, their implementation on real devices would allow us to concretely evaluate their practicality. Furthermore, the application of well defined template attacks on some points could provide clues to justify their security.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AA04] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *2004 IEEE Symposium on Security and Privacy*, pages 3–11, Berkeley, California, USA, May 9–12, 2004. IEEE Computer Society Press. 21

[AB00] Michel Abdalla and Mihir Bellare. Increasing the lifetime of a key: a comparative analysis of the security of re-keying techniques. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 546–559, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany. 96

[AB09] Martin Albrecht and Gregory Bard. The M4RI Library – Version 20130416, 2009. 66

[ABF13] Michel Abdalla, Sonia Belaïd, and Pierre-Alain Fouque. Leakage-resilient symmetric encryption via re-keying. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 471–488, Santa Barbara, California, US, August 20–23, 2013. Springer, Heidelberg, Germany. 47, 94

[ABG+14] Joseph Ayo Akinyele, Gilles Barthe, Benjamin Grégoire, Benedikt Schmidt, and Pierre-Yves Strub. Certified synthesis of efficient batch verifiers. In *27th IEEE Computer Security Foundations Symposium, CSF 2014*. IEEE Computer Society, 2014. To appear. 128

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany. 62, 75

[ADD+15] Marcin Andrychowicz, Ivan Damgård, Stefan Dziembowski, Sebastian Faust, and Antigoni Polychroniadou. Efficient leakage resilient circuit compilers. In Kaisa Nyberg, editor, *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 311–329, San Francisco, CA, USA, April 20–24, 2015. Springer, Heidelberg, Germany. 40

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001. 20

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I*, volume

6755 of *Lecture Notes in Computer Science*, pages 403–415, Zurich, Switzerland, July 4–8, 2011. Springer, Heidelberg, Germany. 62

[APSQ06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14, Yokohama, Japan, October 10–13, 2006. Springer, Heidelberg, Germany. 35

[BBD+15a] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, and Benjamin Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. Cryptology ePrint Archive, Report 2015/506, 2015. `http://eprint.iacr.org/2015/506`. 94, 144, 147

[BBD+15b] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. 47, 94, 122

[BBD+15c] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. Cryptology ePrint Archive, Report 2015/060, 2015. `http://eprint.iacr.org/2015/060`. 123

[BCF+15a] Sonia Belaïd, Jean-Sébastien Coron, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, and Emmanuel Prouff. Improved side-channel analysis of finite-field multiplication. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 395–415, Saint-Malo, France, September 13–16, 2015. Springer, Heidelberg, Germany. 46, 55

[BCF+15b] Sonia Belaïd, Jean-Sébastien Coron, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, and Emmanuel Prouff. Improved side-channel analysis of finite-field multiplication. Cryptology ePrint Archive, Report 2015/542, 2015. `http://eprint.iacr.org/2015/542`. 79

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany. 27

[BDG+14] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2014. 121, 135, 143

[BDH+13] Sonia Belaïd, Fabrizio De Santis, Johann Heyszl, Stefan Mangard, Marcel Medwed, Jorn-Marc Schmidt, Francois-Xavier Standaert, and Stefan Tillich. Towards fresh

re-keying with leakage-resilient PRFs: Cipher design principles and analysis. Cryptology ePrint Archive, Report 2013/305, 2013. `http://eprint.iacr.org/2013/305`. 181

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press. 42, 99

[BDK+10] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 6355 of *Lecture Notes in Computer Science*, pages 46–63. Springer-Verlag, 2010. 128

[Ber12] Daniel J. Bernstein. Implementing "Practical leakage-resilient symmetric cryptography". CHES '12 rump session, 2012. Available at `http://cr.yp.to/talks/2012.09.10/slides.pdf`. 96, 170

[Bet] Luk Bettale. Magma Package: Hybrid Approach for Solving Multivariate Polynomial Systems over Finite Fields. 66

[BFG14] Sonia Belaïd, Pierre-Alain Fouque, and Benoît Gérard. Side-channel analysis of multiplications in GF(2128) - application to AES-GCM. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 306–325, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. 46, 55, 75, 85

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Joye and Moradi [JM15], pages 64–81. 36

[BGHZ11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. 126, 127, 143

[BGP+11] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, April 2011. 31

[BGS15] Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? *Cryptography and Communications*, 7(1):163–184, 2015. 48, 65, 94

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009. 39, 126, 127

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set

decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. 69

[BJV04] Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 432–450, Jeju Island, Korea, December 5–9, 2004. Springer, Heidelberg, Germany. 64

[BK07] Alex Biryukov and Dmitry Khovratovich. Two new techniques of side-channel cryptanalysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 195–208, Vienna, Austria, September 10–13, 2007. Springer, Heidelberg, Germany. 96

[BKMN09] Julien Brouchier, Tom Kean, Carol Marsh, and David Naccache. Temperature attacks. *IEEE Security & Privacy*, 7(2):79–82, 2009. 21

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd Annual ACM Symposium on Theory of Computing*, pages 435–440, Portland, Oregon, USA, May 21–23, 2000. ACM Press. 61

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003. 63, 68, 75

[BRNI13] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. Sleuth: Automated verification of software power analysis countermeasures. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 293–310, Santa Barbara, California, US, August 20–23, 2013. Springer, Heidelberg, Germany. 38, 39, 121, 125, 143

[CB08] D. Canright and Lejla Batina. A very compact "perfectly masked" S-box for AES. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS 08: 6th International Conference on Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 446–459, New York, NY, USA, June 3–6, 2008. Springer, Heidelberg, Germany. 120

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for S-boxes. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany. 59

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 188–205, Busan, South Korea, September 23–26, 2014. Springer, Heidelberg, Germany. 165

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany. 34, 35, 54, 142, 175

[CJRT05] Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors. *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*. Springer, 2005. 211

[Com] Common Criteria Portal. http://www.commoncriteriaportal.org/. 170

[Cor14] Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. 54, 143

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44, Vienna, Austria, September 10–13, 2007. Springer, Heidelberg, Germany. 120, 121, 137, 175

[CPRR14] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany. 37, 38, 120, 121, 132, 138, 139, 142, 143, 149, 150, 154, 156, 157, 158, 160, 161, 174, 199

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28, Redwood Shores, California, USA, August 13–15, 2003. Springer, Heidelberg, Germany. 32, 35, 173

[Cry] Cryptographic Key Length Recommendation. http://www.keylength.com/. 172

[CT91] Thomas M. Cover and Joy A. Thomas. *Information theory*. Wiley series in communications. Wiley, 1991. 64

[CTO+14] Mathieu Carbone, Sébastien Tiran, Sébastien Ordas, Michel Agoyan, Yannick Teglia, Gilles R. Ducharme, and Philippe Maurine. On adaptive bandwidth selection for efficient MIA. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2014. 31

[CZ06] Zhimin Chen and Yujie Zhou. Dual-rail random switching logic: A countermeasure to reduce side channel leakage. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 242–254, Yokohama, Japan, October 10–13, 2006. Springer, Heidelberg, Germany. 33

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. 35, 38, 40, 120, 122, 139, 142, 149, 161

[DEMM14] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, and Florian Mendel. On the security of fresh re-keying to counteract side-channel and fault attacks. In Joye and Moradi [JM15], pages 233–244. 54

[DFS15a] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. 36, 40, 120

[DFS15b] Stefan Dziembowski, Sebastian Faust, and Maciej Skorski. Noisy leakage revisited. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 159–188, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. 36

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th Annual Symposium on Foundations of Computer Science*, pages 293–302, Philadelphia, Pennsylvania, USA, October 25–28, 2008. IEEE Computer Society Press. 40, 96

[DP10] Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 21–40, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. 41, 95, 96

[DPAK12] G. Bertonand J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. KECCAK implementation overview, May 2012. `http://keccak.noekeon.org/`. 138

[DRS+12] François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon. Efficient Removal of Random Delays from Embedded Software Implementations Using Hidden Markov Models. In Stefan Mangard, editor, *CARDIS*, volume 7771 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2012. 65, 176

[DSV+14] François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Jean-Baptiste Mairy, and Yves Deville. Efficient selection of time samples for higher-order DPA with projection pursuits. Cryptology ePrint Archive, Report 2014/412, 2014. `http://eprint.iacr.org/2014/412`. 70

[Eck85] Wim Van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? In *Computer & Security, 4*, pages 269–286, 1985. 23

[EGG+12] Thomas Eisenbarth, Zheng Gong, Tim Güneysu, Stefan Heyse, Sebastiaan Indesteege, Stéphanie Kerckhof, François Koeune, Tomislav Nad, Thomas Plos, Francesco Regazzoni, François-Xavier Standaert, and Loïc van Oldeneel tot Oldenzeel. Compact implementation and performance evaluation of block ciphers in ATtiny devices. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT 12: 5th International Conference on Cryptology in Africa*, volume 7374 of *Lecture Notes in Computer Science*, pages 172–187, Ifrance, Morocco, July 10–12, 2012. Springer, Heidelberg, Germany. 174, 177

[Eur] Europay Mastercard Visa. http://www.emvco.com/. 170

[EW14] Hassan Eldib and Chao Wang. Synthesis of masking countermeasures against side channel attacks. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification, CAV 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2014. 38, 121, 143

[EWS14] Hassan Eldib, Chao Wang, and Patrick Schaumont. SMT-based verification of software countermeasures against side-channel attacks. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 62–77. Springer, 2014. 38, 39, 121, 125, 143

[EWTS14] Hassan Eldib, Chao Wang, Mostafa M. I. Taha, and Patrick Schaumont. QMS: evaluating the side-channel resistance of masked software from source code. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*, pages 1–6. ACM, 2014. 38, 121

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero F5. In *International Symposium on Symbolic and Algebraic Computation Symposium - ISSAC*, 2002. 66

[Fer05] Niels Ferguson. Authentication weaknesses in GCM, 2005. `http://csrc.nist.gov/groups/ST/toolkit/BCM/`. 54, 72

[FH08] Julie Ferrigno and Martin Hlavác. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008. 21

[FLD12] Yunsi Fei, Qiasi Luo, and A. Adam Ding. A statistical model for DPA with novel algorithmic confusion analysis. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 233–250, Leuven, Belgium, September 9–12, 2012. Springer, Heidelberg, Germany. 176

[FPS12] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 213–232, Leuven, Belgium, September 9–12, 2012. Springer, Heidelberg, Germany. 41, 47, 95, 96, 97, 99, 100, 102

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 135–156, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. 35, 40

[FRR+14] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from computationally bounded and noisy leakage. *SIAM Journal on Computing*, 43(5):1564–1614, 2014. 143

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. 18

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442, Washington, D.C., USA, August 10–13, 2008. Springer, Heidelberg, Germany. 30, 31

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press. 44, 96, 170

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 129

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. Workshop NIAT 2011, 2011. 83

[GJL14] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 1–20, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. 62

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261, Paris, France, May 14–16, 2001. Springer, Heidelberg, Germany. 21

[GMO+14] Jake Longo Galea, Daniel P. Martin, Elisabeth Oswald, Daniel Page, Martijn Stam, and Michael Tunstall. Simulatable leakage: Analysis, pitfalls, and new constructions. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 223–242, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. 170

[GMPT15] Jake Longo Galea, Elke De Mulder, Dan Page, and Michael Tunstall. SoC it to EM: ElectroMagnetic side-channel attacks on a complex system-on-chip. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems –*

CHES 2015, volume 9293 of *Lecture Notes in Computer Science*, pages 620–640, Saint-Malo, France, September 13–16, 2015. Springer, Heidelberg, Germany. 185

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Heidelberg, Germany. 34

[GPQ11] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255, Nara, Japan, September 28 – October 1, 2011. Springer, Heidelberg, Germany. 143

[GPS14] Vincent Grosso, Emmanuel Prouff, and François-Xavier Standaert. Efficient masked S-boxes processing - A step forward -. In David Pointcheval and Damien Vergnaud, editors, *AFRICACRYPT 14: 7th International Conference on Cryptology in Africa*, volume 8469 of *Lecture Notes in Computer Science*, pages 251–266, Marrakesh, Morocco, May 28–30, 2014. Springer, Heidelberg, Germany. 54

[GS12] Benoît Gérard and François-Xavier Standaert. Unified and optimized linear collision attacks and their application in a non-profiled setting. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 175–192, Leuven, Belgium, September 9–12, 2012. Springer, Heidelberg, Germany. 96

[GSF13] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: How large is the gap for AES? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 400–416, Santa Barbara, California, US, August 20–23, 2013. Springer, Heidelberg, Germany. 170, 174, 177

[GSS09] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 633–654. IOS Press, 2009. 39, 125

[GST13] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. Cryptology ePrint Archive, Report 2013/857, 2013. `http://eprint.iacr.org/2013/857`. 21

[HGJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. 75

[HLWW12] Carmit Hazay, Adriana Lopez-Alt, Hoeteck Wee, and Daniel Wichs. Leakage-resilient cryptography from minimal assumptions. Cryptology ePrint Archive, Report 2012/604, 2012. `http://eprint.iacr.org/2012/604`. 96

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06: 4th International Conference on Applied Cryptography and Network Security*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, Singapore, June 6–9, 2006. Springer, Heidelberg, Germany. 175

[HP08] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based MAC algorithms. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 144–161, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany. 54

[Imp95] Russell Impagliazzo. A Personal View of Average-Case Complexity. In *Structure in Complexity Theory Conference*, pages 134–147, 1995. 103, 114

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: Keeping secrets in tamperable circuits. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 308–327, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 21

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. 35, 39, 59, 122, 127, 137, 143, 145, 150, 161, 175

[Jaf07] Joshua Jaffe. A first-order DPA attack against AES in counter mode with unknown initial counter. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 1–13, Vienna, Austria, September 10–13, 2007. Springer, Heidelberg, Germany. 54, 96

[JM15] Marc Joye and Amir Moradi, editors. *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*. Springer, 2015. 203, 206

[Jou06] Antoine Joux. Authentication Failures in NIST version of GCM, 2006. `http://csrc.nist.gov/CryptoToolkit/modes/`. 54

[Kir11] Paul Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011. `http://eprint.iacr.org/2011/377`. 62, 68, 75

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany. 21, 24, 26, 175

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany. 21, 23, 26

[Koc03] Paul C. Kocher. Leak-resistant cryptographic indexed key update. Patent, 03 2003. US 6539092. 40, 96, 97, 99

[KSK+10] Toshihiro Katashita, Akashi Satoh, Katsuya Kikuchi, Hiroshi Nakagawa, and Masahiro Aoyagi. Evaluation of DPA Characteristics of SASEBO for Board Level Simulation, 2010. 65, 176

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359, Maiori, Italy, September 6–8, 2006. Springer, Heidelberg, Germany. 62, 68

[LPR+14] Victor Lomné, Emmanuel Prouff, Matthieu Rivain, Thomas Roche, and Adrian Thillard. How to estimate the success rate of higher-order side-channel attacks. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 35–54, Busan, South Korea, September 23–26, 2014. Springer, Heidelberg, Germany. 35, 176

[LSP04] Kerstin Lemke, Kai Schramm, and Christof Paar. DPA on n-bit sized Boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 205–219, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany. 27

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In Chekuri et al. [CJRT05], pages 378–389. 68, 78

[Man04] Stefan Mangard. Hardware countermeasures against DPA? a statistical analysis of their effectiveness. In Tatsuaki Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235, San Francisco, CA, USA, February 23–27, 2004. Springer, Heidelberg, Germany. 65

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251, Worcester, Massachusetts, USA, August 17–18, 2000. Springer, Heidelberg, Germany. 34, 40

[MM12] Amir Moradi and Oliver Mischke. Glitch-free implementation of masking in modern FPGAs. In *HOST*, pages 89–95. IEEE, 2012. 175

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 125–139, Santa Barbara, California, USA, August 17–20, 2010. Springer, Heidelberg, Germany. 96

[MOBW13] Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? an a priori statistical power analysis of leakage detection

tests. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 486–505, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. 120

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007. 20, 26, 32, 35, 170, 176

[MOPT12] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler assisted masking. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 58–75, Leuven, Belgium, September 9–12, 2012. Springer, Heidelberg, Germany. 38, 121, 125, 143

[Mor12] Amir Moradi. Statistical tools flavor side-channel collision attacks. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 428–445, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. 96

[MOS11] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011. 173, 178

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365, San Francisco, CA, USA, February 14–18, 2005. Springer, Heidelberg, Germany. 175

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany. 174, 177, 178

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171, Edinburgh, UK, August 29 – September 1, 2005. Springer, Heidelberg, Germany. 175

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. 35, 41, 60

[MSGR10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10: 3rd International Conference on Cryptology in Africa*, volume 6055 of *Lecture Notes in Computer Science*, pages 279–296, Stellenbosch, South Africa, May 3–6, 2010. Springer, Heidelberg, Germany. 46, 54, 58, 59, 73, 75, 96

[MSJ12] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards super-exponential side-channel security with efficient leakage-resilient PRFs. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 193–212, Leuven, Belgium, September 9–12, 2012. Springer, Heidelberg, Germany. 44, 48, 96, 100, 171, 172, 178, 179, 183, 185, 195

[MV05] D. A. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM), May 2005. 54

[OC14] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An open-source platform for hardware embedded security research. Cryptology ePrint Archive, Report 2014/204, 2014. `http://eprint.iacr.org/2014/204`. 82

[OM07] Elisabeth Oswald and Stefan Mangard. Template attacks on masking - resistance is futile. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 243–256, San Francisco, CA, USA, February 5–9, 2007. Springer, Heidelberg, Germany. 120

[OMPR05] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the AES S-box. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption – FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423, Paris, France, February 21–23, 2005. Springer, Heidelberg, Germany. 120

[PBB+10] François Poucheret, Lyonel Barthe, Pascal Benoit, Lionel Torres, Philippe Maurine, and Michel Robert. Spatial EM jamming: A countermeasure against EM analysis? In *18th IEEE/IFIP VLSI-SoC 2010, IEEE/IFIP WG 10.5 International Conference on Very Large Scale Integration of System-on-Chip, Madrid, Spain, 27-29 September 2010*, pages 105–110. IEEE, 2010. 186

[PC14] Gordon Procter and Carlos Cid. On weak keys and forgery attacks against polynomial-based MAC schemes. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 287–304, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany. 54

[Pie06] Krzysztof Pietrzak. Composition implies adaptive security in minicrypt. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 328–338, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 115

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany. 43, 95, 96, 97, 98, 101, 112, 116

[Pie12] Krzysztof Pietrzak. Cryptography from learning parity with noise. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer Berlin Heidelberg, 2012. 78

[PL14] Martin Pettai and Peeter Laud. Automatic proofs of privacy of secure multi-party computation protocols against active adversaries. Cryptology ePrint Archive, Report 2014/240, 2014. `http://eprint.iacr.org/2014/240`. 39, 125

[PM05] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 172–186, Edinburgh, UK, August 29 – September 1, 2005. Springer, Heidelberg, Germany. 33

[PR11] Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78, Nara, Japan, September 28 – October 1, 2011. Springer, Heidelberg, Germany. 174, 177

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. 35, 38, 40, 120, 139, 142

[PS08] Krzysztof Pietrzak and Johan Sjödin. Weak pseudorandom functions in minicrypt. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 423–436, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany. 115

[PSDQ05] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved higher-order side-channel attacks with FPGA experiments. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 309–323, Edinburgh, UK, August 29 – September 1, 2005. Springer, Heidelberg, Germany. 120, 176

[PSP+08] Christophe Petit, François-Xavier Standaert, Olivier Pereira, Tal Malkin, and Moti Yung. A block cipher based pseudo random number generator secure against side-channel key recovery. In Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08: 3rd Conference on Computer and Communications Security*, pages 56–65, Tokyo, Japan, March 18–20, 2008. ACM Press. 43

[Pug89] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. In *WADS*, pages 437–449, 1989. 97, 101

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001. 21

[Riv09] Matthieu Rivain. On the exact success rate of side channel analysis in the gaussian model. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 165–183, Sackville, New Brunswick, Canada, August 14–15, 2009. Springer, Heidelberg, Germany. 35, 176

[Rot12] Guy N. Rothblum. How to compute under $\dashv\rfloor^0$ leakage without secure hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 552–569, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. 40

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, California, USA, August 17–20, 2010. Springer, Heidelberg, Germany. 37, 38, 39, 47, 54, 120, 133, 134, 137, 138, 139, 140, 142, 143, 146, 149, 150, 154, 155, 158, 160, 174, 177, 197, 199

[RP11] Thomas Roche and Emmanuel Prouff. Higher-order glitches free implementation of the AES using secure multi-party computation protocols – extended version – . Cryptology ePrint Archive, Report 2011/413, 2011. `http://eprint.iacr.org/2011/413`. 184

[RS09] Mathieu Renauld and François-Xavier Standaert. Algebraic side-channel attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Inscrypt*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2009. 173

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. 18

[RSV09] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the AES: Why time also matters in DPA. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111, Lausanne, Switzerland, September 6–9, 2009. Springer, Heidelberg, Germany. 173

[RYS] Francesco Regazzoni, Wang Yi, and François-Xavier Standaert. FPGA implementations of the AES masked against power analysis attacks. In Sorin Huss and Werner Schindler, editors, proceedings of *COSADE* 2011, pp 56-66, Darmstadt, Germany, February 2011. 174, 177

[SA03] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12, Redwood Shores, California, USA, August 13–15, 2003. Springer, Heidelberg, Germany. 21

[Saa12] Markku-Juhani Olavi Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 216–225, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany. 54

[SGV09] François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC 08: 11th International Conference on Information Security and Cryptology*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267, Seoul, Korea, December 3–5, 2009. Springer, Heidelberg, Germany. 31

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/2004/332`. 105, 106, 109

[Sko05] Sergei P. Skorobogatov. Semi-invasive attacks – a new approach to hardware security analysis, 2005. 21

[Sko09] Sergei P. Skorobogatov. Using optical emission analysis for estimating contribution to power analysis. In Luca Breveglieri, Israel Koren, David Naccache, Elisabeth Oswald, and Jean-Pierre Seifert, editors, *Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009*, pages 111–119. IEEE Computer Society, 2009. 21

[SLFP04] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A collision-attack on AES: Combining side channel- and differential-attack. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 163–175, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany. 96

[SM03] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003. 124

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany. 32, 173, 176

[SNK+12] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of AES - photonic side channel analysis for the rest of us. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 41–57, Leuven, Belgium, September 9–12, 2012. Springer, Heidelberg, Germany. 21

[SP06] Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225, San Jose, CA, USA, February 13–17, 2006. Springer, Heidelberg, Germany. 120, 121, 138

[SPY+10] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage Resilient Cryptography in Practice. *Towards Hardware-Intrinsic Security, Information Security and Cryptography*, pages 99–134, 2010. 41, 42, 95

[SPY13] François-Xavier Standaert, Olivier Pereira, and Yu Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 335–352, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. 170

[SS79] Richard Schroeppel and Adi Shamir. A T sˆ2 = oˆn) time/space tradeoff for certain np-complete problems. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 328–336. IEEE Computer Society, 1979. 75

[SVO+10] François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. 142, 176, 183

[Tur93] Berwin A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, 1993. 31

[TV03] Kris Tiri and Ingrid Verbauwhede. Securing encryption algorithms against DPA at the logic level: Next generation smart card technology. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 125–136, Cologne, Germany, September 8–10, 2003. Springer, Heidelberg, Germany. 33

[Ven15] Srinivas Vivek Venkatesh. *Practical provable security against side-channel attacks.* Phd thesis, University of Luxembourg, 2015. 42

[VGRS13] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012: 19th Annual International Workshop on Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406, Windsor, Ontario, Canada, August 15–16, 2013. Springer, Heidelberg, Germany. 71

[VGS13] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 126–141, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. 32, 33, 72, 173, 179

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. 96

[VRG] TELECOM ParisTech VLSI Research Group. The DPA Contest 2008/2009. 24, 25, 28, 195

[VS09] Nicolas Veyrat-Charvillon and François-Xavier Standaert. Mutual information analysis: How, when and why? In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 429–443, Lausanne, Switzerland, September 6–9, 2009. Springer, Heidelberg, Germany. 31

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. 79

[WO11a] Carolyn Whitnall and Elisabeth Oswald. A comprehensive evaluation of mutual information analysis using a fair evaluation framework. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 316–334, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. 31

[WO11b] Carolyn Whitnall and Elisabeth Oswald. A fair evaluation framework for comparing side-channel distinguishers. *J. Cryptographic Engineering*, 1(2):145–160, 2011. 31

[YS13] Yu Yu and François-Xavier Standaert. Practical leakage-resilient pseudorandom objects with minimum public randomness. In Ed Dawson, editor, *Topics in Cryptology – CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 223–238, San Francisco, CA, USA, February 25 – March 1, 2013. Springer, Heidelberg, Germany. 43, 95, 102, 115, 116, 117, 195

[ZZT09] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1), 2009. 21

Résumé

Les attaques par canaux auxiliaires sont les attaques les plus efficaces contre les systèmes cryptographiques. Alors que les attaques classiques n'exploitent que les entrées et sorties des algorithmes cryptographiques, les attaques par canaux auxiliaires utilisent également les fuites physiques du composant sous-jacent. Dans cette thèse, nous nous intéressons aux attaques par canaux auxiliaires qui exploitent la consommation de courant des composants pour retrouver les clefs secrètes. Ces attaques sont désignées par le terme *attaques par analyse de courant*.

La majorité des attaques par analyse de courant existantes repose sur l'observation de variables dépendant uniquement de quelques bits de secret avec la stratégie diviser-pour-régner. Dans cette thèse, nous exhibons de nouvelles attaques qui exploitent l'observation de variables intermédiaires largement dépendantes de grands secrets. Notamment, nous montrons qu'en observant uniquement la fuite physique du résultat d'une multiplication de Galois entre une clef secrète de 128 bits et plusieurs messages connus, nous pouvons en déduire un système d'équations avec erreurs puis retrouver cette clef secrète.

En parallèle, nous nous intéressons aux deux contre-mesures algorithmiques les plus répandues contre ces attaques par analyse de courant : les fonctions intrinsèquement résistantes aux fuites physiques et les schémas de masquage. Dans un premier temps, nous définissons un schéma de chiffrement résistant aux fuites physiques qui repose sur un rafraîchissement régulier de la clef secrète. Nous prouvons la sécurité de ce schéma dans le modèle de cryptographie résistante aux fuites (en anglais, *leakage-resilient cryptography*). Dans un second temps, nous construisons, à l'aide des méthodes formelles, un outil permettant de vérifier automatiquement la sécurité d'implémentations masquées. Nous exhibons également de nouvelles propriétés de sécurité, ainsi que des propriétés de composition qui nous permettent de générer une implémentation masquée à n'importe quel ordre à partir d'une implémentation non protégée. Finalement, nous présentons une étude de comparaison entre ces deux contre-mesures algorithmiques dans le but d'aider les experts industriels à déterminer la meilleure protection à intégrer dans leurs produits en fonction de leurs contraintes en termes de sécurité et de performances.

mots-clés : attaques par canaux auxiliaires, attaques par analyse de courant, cryptographie résistante aux fuites physiques, masquage aux ordres supérieurs.

Abstract

Side-channel attacks are the most efficient attacks against cryptosystems. While the classical black-box attacks only exploit the inputs and outputs of cryptographic algorithms, side-channel attacks also get use of the physical leakage released by the underlying device during algorithms executions. In this thesis, we focus on one kind of side-channel attacks which exploits the power consumption of the underlying device to recover the algorithms secret keys. They are gathered under the term *power-analysis attacks*.

Most of the existing power-analysis attacks rely on the observations of variables which only depend on a few secret bits using a divide-and-conquer strategy. In this thesis, we exhibit new kinds of attacks which exploit the observation of intermediate variables highly dependent on huge secrets. In particular, we show how to recover a 128-bit key by only recording the leakage of the Galois multiplication's results between several known messages and this secret key.

We also study two commonly used algorithmic countermeasures against side-channel attacks: leakage-resilience and masking. On the one hand, we define a leakage-resilient encryption scheme based on a regular update of the secret key and we prove its security. On the other hand, we build, using formal methods, a tool to automatically verify the security of masked algorithms. We also exhibit new security and compositional properties which can be used to generate masked algorithms at any security order from their unprotected versions. Finally, we propose a comparison between these two countermeasures in order to help industrial experts to determine the best protection to integrate in their products, according to their constraints in terms of security and performances.

keywords: side-channel attacks, Differential Power Analysis, leakage-resilient cryptography, higher-order masking.