# Pre-silicon evaluation of secured circuit against side-channel attacks

Sofiane Takarabt

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAT015

Thèse de doctorat

TELECOM Paris

IP PARIS

# Évaluation pré-silicium des circuits sécurisés face aux attaques par canal auxiliaire

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Secure-IC & Télécom Paris

École doctorale IP Paris (ED-626)
Spécialité de doctorat : Electronique et optoélectronique

Thèse présentée et soutenue à Paris, le 06/07/2021, par

**SOFIANE TAKARABT**

Composition du Jury :

| | |
|---|---|
| M. Guénaël RENAULT<br>Professeur Chargé de Cours, École Polytechnique, ANSSI | President |
| Mme Svetla NIKOVA<br>Research Professor, Katholieke Universiteit Leuven | Rapporteur |
| M. Lilian BOSSUET<br>Professeur, Université Jean Monnet, Saint-Etienne | Rapporteur |
| Mme. Karine HEYDEMANN<br>Maître de conférence, Sorbonne Université, Paris 6 | Examinateur |
| M. Emmanuel PROUFF<br>Chercheur Associé, Sorbonne Université, ANSSI | Examinateur |
| M. Benoit GÉRARD<br>Chercheur associé, IRISA, DGA-MI | Invité |
| M. Yves MATHIEU<br>Directeur d'études, Télécom Paris | Directeur de thèse |
| M. Laurent SAUVAGE<br>Maître de conférence, Télécom Paris | Co-encadrant de thèse |
| M. Youssef SOUISSI<br>Ingénieur de Recherche, Secure-IC | Co-encadrant de thèse |

II

# Contents


**1 Introduction** **1**

1.1 Context . . . . . 1

1.2 Physical threats . . . . . 2

1.3 Protection . . . . . 2

1.4 Pre-silicon evaluation . . . . . 3

1.4.1 Empirical evaluation . . . . . 3

1.4.2 Formal evaluation . . . . . 4

1.5 Objectives . . . . . 5

**2 Related works** **7**

2.1 Modern cryptography . . . . . 7

2.1.1 Symmetric cryptography . . . . . 7

2.1.2 Asymmetric cryptography . . . . . 8

2.2 Physical attacks . . . . . 9

2.2.1 Timing attack . . . . . 9

2.2.2 Horizontal attack . . . . . 10

2.2.3 Differential power attack . . . . . 10

2.3 Countermeasure against physical attacks . . . . . 17

2.3.1 Hiding . . . . . 17

2.3.2 Masking . . . . . 18

2.3.3 Boolean masking in presence of glitches . . . . . 20

2.4 High-order DPA . . . . . 24

2.5 Evaluation methods . . . . . 25

2.5.1 Probing model . . . . . 25

2.5.2 Formal verification methods . . . . . 26

2.5.3 Pre-silicon security verification . . . . . 29

2.6 SCA & performance issue . . . . . 31

2.7 Fault analyses . . . . . 31

2.7.1 Fault model . . . 32
2.7.2 DFA on asymmetric algorithms . . . 32
2.7.3 DFA on symmetric algorithms . . . 32
2.8 FIB for probing . . . 34
2.8.1 FIB - Brief description . . . 35
2.8.2 Micro-probing attack . . . 37

**I Passive attack and countermeasures 39**

**3 Pre-silicon to Post-silicon Analysis 41**
3.1 Introduction . . . 41
3.2 Leakage and security level . . . 42
3.3 End-to-end security evaluation . . . 43
3.3.1 Experimental observations . . . 43
3.3.2 From virtual to real SCA . . . 46
3.4 Discussion . . . 47
3.5 Conclusion . . . 48

**4 Post-Synthesis Analysis of a Masked Implementation 49**
4.1 Introduction . . . 49
4.2 Analysis of a masked implementation . . . 50
4.2.1 RTL analyses . . . 50
4.2.2 PS analyses . . . 52
4.2.3 High-level leakage assessment . . . 56
4.2.4 Leakage exploitation - Collision attack . . . 56
4.3 Discussion . . . 59
4.4 Conclusion . . . 59

**5 Formal Analysis of a Masked Implementation 61**
5.1 Introduction . . . 61
5.2 Preliminaries . . . 62
5.2.1 Notations . . . 62
5.2.2 Concepts . . . 63
5.3 Formalization of netlist static analysis . . . 64
5.3.1 Motivating examples . . . 64
5.3.2 Formal model - Glitch-extension . . . 66
5.3.3 Leakage detection algorithm . . . 67

5.3.4 Analysis of the masked *AND* gate . . . 68
5.3.5 Masked *AND* with propagation time . . . 69
5.3.6 Securing the masked *AND* gate . . . 71
5.3.7 Our glitch-resistant masked AND gate . . . 72
5.4 Practical case: masked inversion in $GF_{2^4}$ . . . 73
5.4.1 Canright AES S-Box . . . 73
5.4.2 Formal based evaluation of Canright inverter . . . 74
5.4.3 Our $GF_{2^4}$ inverter - Compact and provably secure . . . 75
5.5 Actual exploitation of vulnerable netlists . . . 79
5.5.1 SCA evaluation of Canright inverter - Digital simulation . . . 79
5.5.2 SCA evaluation of Canright inverter - EM Acquisition . . . 80
5.6 Conclusion . . . 85

**II Active attack and countermeasures 87**

**6 Fault Injection Analyses Assisted by Simulation 89**
6.1 Introduction . . . 89
6.2 Fault model . . . 90
6.2.1 Clock-glitch injection . . . 90
6.2.2 Laser injection . . . 91
6.3 Fault detection on protected implementation . . . 92
6.3.1 AES-EDC implementation . . . 92
6.3.2 Design evaluation . . . 95
6.4 Conclusion . . . 98

**7 Evaluation Against Focused Ions Beam for Probing Attack 99**
7.1 Introduction . . . 99
7.2 Probing model . . . 100
7.3 FIB for probing attack . . . 100
7.4 Methodology of FIB for probing . . . 101
7.4.1 Sensitive signals identification . . . 101
7.4.2 Sensitive signal location . . . 103
7.4.3 FIB probing model . . . 103
7.4.4 FIB access methodology . . . 104
7.5 Study-case on AES . . . 108
7.5.1 Target IP . . . 108
7.5.2 Sensitive signal location . . . 109

7.5.3 FIB-probing evaluation . . . 109
7.5.4 Security improvements . . . 111
7.5.5 Discussion . . . 112
7.6 Conclusion . . . 113

**8 Conclusion & Perspectives** **115**
8.1 Conclusion . . . 115
8.2 Perspectives . . . 117
8.3 List of Publications . . . 119
8.3.1 Book chapter . . . 119
8.3.2 Journal . . . 119
8.3.3 Conference . . . 119
8.3.4 Workshop . . . 120

**A** **121**
A.1 Evaluation of delay insertion countermeasure . . . 121
A.2 Masked inversion in $GF_{16}$ . . . 122
A.3 Uniformity and mask reuse . . . 123

# Remerciements

C'est par ces lignes que s'achèvent trois longues et merveilleuses années de thèse, grâce à un travail de collaboration très solide et unique entre l'entreprise Secure-IC et l'école Télécom-Paris. Je tiens à remercier chaleureusement toutes les personnes qui m'ont aidé de près ou de loin durant la préparation de ce manuscrit et l'édition de ma thèse, notamment mon directeur M. Yves MATHIEU, et mon co-encadrant M. Laurent SAUVAGE. Je leurs exprime toute ma reconnaissance et considération, pour leurs disponibilités et leur travail de supervision, sans oublier toutes les connaissances que j'ai pu acquérir grâce à eux. Je n'oublierai certainement pas tous ces moments d'échanges précieux et exceptionnels.

J'en profite parallèlement pour exprimer toute ma gratitude à tous les membres du jury. Merci à M. Guénaël RENAULT, Mme. Svetla NIKOVA, M. Lilian BOUSSUET, Mme Karine HEYDEMANN, M. Emmanuel PROUFF et M. Benoit GÉRARD de m'avoir fait l'honneur d'accepter d'être les membres de mon jury de thèse. Plus particulièrement, je voudrais remercier M. Lilian BOUSSUET et Mme Svetla NIKOVA pour avoir examiné mon travail de thèse, et ainsi participé à l'amélioration du présent manuscrit. Je remercier également M. Guénaël RENAULT, avec qui j'ai partagé un long parcours scolaire, pour avoir présidé la soutenance, et participé au comité de suivi individuel de mi-parcours. Merci à Mme Roselyne CHOTIN AVOT pour sa participation et son implication dans ce processus.

Je tiens fortement à remercier M. Sylvain GUILLEY qui m'a accompagné durant cette thèse et avec qui j'ai collaboré sur divers projets intéressants et motivants. Je remercie également et au même titre, mon co-encadrant industriel M. Youssef SOUISSI, pour son soutien et son accompagnement durant ces trois années de thèse.

J'aimerais aussi adresser toute ma reconnaissance à M. Hassan TRIQUI pour m'avoir accueilli au sein de l'entreprise Secure-IC, et avoir ainsi créé l'opportunité de cette collaboration. Merci également à M. Philippe NGUYEN et M. Robert NGUYEN pour leur accueil au sein de l'équipe Threat Analysis.

Cette thèse n'aurait pas pu aboutir sans l'assistance de toute l'équipe administrative. Un grand merci à Mme Chantal CADIAT, Mme Marianna BAZIZ, Mme Florence BESNARD, Mme Yvonne BANSIMBA, Mme Anne VIDAL et M. Alain SIBILLE coté Télécom-Paris, et à Mme Anne-Sophie DAVID et Mme Julie ROSÉ coté Secure-IC.

Je remercie tous les membres de l'équipe SSH de Télécom-Paris pour leur assistance, accueil, et tous les moments d'échanges et interactions. Merci à M. Jean-Luc DANGER, M. Tarik GRABA, M. Abdelmalek SI MERABET et M. Ulrich KÜHNE.

Certainement, je n'oublierai pas mes collègues et mes amis de Secure-IC, Meziane HAMOUDI, Khaled KARRAY, Oualid TRABELSI, Xuan Thuy NGO, Nicolas BRUNEAU, Sébastien CARRÉ, Adrien FACON, Florent LOZAC'H, Alexander SCHAUB, Michaël TIMBERT, Tristan GAUDRON, Ismail GUEDIRA, Michaël PAUCARD, Fati ELHASSANI, SAMI TOUILI, Janie QUENOUILLER, Valentin PELTIER, Antoine BOUVET, Aurélien LOUVET, Lukas VLASAK, Amina BEL-KORCHI, Patrick LEJOLY, Marie-Jeanne DALMEIDA, Mohammed YOUSNI et Azeddine AMARIR pour tous les moments conviviaux que nous avons passés ensemble.

Enfin, je voudrais remercier énormément toute ma famille (mon père, ma mère, tous mes frères et soeurs, et ma femme) pour leur soutien infaillible, leur encouragement ainsi que leur implication. Leur assistance m'a été indispensable et essentielle.

# Abstract

All modern systems such as smart cards, smartphones and IoTs are based on embedded electronic systems. To communicate with the outside world, these systems implement cryptographic functions and manipulate sensitive data. Indeed, to guarantee the confidentiality, authenticity and integrity of sensitive data, modern cryptographic algorithms offer robust primitives based on very strong theoretical foundations. However, these same implementations can leak information related to the sensitive data they handle, such as secret and private keys.

This information leak can be exploited by side-channel attacks or fault injections. For this reason, cryptographic circuits should be certified, to be able to deploy them in real applications. This step is necessary and very expensive when several iterations between the different actors are repeated. The main objective of this thesis is to improve the methods of pre-silicon evaluation.

Better characterising the exploitable leakages by a potential attacker allows to define, before manufacture of the final circuit, the expected level of security. To do this, we conduct an end-to-end comparison between a virtual and a real analysis. We use some prototypes of circuits implementing cryptographic functions for which we have, on the one hand, samples allowing measurements of electromagnetic radiation, and on the other hand, design data enabling simulations to be carried out at digital level.

Thus, we can limit the differences related to the intrinsic behavior of the target on the one hand, and identify the factors making these implementations vulnerable against certain attacks on the other hand. Simulations are performed at the numerical level to best assess all information related to the measurement of changes in power consumption. Once the leakage sources have been identified, the simulation can be accelerated by limiting the assessment to the critical parts only, and depending on the considered security level define by an attacke rmodel. These evaluations cover the different design levels, namely, Register Transfert Level, Post-SYntheis, Place & Route and Post-Layout.

# Résumé de la thèse en français

## Chapitre 1 – Introduction

La protection de la vie privée demeure une question importante depuis des siècles, et la capacité à communiquer des informations sensibles joue un rôle fondamental dans la société. La cryptographie antique se basait habituellement sur des permutations simples afin de dissimuler les messages envoyés, comme par exemple, les chiffrements de César et Vigenère. Des techniques plus fiables ont été inventées ensuite, en adoptant des propriétés robustes afin d'assurer un chiffrement quasi-parfait. Les premiers algorithmes de chiffrement modernes sont principalement le DES, le Triple-DES et l'AES. Leur sécurité mathématique est fortement liée à la longueur de la clé.

Lorsque deux personnes veulent chiffrer leurs communications sur un canal non sûr, ils doivent au préalable générer une clé (symétrique) partagée. Ce mécanisme est assuré par les protocoles asymétriques. Les algorithmes les plus utilisés actuellement sont basés sur le problème de factorisation des grands nombres et le logarithme discret. Théoriquement, ces algorithmes sont considérés comme sûrs et sécurisés. Les attaques mathématiques les plus efficaces connues jusqu'à aujourd'hui ont une complexité exponentielles (au mieux sous-exponentielles) en fonction de la taille de la clé. Néanmoins, sur les systèmes électroniques embarqués, la robustesse mathématique ne suffit pas pour sécuriser les données sensibles. Les attaques physiques sont considérées aujourd'hui comme une réelle menace contre les implémentations cryptographiques, et elles ont déjà mis en échec certaines applications.

Pour protéger les données sensibles contre de telles attaques, il est nécessaire de mettre en œuvre des variantes robustes adaptées aux différents angles d'attaques offerts à un attaquant potentiel. Pour les cibles distantes, une protection contre les attaques temporelles suffit. Dans le cas des cibles accessibles physiquement (comme les cartes à puce), une couche de protection supplémentaire est indispensable pour lutter contre les attaques exploitant la consommation de courant. Le principe de base consiste à éliminer, ou à réduire la source de fuite, en ajoutant du bruit pour rendre la corrélation avec le signal mesuré plus difficile, ou à insérer des opérations fictives, ou à implémenter un schéma de masquage. Ce dernier est considéré comme la contre-mesure la plus fiable, compte tenu de son efficacité théorique.

Ce problème implique plus de contraintes pour les concepteurs qui doivent, non seulement assurer le bon fonctionnement de l'application mais aussi, de garantir un niveau de sécurité contre certains types d'attaques. Habituellement, le niveau de sécurité est relatif au nombre d'observations nécessaires pour retrouver la clé secrète. Il est déterminé suite à une phase d'évaluation, réalisée notamment par un laboratoire de certification. Cependant, si ce processus est réitéré plusieurs fois, le coût de fabrication peut augmenter très rapidement. Il est donc nécessaire d'éliminer à un stade précoce de conception toutes les sources de vulnérabilités, c'est ce que nous appelons "évaluation pré-silicium".

Cette évaluation vise à vérifier l'absence de vulnérabilités, soit par une analyse horizontale, soit par une analyse verticale. Les fuites horizontales peuvent être détectées en comptant simplement le nombre de cycles. Pour les fuites verticales, une estimation de la consommation peut être effectuée à partir de l'activité du circuit, ainsi que de son état statique.

**Contributions.** Dans la première partie de cette thèse, nous avons mené une évaluation pré-silicium sur différentes implémentations matérielles (protégées et non-protégées). Cette étude vise à estimer le niveau de sécurité attendu sur une vraie cible (circuit réel). En effet, nous avons montré que grâce à une approche basée sur des simulations numériques, des fuites peuvent être détectées et caractérisées.Nous avons ensuite exploré un moyen plus efficace et plus exhaustif, permettant de tester une implémentation masquée contre les vulnérabilités induites par les changements transitoires des signaux (glitches). Nous avons profité dans cette approche pour modéliser au mieux ce phénomène, et expliciter la forme de la fuite engendrée. Ceci a permis d'expliquer l'inefficacité des modèles de fuites standards, et pourquoi seule une attaque par profilage permet de l'exploiter. En effet, d'un point de vue conception, il est difficile de déduire la variable servant comme masque pour un signal donné. Par conséquent, des failles peuvent être créées non seulement à cause d'un démasquage complet non-intentionné, mais aussi à cause d'une combinaison de signaux ayant une faible entropie. Ce type de vulnérabilité nécessite un nombre important de traces pour être exploité (même en simulation), ce qui peut empêcher sa détection quand le nombre d'observations est relativement faible.

Lorsque les phénomènes qui engendrent des fuites exploitables sont connus, une analyse formelle peut être adoptée pour les détecter de façon fiable, et ainsi offrir un moyen de les éviter. Néanmoins, cette analyse est limitée par le modèle adopté, et bride le spectre de fuites détectables. Il est donc nécessaire de procéder à une évaluation empirique à différents niveaux pour avoir plus de visibilité sur le comportement lié aux dispersions technologiques. Plus particulièrement, rien ne garantit que la consommation des différentes portes logiques

réalisant la même fonction booléenne soit identique. Des combinaisons de courant peuvent s'effectuer entre les différents blocs combinatoires, et ainsi engendrer des fuites critiques.

Dans la deuxième partie de cette thèse, nous avons exploré les attaques actives et présenté une étude sur les injections de fautes. L'implémentation cible est protégée par un schéma de codes correcteurs d'erreurs. Après avoir synthétisé le design avec différentes options, le taux de détection varie selon le critère d'optimisation. Comme toutes les contre-mesures basées sur la redondance, le synthétiseur peut éliminer l'ensemble, ou une partie du bloc de détection, ce qui rend le design vulnérable.

Dans le même contexte d'attaques actives, nous avons présenté une étude de sécurité au niveau "post-layout", contre les attaques en "micro-probing". Nous avons proposé une méthodologie complète et automatisable permettant d'évaluer une implémentation contre les attaques en micro-probing. Elle permet d'estimer le niveau de sécurité en analysant l'accessibilité des signaux sensibles par une station FIB.

## Chapitre 2 – Analyse du pre-silicium vers du post-silicium

Dans ce chapitre, nous avons mené une étude comparative de sécurité entre une cible virtuelle et une cible réelle face aux attaques physiques. Le but était d'étudier la disparité entre les mesures réelles et les mesures issues de la simulation, réalisées sur une implémentation identique d'un Advanced Encryption Standard (AES) matériel synthétisé sur un FPGA. Le nombre de traces nécessaires pour retrouver la clé secrète était relativement faible ($\leq 2K$). La même implémentation est analysée dans plusieurs conditions de bruit, et nous avons montré que le nombre de traces nécessaires pour retrouver la clé secrète peut être extrapolé à partir d'une mesure de référence.

Nous notons que cette relation est indépendante du modèle de fuite utilisé. Le seul paramètre pertinent est le niveau de bruit sur la nouvelle cible. Ce dernier ne dépend que de l'environnement de mesure et de l'équipement utilisé. Sans contre-mesures significatives, la difficulté d'une attaque sur une cible réelle est déterminée par le bruit environnant et le niveau de rayonnement électromagnétique de la puce. Les résultats sont illustrés avec une attaque par corrélation et la métrique de variance interclasses normalisée.

## Chapitre 3 – Analyse au niveau post-synthèse

Pour effectuer une analyse au niveau Post Synthesis (PS), nous avons synthétisé une implémentation AES sur un FPGA Xilinx. Le but était de refléter le comportement du circuit en prenant en compte les temps de propagation des portes logiques. Pour nous assurer que le synthétiseur

n'avait pas introduit de vulnérabilités suite à l'optimisation du design, nous avons effectué cette analyse en deux temps.

Dans la première expérience, nous avons retiré les informations de propagation de temps, et l'analyse n'avait pas détecté de signaux vulnérables. En effet, nous avons forcé le synthétiseur à conserver la hiérarchie du design, ainsi que les signaux internes de chaque module dans la description Register Transfer Level (RTL). Les résultats d'analyse sur cette netlist étaient donc similaires à ceux obtenus au niveau RTL. Dans la deuxième expérience, nous avons intégré les temps de propagation à la simulation, et l'analyse avait identifié plusieurs signaux vulnérables (une corrélation de 100%) avec le modèle de fuite prenant en compte la bonne hypothèse de clé. En particulier, l'entrée de la fonction $\ll$ SubByte $\gg$ a été démasquée (pour un bref délai) pendant le denier tour.

Pour corriger cette vulnérabilité, nous avons décidé de conserver le chiffré intermédiaire masqué un tour de plus, et de procéder au démasquage une fois le calcul terminé. Suite à ce correctif, l'analyse précédente n'avait pas détecté de vulnérabilité dans le design. En effet, nous avons séparé le signal de masque des tours intermédiaires, du signal de masque du dernier tour, et ainsi évité un démasquage probable (dû au retard) sur l'entré de la S-Box.

Pour rester rigoureux dans notre évaluation, nous avons entièrement supprimé cette phase de démasquage. Ainsi, le chiffré renvoyé est masqué. Nous pouvons donc supposer qu'aucun signal n'est démasqué en interne. En analysant chaque signal séparément, nous avons validé la conformité de la netlist avec le schéma de masquage. Cependant, nous ne pouvons pas affirmer ç ce stade que le design, dans son ensemble, ne présente pas de fuite du premier ordre.

Pour pousser l'analyse encore plus loin, nous avons généré des traces de consommations basées sur des simulations numériques. Ces traces sont construites à partir de l'activité du circuit (nombre de transitions) et de son état statique. En fait, cette combinaison permet d'examiner les deux types de fuites, liées soit à la valeur soit à l'activité des signaux. En effectuant une analyse du premier ordre, nous avons pu détecter des pics de corrélation entre les valeurs sensibles et les traces simulées. Une analyse par module nous a permis d'isoler la fonction vulnérable (Substitution Box (S-box)), et ainsi de restreindre l'étude uniquement à cette partie du design dans les analyses ultérieures.

## Chapitre 4 – Analyse formelle d'une implémentation masquée au niveau post-synthèse

Dans ce chapitre, nous avons étudié les fuites sur les schémas de masquage. Sur une description algorithmique, il est relativement simple de vérifier si chaque signal est masqué par

une variable aléatoire. Au niveau matériel, cette propriété doit être vraie non seulement à chaque cycle d'horloge (ou instruction), mais aussi lors des commutations des portes logiques (calculs effectués en combinatoire). Malheureusement, en raison des retards induits par les temps de propagation des portes combinatoires, des valeurs intermédiaires mélangeant les états antérieurs et courants des signaux peuvent être calculés (glitches). Ce phénomène induit ainsi des transitions extra-algorithmiques non contrôlées.

Plusieurs études ont montré que ces transitions peuvent dépendre de la valeur démasquée. Les protections proposées pour lutter contre ce type de défauts tentent soit d'éviter ces transitions et d'assurer qu'aucune autre fuite ne se produise, soit en séparant les portes combinatoires manipulant les masques et les données masquées, soit en rajoutant des barrières logiques (registres). Non seulement ces propositions sont assez contraignantes, mais la nature de la fuite ainsi que la raison exacte de son apparition n'est pas complètement expliquée.

Nous avons proposé une nouvelle approche moins abstraite, qui consiste à vérifier que toutes les configurations possibles, relatives aux retards, ne génèrent pas de fuites d'informations. Nous profitons dans cette approche pour valider la sécurité de quelques netlists masquées, optimisées ( en nombre de portes et en nombre de cycle) par rapport aux schémas de masquages résistants aux glitches, déjà présentés dans l'état de l'art. Nous présentons également des exemples de netlists plus petites ne respectant pas systématiquement les principes de conception résistants aux glitches couramment utilisés, mais nous essayons de masquer les transitions introduisant des fuites, uniquement sur les parties critiques du calcul.

Nous avons validé la sécurité de nos implémentations à l'aide de simulations logiques dans un premier temps, et sur des mesures réelles (des traces de rayonnement électromagnétique) dans un second temps. Nous avons également illustré la régression progressive de la fuite, suite à l'application des correctifs sur les parties du circuit identifiées comme vulnérables.

## Chapitre 5 – Analyse d'une implémentation protégée contre les injections de fautes

Les injections de fautes sont catégorisées parmi les attaques actives. Le but de l'attaquant consiste à introduire une erreur pendant une opération bien choisie. Des techniques d'injection moins contraignantes existent et elles sont généralement globales. On parle alors de perturbation de l'alimentation, de l'horloge ou de température. Pour des injections plus précises, nous pouvons citer par exemple les injections laser et électromagnétiques, qui nécessitent des équipements plus sophistiqués. Dans le contexte pré-silicium, il est plus facile de reproduire l'effet d'une faute sur un circuit. Cela permet de vérifier rapidement les protections contre ce type d'attaques, ainsi que l'aspect fonctionnel d'une contre-mesure.

Dans ce chapitre, nous avons étudié l'effet d'une injection de faute sur un bloc matériel AES-128, implémentant une contre mesure basée sur les codes correcteurs d'erreurs qui consiste à vérifier la parité de la matrice d'état. Pour vérifier que cette contre-mesure reste fonctionnelle aux différent niveaux d'abstractions, nous avons synthétisé cette implémentation sur un FPGA Xilinx Virtex-V au niveau PS et comme attendu, la détection était de 100%.

Dans un deuxième test, nous avons changé les options du synthétiseur, pour améliorer les performances de l'implémentation et optimiser les calculs combinatoires. En conséquence, toute la logique de vérification a été supprimée (elle est considérée comme un calcul redondant). Ceci a empêché la détection des fautes injectées, et a rendu l'implémentation vulnérable.

Finalement, dans un troisième test, nous avons changé les options de synthétise pour effectuer l'optimisation de façon incrémentale (donc en commençant par les chemins critiques). Dans ce cas, uniquement la partie dépendante du chemin de données a été supprimée et par conséquent, la détection était partielle.

Grâce à cette étude, nous avons montré que le processus de synthèse peut enlever complètement ou partiellement une contre-mesure. Ainsi, la vérification doit être effectuée à chaque niveau d'abstraction, pour éviter la propagation des vulnérabilités d'un niveau à l'autre.

## Chapitre 6 – Évaluation contre les attaques par sondage

Les attaques par sondage sont considérées comme les plus puissantes. Le but est de s'infiltrer à l'intérieur du circuit, créer des connexions avec les fils sensibles, et corréler les observations avec un modèle hypothétique pour extraire les données sensibles. Les protections contre ce genre d'attaques sont généralement basées sur l'insertion d'une couche de métal (shield) permettant de détecter les intrusions de façon active. Le masquage est également considéré comme un moyen algorithmique très efficace pour rendre ces attaques plus difficiles.

Dans cette étude, nous avons analysé l'efficacité d'un shield contre ce genre d'attaques, en prenant en compte les différents paramètres pertinents, à savoir le ratio du Focused Ion Beams (FIB), l'espacement et la largeur des fils qui composent le shield. Nous avons également proposé des pistes afin de renforcer une telle protection. Sur un exemple de circuit concret, nous avons pu montrer à travers une analyse pré-silicium, qu'une seule couche de shield n'apporte pas de protection significative. Nous avons ajouté (virtuellement) une deuxième couche de shield avec deux orientations différentes:

- Shield avec la même orientation, mais décalée par rapport au premier.
- Shield avec une orientation orthogonale par rapport au premier.

Pour comparer ces deux propositions, nous avons calculé la meilleure surface exposée dans les deux cas. Dans le premier cas, l'amélioration était négligeable, voire inexistante. La largeur ainsi que la longueur des surfaces exposées sont de l'ordre de $780\ nm$ et $15.8\ \mu m$ respectivement. En effet, la largeur est limitée par les caractéristiques du shield (dimension des fils et leurs espacement).

En revanche, la deuxième solution offre plus de protection. Les surfaces possédant une grande longueur au niveau $M7$ seront découpées lors de leurs projections au niveau $M8$ (de façon orthogonale). En fait, le diamètre des trous qu'on pourrait creuser à partir de $M8$, aura moins de $780\ nm$ au niveau $M7$. En conséquence, la profondeur maximale atteignable sera également limitée.

Grâce à cette procédure, nous pouvons déterminer les différents moyens permettant de sécuriser une implémentation donnée contre les attaques par sondage. Par exemple, le routage manuel des signaux trop exposés dans les premiers niveaux de métal, et la délocalisation de certains signaux (non-sensibles) dans les zones creuses donnant accès aux signaux sensibles, permet de limiter la surface exposée, et ainsi renforcer la sécurité du circuit.

# Acronyms

**AES** Advanced Encryption Standard. XIII, XXIII, XXIV, 1, 5, 7–9, 13, 15, 17, 18, 20–24, 26, 28, 32, 37, 41, 50, 53, 60, 62, 73, 80, 82, 90, 92–94, 100, 108, 109, 112, 117, 123

**CBC** Cipher-Block Chaining. 8

**CCA** Collision-Correlation Power Analysis. 17, 79

**CFB** Cipher Feedback. 8

**CPA** Correlation Power Analysis. XXIII–XXV, 12–14, 19, 31, 42–45, 51, 52, 55–60, 77, 79, 80, 122

**CRT** Chinese Reminder Theorem. 9

**DEF** Design Exchange Format. 100

**DES** Data Encryption Standard. 1, 7, 32, 37, 100

**DFA** Differential Fault Analysis. XXIII, XXIV, 31–34, 90–92

**DLP** Discrete Logarithm Problem. 1, 9

**DOM** Domain Oriented Masking. 23, 24, 28, 29, 69–71, 78, 85

**DPA** Differential Power Analysis. XXIII, 4, 10, 11, 15, 18–20, 24, 47, 121

**DRL** Dual-rail logic. 18

**EB** Electron Beam. 37

**ECB** Electronic Code Book. 8

**ECC** Elliptic Curve Cryptography. XXIII, 1, 8–10, 32

**EDC** Error Detecting Code. 93, 95

**EM** Electromagnetic Emanation. XXIII, XXIV, 2, 11, 14, 16, 20, 25, 32, 42–46, 57–60, 79, 80, 82, 85, 116, 118

**FIA** Fault Injection Analysis. 89, 98

**FIB** Focused Ion Beams. V, XVI, XXIV, 2, 5, 30, 34–37, 100–104, 106, 107, 109–111, 113, 116–118

**FPGA** Field Programmable Gates Array. 20, 43, 50, 52, 58, 79, 95

**GCD** Greatest Common Divisor. 32

**GE** Gate Equivalent. 78

**HD** Hamming Distance. 11, 12, 31, 43, 56, 58

**HO-DPA** High Order DPA. 19

**HW** Hamming Weight. XXIII, XXIV, 11–13, 15, 24, 45, 56, 58, 65, 80, 81

**IC** integrated circuit. 29, 30

**KSA** Kolmogorov-Smirnov Analysis. 12

**LEF** Library Exchange Format. 100

**LLVM** Low-Level Virtual Machine. 26

**LRA** Linear Regression Analysis. 12, 16, 31

**MIA** Mutual Information Analysis. 12

**NI** Non-Interference. 28, 29

**NICV** Normalized Inter-Class Variance. XXIV, 16, 17, 42, 56, 58, 60, 82, 84, 100, 102, 103, 109, 116, 125

**NIST** National Institute of Standards and Technology. 7

**NUEVA** Non-Uniform Error Value Analysis. 34, 90

**OTP** One-Time-Pad. 1

**PDF** Probability Density Function. 36

**PoE** Point of Event. 122

**PR** Place & Route. 3, 117

**PS** Post Synthesis. XIII, 3–5, 52, 53, 57, 71, 73, 77, 90, 117

**PUF** Physical Unclonable Function. 108

**RSA** Rivest Shamir Adleman. 1, 8–10, 20, 32, 37

**RTL** Register Transfer Level. XIV, XXIII, 3–5, 22, 23, 30, 43, 50–52, 73, 79, 90, 91, 95, 117

**S-box** Substitution Box. XIV, XXIII, XXIV, 13, 15, 17, 20–24, 26, 28, 31, 45, 50–53, 55–58, 60, 62, 73, 80–82, 84, 85, 90, 92–95, 100, 109, 110, 117, 123, 125

**SCA** Side-Channel Attack. 2, 3, 9, 14, 17, 21, 25, 42, 48, 49, 60, 85, 117

**SDF** Standard Delay Format. 79, 90

**SNI** Strong Non-Interference. 28, 29

**SNR** Signal to Noise Ratio. XXIII, 5, 13, 14, 16, 17, 19, 42, 43, 46–48, 116

**SPA** Simple Power Analysis. 4

**SR** Success Rate. XXIII, 13, 14, 16, 19, 42, 45

**TI** Threshold Implementation. 22–24, 27–29, 68, 71, 78, 85

**TINC** TI Non-completeness Property. 73, 76, 77, 123

**WT** Walsh Transform. 63, 65, 80

# List of Figures


2.1 AES block diagram for encryption and decryption. . . . . 8
2.2 Timing attack workflow . . . . . 9
2.3 Key recovery of an ECC double-and-add implementation (from [1]). . . . . 10
2.4 DPA workflow . . . . . 11
2.5 Power consumption in the HW model . . . . . 13
2.6 CPA based SR for different values of the SNR. . . . . 14
2.7 Confusion coefficients of the first bit of the AES S-box . . . . . 15
2.8 $2^{nd}$ order CPA based SR for SNR $= 0.5$. . . . . 19
2.9 Masked AND gate of [2]. . . . . 21
2.10 Canright masked inversion over $\mathrm{GF}_{256}$ . . . . . 21
2.11 Illustration of probing attack model . . . . . 25
2.12 Part of masked Keccak . . . . . 27
2.13 Propagation of labels for a small circuit . . . . . 28
2.14 Propagation of stable signals and transient signals . . . . . 28
2.15 DFA error distribution . . . . . 33
2.16 Probing a protected design with an active shield (from [3]). . . . . 35
2.17 Different components of the ion beam column. . . . . 36

3.1 Power traces. Left: virtual trace, right: EM measurement. . . . . 43
3.2 CPA using 10,000 traces Left: Virtual trace, Right: Real EM measurement . . . . 44
3.3 CPA convergence, (a) virtual trace, (b) real EM measurement . . . . . 45
3.4 SR convergence of virtual traces . . . . . 45
3.5 SR convergence of EM traces . . . . . 45
3.6 The mapping function between SNR and SR . . . . . 47

4.1 CPA result for an unprotected and protected implementation at RTL level. . . . . 51
4.2 CPA result on vulnerable signals . . . . . 52
4.3 Simplified block diagram of the masked AES . . . . . 53
4.4 Post synthesis simulation of vulnerable masked AES . . . . . 54

4.5 Illustration of signal unmasking . . . 54
4.6 CPA result on a vulnerable signal . . . 55
4.7 Superimposed NICV and a raw trace . . . 56
4.8 Result of the CPA using eq. 4.3 . . . 57
4.9 Raw trace superimposed with the NICV. Two leaking rounds are identified. . . . 58
4.10 CPA result based on eq. 4.3 . . . 59

5.1 Different ways to implement $f$ of example 2 . . . 67
5.2 Masked $AND$ gates . . . 69
5.3 Masked circuit computing $csb_1$. The leaking signals (red color) are $csa_1$ and $csb_1$. 74
5.4 First (a) and second (b) order CPA on $S_1$ of eq. 5.9 . . . 77
5.5 Our new design of one bit $GF_{2^4}$ inversion . . . 78
5.6 CPA on $csa_1$ activity. Only 75 traces are sufficient to recover the secret key . . . 80
5.7 WT applied to EM traces. $Basis = HW(W)$. . . . 80
5.8 Masked design of S-box'. . . . 81
5.9 Different implementations of the $GF_{16}$ inverter . . . 81
5.10 NICV based on an unmasked intermediate state of AES' . . . 82
5.11 Masked $GF_{2^4}$ inverter synthesis result . . . 83
5.12 NICV based on an unmasked intermediate state of AES . . . 84
5.13 AES S-box scheme using our secure $GF_{2^4}$ inverter . . . 84

6.1 Erroneous bits according to the glitch duration. . . . 91
6.2 Analysis of clock-glitch injection results using DFA AES-128 Giraud metric. . . . 91
6.3 Analysis of laser injection results using DFA AES-128 Giraud metric. . . . 92
6.4 Datapath of the AES parity check implementation against fault injection. . . . 94
6.5 Datapath of the AES S-box parity check against fault injection . . . 94
6.6 Extract from the XST synthesis options for Xilinx FPGAs. . . . 96
6.7 Total simplification of fault detection logic upon synthesis. . . . 96
6.8 Partial simplification of fault detection logic upon synthesis. . . . 97

7.1 Global workflow for probing evaluation threats . . . 102
7.2 First projection of a sensitive wire to the top layer. . . . 105
7.3 The projected area is crossed by one wire. It will be divided into small rectangles. 106
7.4 Cross-section of projected sensitive wire . . . 107
7.5 Illustration of the FIB model for milling. . . . 107
7.6 The circuit used for the evaluation . . . 108
7.7 Best area for milling . . . 110
7.8 $I$ score with different shield configuration. . . . 112

A.1 First (a) and second (b) order CPA on $csb_1$ . . . . . 122
A.2 Vulnerable circuit against glitches . . . . . 124
A.3 Secure version of circuit fig. A.2 . . . . . 125

# Chapter 1

# Introduction

Contents


1.1 Context . . . . . 1
1.2 Physical threats . . . . . 2
1.3 Protection . . . . . 2
1.4 Pre-silicon evaluation . . . . . 3
1.5 Objectives . . . . . 5


## 1.1 Context

Privacy has always been a milestone for centuries. Being able to communicate sensitive information play a fundamental role in society. Modern cryptography is the result of an evolutionary process across past generations of the art of text dissimulation. Starting from the Caesar cipher and then Vigenere cipher, more reliable and robust techniques were invented next. Properties were then formalised in order to ensure perfect secrecy. The first modern encryption algorithms are particularly Data Encryption Standard (DES), Triple-DES [4] and Advanced Encryption Standard (AES) [5]. They are classified under the category of symmetric block cipher using a secret key. There are other algorithms known as stream ciphers like RC5. The idea behind the latter is to approach the perfect (proven secure) One-Time-Pad (OTP) encryption.

However, when people want to exchange a secret key through an insecure network, they have to use other reliable means beforehand in a secure way. This is the role of asymmetric algorithms. The most currently used algorithms are Rivest Shamir Adleman (RSA) and Elliptic Curve Cryptography (ECC). The first one is based on the factorization problem of large numbers, while the second one is based on the Discrete Logarithm Problem (DLP). Theoretically and mathematically, these algorithms are considered to be safe and secure. The most

powerful and effective mathematical attacks known until today remain exponential, or at best sub-exponential in terms of the scalar size in bits. On the other hand, the implementations of these algorithms on embedded systems offer other angles of attack, either symmetric or asymmetric algorithms. Amongst these attacks we find Side-Channel Attack (SCA).

## 1.2 Physical threats

Physical attacks or SCAs exploit the flaws generated during the execution of a cryptographic program involving secret data. It can be divided into two classes:

- **Passive attacks**: they are non-invasive attacks that aim at observing and exploiting a physical property of the device when running some cryptographic operation. The physical property can be for instance the power consumption, Electromagnetic Emanation (EM), the computation time, the sound vibration or the thermal activity.
- **Active attacks**: they interact with the device by altering its behaviour. Such abnormal behaviour is obtained by tempering for instance with the clock or power supply of the system, or by injecting optical or EM pulses. Such analyses require sophisticated platforms and high skills to make the injection. A Focused Ion Beams (FIB) station can also be used for circuit editing, and accessing internal signals.

Timing vulnerabilities were the first to be exploited against asymmetric algorithms. Then, they have been extended to any type of implementation that shows variation in execution time. The reason of the leakage may vary depending on the type of implementation or the runtime environment. When the temporal variations depend on a sensitive value, a timing attack becomes possible. Regarding power consumption or electromagnetic emanation, the range of attacks is much wider. From a high-level point of view, those attacks can be divided into two categories: horizontal attacks and vertical attacks. Horizontal attacks exploit one or few traces to break the secret key, by exploiting either local and temporal information, or the pattern of power consumption according to an intermediate sensitive value. On the other hand, vertical attacks exploit the variation linked to the intermediate manipulated data. Several traces of power consumption are necessary to carry out an attack and be able to recover the secret key.

## 1.3 Protection

To protect sensitive data against such attacks, it is required to implement more secure and robust variants, depending on different angles of attack that are offered to a potential attacker. For remote targets, it is sufficient to protect against timing attacks. For targets that can be accessed physically (such as smart-cards), an additional layer of protection must be added to

also cover horizontal and vertical attacks, besides of timing attacks. Several ways are available to protect our application. The overall idea is to eliminate and reduce the source of exploitable leakages, by adding (independent) noise to make the correlation with the signal more difficult, insertion of dummy operations, randomizing the operations or implementing a masking scheme. The latter is the most studied countermeasure, given its theoretical efficiency.

This implies more constraints for designers, who must not only ensure the proper functioning of the device but also provide a security level against some kind of attacks. Usually, the security level is based on the number of observations necessary to find the secret key. This level is determined after an evaluation phase by a certification laboratory. On the other hand, if this process is repeated several times, the manufacturing cost increases very quickly. For this reason, a designer wants to eliminate at an early stage of conception, the source of vulnerabilities as much as possible. This what we call pre-silicon evaluation.

## 1.4 Pre-silicon evaluation

To evaluate a device at an early stage of conception and avoid a waste of time and money, it is necessary to have very effective evaluation tools. The evaluation should check that:

1. The protection specification is well respected and the countermeasures are well implemented;
2. No vulnerability is observed according to a given number of observations.

The first point is more or less obvious to be respected by an experienced designer, and can be verified by digital simulations. On the other hand, the second point is more difficult to guarantee. Other leakages can arise due to an imperfection of the circuit modelling, which may lead to a significant difference between the expected leakage and the one observable on real targets.

### 1.4.1 Empirical evaluation

The best assessment of an SCA leakage is possible by a better modelling of the hardware design. In the context of integrated circuits, these modelling could be considered as the different abstraction level of the corresponding target design, namely: Register Transfer Level (RTL), Post Synthesis (PS), Place & Route (PR) and post-layout. To advance towards a real evaluation, the power consumption traces can be estimated using either digital or electrical simulation. For digital simulations, all levels of abstraction can be considered (RTL $\rightarrow$ PR), to carry out an exhaustive analysis. An example of a such progressive proceeding is outlined in chapter 3 and 4.

The first step at RTL level could detect any algorithmic and coding leakages, and allows designers to eliminate them at an early stage.

This evaluation includes the verification of all signals, and detects several types of vulnerabilities, either by horizontal or vertical analyses. Horizontal leakages can be detected by simply counting the number of cycles for each operation. It allows among other things to detect timing (local and global) and Simple Power Analysis (SPA) leakages. For vertical leakages, we can estimate the activity of the circuit based on the switching signals and their static states. We recall that by design, a CMOS gate consumes only when a change occurs on one of its inputs, which validates the toggle count model. These traces could be used to perform statistical analyses, like Differential Power Analysis (DPA) to detect vertical leakages.

To get closer to a real circuit, a timing-annotation PS netlist can be considered instead. In fact, new leakages could be identified at this stage. Combinatorial calculations mixing previous and current values of signals can be carried out by the same gate, which can generate an extra flaw.

At digital level, the estimated power consumption can be improved by considering each gate separately, and by exploiting the information provided or extracted from an electrical simulation. As a function of the input values, the power consumption of a gate is taken from a pre-defined table. This gives a more precise estimation of the power consumption of the circuit.

### 1.4.2 Formal evaluation

It is worthy to consider that a simulation does not cover all possible instances of a given implementation. Depending on the adopted technology, behaviours favouring or preventing leakage may occur, namely the ones generated because of propagation time. We can therefore use stronger means and properties to verify the security criteria. A formal approach aiming to model the impact of known physical phenomena may be more effective, and more reassuring for a designer ignoring the final technology.

These kinds of evaluations are based on more or less strong security properties. In particular, when checking a masking scheme, the used model is generally based on probing attack. This model allows an attacker to place probes on the internal signals of the circuit. If the secret cannot be reconstructed from the values observed with these probe, thus the circuit is considered to be secure. When the masking is based on $d$ shares, we speak about masking at order $d$, and security at $d$.

Indeed, although some schemes are proved secure at the algorithmic level, a first order leakage is identified on synthesised and time-annotated netlists. This leakage is exploitable on real targets, and can only be observed when taking into account the propagation time in the logic gates for instance.

## 1.5 Objectives

The main objective of the thesis is to improve pre-silicon evaluation methods. A better characterization of the leakages that can be exploited by an attacker, allows us to estimate the expected level of security before manufacturing the final circuit. For this purpose, we conduct an end-to-end comparison between virtual and real analyses.

In the first part, we focus on the characterization of the side-channel leakage at pre-silicon and post-silicon levels (RTL and PS), based on the same unprotected hardware designs, and by considering different Signal to Noise Ratio (SNR) levels. In the same way, we show an evaluation of a protected implementation, which aims to identify the different sources of non-obvious leakages. This may be present at the design level, namely those caused by the propagation times and glitches. We thus, propose a method to study this last phenomenon, and the different existing ways that allows us to prevent such vulnerabilities, by relying on thorough characterization and a formal evaluation.

In the second part, we focus on active attacks, namely, fault injection and micro-probing attack. For fault injection, we have implemented a compact protected version of AES, as presented in the state of the art. We have studied which impact the synthesis could have on such an implementation, which presents a certain computational redundancy to guarantee the data integrity.

Finally, we present an end-to-end methodology allowing to quantify the difficulty of a probing attack using a FIB. To estimate the security level, we take into account the layout of the design and the performance of the FIB.

# Chapter 2

# Related works

Contents


2.1 Modern cryptography . . . . . 7
2.2 Physical attacks . . . . . 9
2.3 Countermeasure against physical attacks . . . . . 17
2.4 High-order DPA . . . . . 24
2.5 Evaluation methods . . . . . 25
2.6 SCA & performance issue . . . . . 31
2.7 Fault analyses . . . . . 31
2.8 FIB for probing . . . . . 34


## 2.1 Modern cryptography

### 2.1.1 Symmetric cryptography

To encrypt sensitive data or a communication between two entities, the AES algorithm is the most widely used in the world [5]. It was designed in 1997 during the National Institute of Standards and Technology (NIST) competition, to standardize an alternative algorithm to DES, and Triple-DES.

AES is based on three basic functions, which are executed a given number of time $N$ (10, 12 or 14) depending on the key size, as presented in fig. 2.1. At each round, the state (presented as a $4 \times 4$-byte matrix) is updated with the following sub-functions:

- $AddRoundKey$: The state is xored with a secret key-round derived from the master key;
- $SubBytes$: It is a bijective byte substitution function, applied to each byte of the state.
- $ShiftRows$: The row $i$ is rotated by $i$ position(s) to the left;

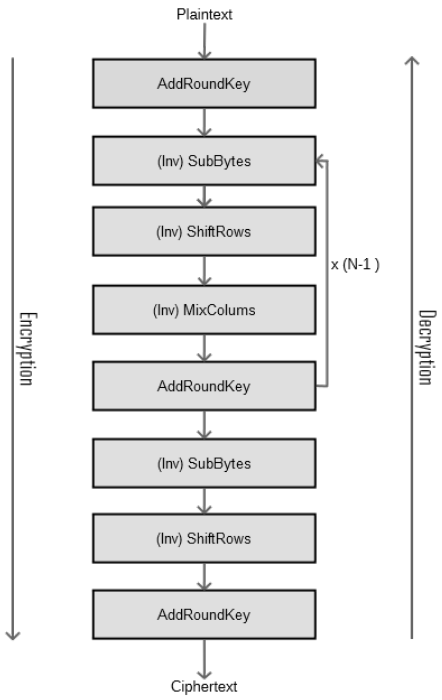



Figure 2.1: AES block diagram for encryption and decryption.

- $MixColumns$: The state matrix is multiplied by a constant matrix, where each byte is considered as an element of the Galois field $\mathrm{GF}_{256}$.

The different round keys are derived from the master key, using a specific process based on permutation and the $SubBytes$ function. Moreover, $SubBytes$ is used to ensure the confusion property, $ShiftRows$ and $MixColumns$ are used to ensure the diffusion property. Both properties are fundamental for a symmetric encryption algorithm to be secure [6]. The decryption process is performed using the inverse of each sub-function in the reverse order. To encrypt a long stream of data, AES is applied to each block of 128 bits using a specific chaining mode, such as: Electronic Code Book (ECB), Cipher-Block Chaining (CBC), Cipher Feedback (CFB) and so on.

### 2.1.2 Asymmetric cryptography

To ensure a key exchange between two remote entities communicating trough a non-trusted channel, it is necessary to use a protocol based on asymmetric algorithms. These algorithms work with two keys. A private key kept secret that is used to decrypt data, and a public key used by the other users, who want to communicate with the owner of the private key. The two algorithms used in the current applications are RSA and ECC.

RSA is based on the big number factorisation problem. By choosing two prime numbers $p$ and $q$ ($\geq 2048$ bits), two keys are constructed as follows:

- $(N, e)$ (public) with $N = pq$ and $e$ chosen small (ex: 17 or 65537);
- $(p, q, d)$ (private) such that $d \times e = 1 \bmod (p-1)(q-1)$.

To encrypt a message $m$, the sender computes $c = m^e \bmod N$. To decrypt $c$, the receiver computes $m = c^d \bmod N$. Besides, in the case of a signature, RSA can be seen as a DLP (knowing the signature $s = m^d$ of a known message $m$, find $d$). Thus, key exchange protocols like Diffie-Hellman can also be used [7]. Key exchange and digital signature can also be designed based on ECC [8].

## 2.2 Physical attacks

### 2.2.1 Timing attack

Timing attack is the object of the first known SCA in the state of the art [9]. The exploit involves a basic implementation of RSA. The overall execution time of a modular exponentiation depends on the key and on the input message.

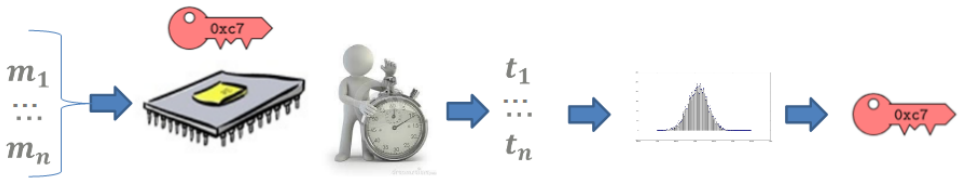



Figure 2.2: Principle of timing attack. The timing distribution is computed for each key hypothesis.

A measurement of the execution time variation allows an attacker to find the value of the secret key recursively (fig. 2.2). Applications using RSA usually implement an alternative version based on Chinese Reminder Theorem (CRT). Thus, the modular exponentiation can be speeded-up by a factor of four ($\times 4$). Attacks targeting this version are also presented in [10, 11].

A cryptographic implementation is vulnerable to a timing attack when variations in the execution time depend on sensitive data. These variations can be a consequence of either the implementation or the hardware behaviour. More commonly, this vulnerability is present at algorithmic description level. It is characterised by a non-constant time functions or instructions, such as big-number multiplication, modular inversion or different processing based on sensitive values, such as conditional branching. The latter can be qualified as micro-architectural vulnerability. It is the cause of the latency when loading data, either from the main memory or from the cache memory. A timing attack targeting an AES implementation is presented in [12]. The

targeted implementation is in fact constant time from algorithmic view point , but the execution time variations are due to the cache access latency, which varies according to whether the data are present or not in the cache memory.

### 2.2.2 Horizontal attack

Horizontal attacks exploit local characteristics of the side-channel trace, based either on temporal or vertical information. One of the simplest attack is the one targeting a non-regular asymmetric implementation of RSA (square-and-multiply) or an ECC (double-and-add) [13]. We recall that these implementations scan the scalar (on binary form) and perform one or two operations depending on the value of the current bit (one operation for '0', two operations for '1'). On a such implementation, one trace is enough to recover the exponent or the secret scalar (see fig. 2.3). When the current bit is '$0$', only the square operation is executed, but when it is equal to '$1$', both square and multiplication operations are executed.

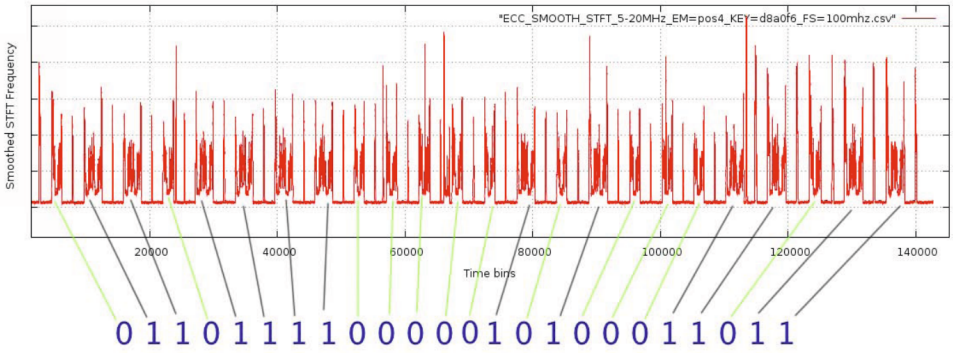



Figure 2.3: Key recovery of an ECC double-and-add implementation (from [1]).

Besides, on asymmetric implementations, dealing with large integers induces many constraints when trying to make the execution of each elementary operation completely constant-time. Thus, even on regular versions (like Montgomery ladder), some attacks exist. These attacks characterise the pattern of the power consumption according to a given deterministic or probabilistic criterion. They aim for example at characterising the number of (extra)-modular reductions [14], or characterising the modular multiplication of the intermediate hypothetical value, such as doubling-attack [15] or big-mac attack [16].

### 2.2.3 Differential power attack

DPA is the most addressed and studied in the state-of-the-art existing attacks. It exploits the variation of the power consumption in terms of the processed data at a given time [17, 18]. Indeed, an electronic component does not consume the same amount of energy when performing the same operation on different data. When sensitive data are processed, the power

consumption may give an image of the sensitive information to an external attacker. Based on some assumptions, the attacker can extract the secret key of a cryptographic implementation.

We should also note that the power consumption of an electronic device depends on the number of transitions inside the logic gates. Thus, DPA can be mounted by making some hypotheses about the processed data, that involve the secret key [19].

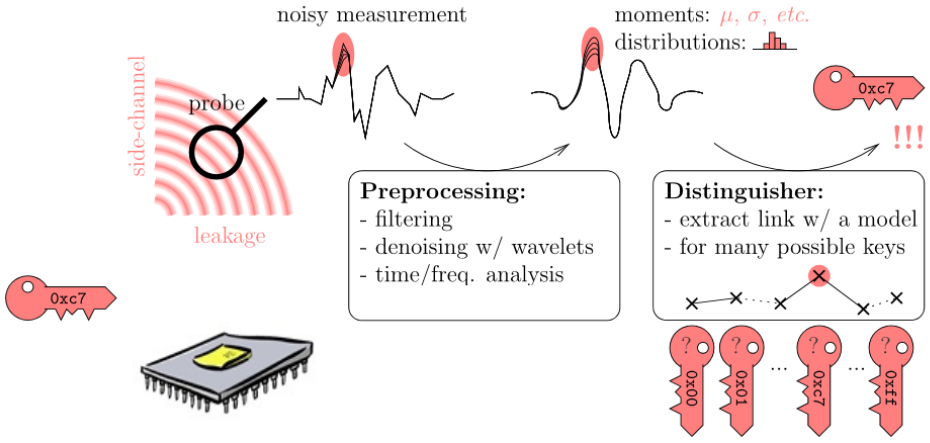



Figure 2.4: All steps for a DPA. The traces can be filtered in presence of noise before performing the attack.

The global workflow of a DPA is presented in fig. 2.4. The attacker starts by acquiring traces of power consumption or EM emanation on the target device. After preforming some filtering and denoising steps, a statistical analysis is applied to extract the secret key. There are two important concepts at this step: The leakage model and the distinguisher, which we detail below.

#### 2.2.3.1 Distinguisher & leakage model

When a device processes a data $X$, the power consumption $P$ can be modelled as a combination of a deterministic component, function of the manipulated value $\varphi(X)$, and a random component which models an independent noise $N$:

$$P = \varphi(X) + N \tag{2.1}$$

**Definition 1** (Leakage model)**.** A leakage model is a theoretical prediction function that estimates an equivalent image of the power consumption given an intermediate value.

The most known and used leakage models in the state-of-the-art existing attacks are generally based on the Hamming Weight (HW), Hamming Distance (HD), mono-bit and multi-bit

models. Those leakage models can be generalised by:

$$\varphi(x) = \sum_{i=0}^{n-1} \omega_i x_i \tag{2.2}$$

where $\omega_i \in [0, 1]$, and $x = \sum_{i=0}^{n-1} 2^i x_i$ is either an intermediate value or the result of some combined intermediate data (HD case). More sophisticated leakage models can be built with a posterior knowledge of the target. This function involves the value of the secret key. To distinguish the right key from the wrong ones, we use a statistical metric which is known as a distinguisher.

**Definition 2** (Distinguisher)**.** A distinguisher $\mathcal{D}$ is a statistical metric that allows distinguishing the secret key using the observations. We note also $\hat{\mathcal{D}}$ an empirical estimator of $\mathcal{D}$ based on $n$ observations.

According to definition 2, the secret key can be extracted by maximising the distinguisher value over the set of key hypotheses:

$$k^* = \underset{k}{\operatorname{argmax}}(\hat{\mathcal{D}}(k))$$

which gives the most probable key hypothesis.

The distinguisher measures the level of similarity between the leakage model (specific to a target node and characterised by a key hypothesis) and the side-channel leakage. There are several more or less efficient distinguishers depending on the situation (noise level, number of traces, etc.).

One of the most powerful distinguishers is the correlation, and known as Correlation Power Analysis (CPA) [19]. CPA is very efficient when the side-channel trace is linear according to the leakage model and the noise is Gaussian [20]. Linear Regression Analysis (LRA) is the generalized version of the correlation in the multi-dimensional case, when considering each bit separately. When the nature of the leakage is not usual (such as HW), other distinguishers can be used, such as Mutual Information Analysis (MIA) [21, 22] or Kolmogorov-Smirnov Analysis (KSA) [23]. They compare the distributions between the leakage model and the side-channel traces, without making any assumption about their forms.

An extensive comparison between these distinguishers is presented in [24]. They showed in which case MIA takes advantage from CPA (when the leakage model diverges from the practical measurements), and the different factors that influence the success of the attack, such as noise, and the nature of the leakage signal.

We show in fig. 2.5 two curves representing the average of the power consumption for two different HW of the processed value. In this case, the average allows an attacker to distinguish

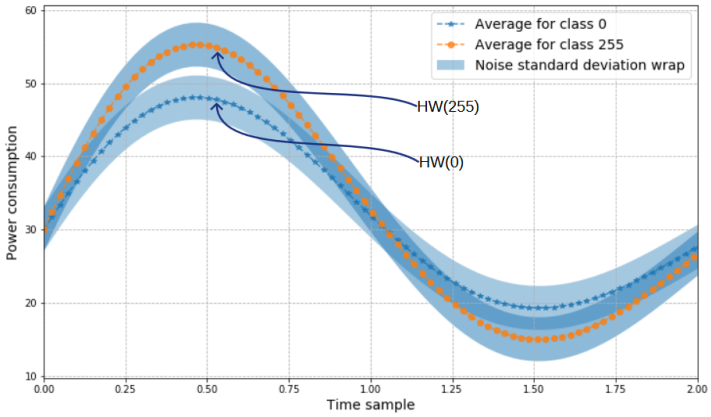



Figure 2.5: Power consumption in the HW model. The different classes can be distinguished with the average of consumption for each class.

the different classes and thus, by exploiting this variation (inter-class) she can extract the associated sensitive value. We can also see that, if the noise envelope is larger (the standard deviation of the noise), it becomes more difficult to separate each class, and therefore the attack becomes more difficult and will require more traces. The higher the noise is, the more difficult the attack is.

A relation between these two parameters can be established using the SNR.

**Definition 3** (Signal to Noise Ratio)**.** The SNR is defined as the signal ($S$) variance divided by the noise ($N$) variance:

$$SNR = \frac{\mathbb{V}[S]}{\mathbb{V}[N]}$$

To illustrate this relationship, we simulate traces based on eq. (2.2), using random messages $m$ and a fixed (secret) key $k^*$. The signal part ($\varphi$) is calculated as the HW of the AES Substitution Box (S-box) output, and the noise $N$ follows a centered Gaussian distribution. Thus, we have:

$$P = HW(\text{S-Box}(k^* \oplus m)) + N$$

To measure the efficiency of a given analysis, we use the SR metric.

**Definition 4** (Success Rate)**.** The SR for a given number of observations $n$, relative to a distinguisher $\mathcal{D}$ is defined as:

$$SR(n) = \mathbb{P}(\hat{\mathcal{D}}_n(k^*) > \hat{\mathcal{D}}_n(k)_{k \neq k^*})$$

where $\hat{\mathcal{D}}_n$ is the estimated value of the distinguisher based on $n$ observations. This metric measures the probability of finding the secret key with a given number of traces.

For different values of the SNR, we plot in fig. 2.6, the curves of the SR, based on the result of CPA. To estimate theoretically the SR, several proposals are already being considered in the

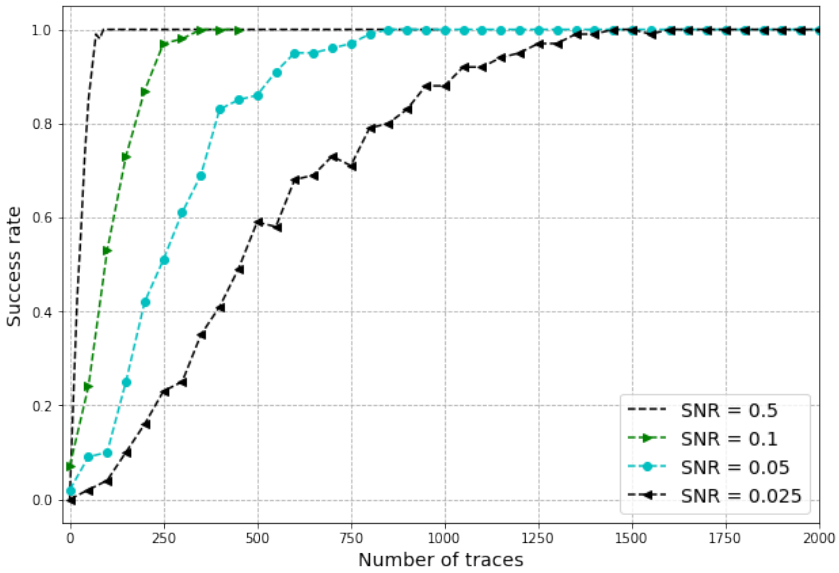
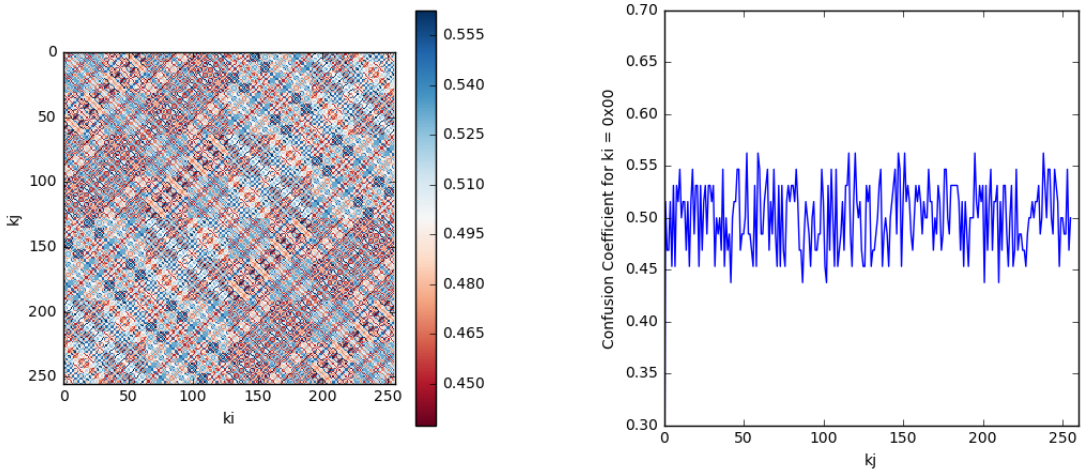



Figure 2.6: CPA based Success Rate (SR) for different values of the SNR.

state of the art. The first one is described in [25]. Subsequently, other models are proposed, such as [26, 27, 28]. In [26], Rivain has established a way to approximate the SR of a first order SCA, by analysing the distribution of the distinguisher scores. For this purpose, he defined a comparison vector ($C_k$), and the distinguisher is evaluated for the right key and the wrong keys ($C_k = \mathcal{D}(k^*) - \mathcal{D}(k)$). When a Gaussian leakage model is assumed, the comparison vector follows a multi-variate Gaussian distribution, which allows us to estimate the SR of the attack. He validates this metric on simulated traces, on both CPA and profiled attack. In [28], Lomné et al. extended this approach to high-order SCA on masked implementation. With the same method, they defined the SR of a CPA and profiled SCA, according to the multiplicative combining technique of samples. They validate their estimations on simulated traces and EM traces up to order four ($4^{th}$O-SCA). This methodology is also described in [29]. In [30], the authors made an in-depth study of these different approaches, and exposed a good comparison between these different estimators.

In practice, the SR depends on the performance of the considered distinguisher. The latter, in turn, depends on the leakage model used. The SR can be estimated by repeating the attack several times for a fixed number of observations. For the same signal quality and the same leakage model, it may vary depending on the used distinguisher. An in-depth study of diverse distinguishers is presented in [31]. They also derived an SR metric based on a success exponent.

From fig. 2.6, we can notice that the required number of traces to recover the secret key is roughly inversely proportional to the SNR. To estimate the SR of CPA, we have repeated the attack $100$ times with different traces.




(a) (b)

Figure 2.7: Confusion coefficients of the first bit of the AES S-box. (a) Confusion matrix for all keys. (b) Confusion coefficient for the key 0x00.

In [27], the authors have established a success rate estimation metric as a multi-variate Gaussian distribution, which takes into account different parameters involved in a DPA, namely the quality of the signal and the targeted function properties. The latter is known as the confusion coefficient:

**Definition 5** (Confusion Coefficient)**.** The confusion coefficient $\kappa$ of a function $f : k \mapsto f(k)$ is defined as: $\kappa = \mathbb{P}(f(k_i) \neq f(k_j))$. It is the probability that two random (different) keys give different outputs.

In an equivalent way, the *collision coefficient* $\xi$ which is the complementary of the confusion coefficient can be defined as: $\xi = 1 - \kappa$.

If we consider the AES S-box as a target function, the confusion coefficients of the first bit output is plotted in fig. 2.7. The result for all possible keys is shown in the confusion matrix (see fig. 2.7a). Each pixel of the image gives the portion of messages that leads to the same output bit for two keys $k_i$ and $k_j$. We intentionally set the diagonal to $50\%$ (instead of $0\%$) for more clarity. We can easily identify the symmetry of the matrix following the diagonal. In fig. 2.7b, we plotted the first line of the matrix, which corresponds to the key $0x00$. The minimum value is $0.4375$, which means that at least one wrong key gives 56% of collisions. In the same way, we can compute the confusion coefficient for any output bit, or a specific processing on the output of the targeted function, such as the HW [27, 30].

Another metric that can be used also for the same purpose is the rank filter [32]. It gives the rank of the specified key after processing $n$ traces. The attack is successful when the right key

is ranked first.

These concepts are reused in chapter 3 to demonstrate the sense and the value of a pre-silicon evaluation towards an EM one. To this end, we study the convergence of different metrics, such as Normalized Inter-Class Variance (NICV) and SR in different scenarios, with different value of SNR.

#### 2.2.3.2 Profiling based analysis

In some cases, the leakage model is very difficult to predict. As we have seen previously, we cannot exclude the case where each bit of the processed value does not consume the same amount of energy, so the weights $\omega_i$ are difficult to estimate directly. A regression-based analysis (LRA) could be used to avoid such unforeseen issues. On the other hand, when the side-channel leakage is not of the same form as eq. (2.2) (for example a combination of a $xor$ of some bits), it becomes more difficult to predict. Profiling-based attacks are a very effective way to overcome these kinds of constraints. A more general version of this type of analysis is known as Template attacks [33, 34]. The idea is to characterise the leakage on a clone device for different key hypotheses, and use this database to attack the target device. Generally, the leakage is characterised by its average and its co-variance matrix using a multi-dimensional Gaussian distribution. The distinguisher is then based on the maximum likelihood. This analysis can be divided in two main stages:

1. Profiling phase:
   - the attacker collects a large number of leakage traces on a clone device with different (known) keys $k \in \mathcal{K}$;
   - for each key hypothesis $k$, the attacker computes the average $M_k$ and the co-variance matrix $C_k$ of $n$ points of interest.
2. Extraction phase:
   - Using one or few traces $\{T_i\}$ from the target device, the attacker computes the most likely class from the built templates, based on the $n$-dimensional multi-variate Gaussian distribution.

$$Pr(T_i) = \frac{1}{\sqrt{2\pi^{\frac{n}{2}}|C_k|}} exp(-\frac{1}{2}(T_i - M_k)' C_k^{-1}(T_i - M_k)).$$

   - The extraction of the most probable key $k^*$ may be achieved using the maximum likelihood:

$$k^* = \underset{k \in \mathcal{K}}{\operatorname{argmax}} \left\{ \prod_i Pr(k|T_i) \right\}$$

Another simpler variant of template analysis is based only on the average of the side-channel trace, and it does not require a clone device. It is enough to characterise the leakage on a small part of a redundant computation (for example only one S-box of the AES), and to attack the rest of the computations using the correlation as a distinguisher for example [35, 36, 37]. This analysis is known as Collision-Correlation Power Analysis (CCA).

As Prouff et al. have demonstrated in [38], when the leakage model is equal to the conditional mean according to the sensitive value ($\mathbb{E}[L|Z = z]$), the correlation is optimal under the Gaussian assumption. This last finding was used in [39, 40] as a leakage detection metric known as NICV.

In [41], Oswald and Mangard presented different Template attacks on masked software implementations. They showed that when taking the mask value into account in the profiling stage, the attack is more efficient. Only fifteen $(15)$ traces are required to recover the right secret key.

In chapter 4 and chapter 5, we use such sophisticated distinguishers and techniques to characterise and exploit the SCA leakage. We also compare their effectiveness against a standard attack; using a distinguisher combined with a leakage model.

## 2.3 Countermeasure against physical attacks

There are many ways to protect cryptographic implementations against SCA. The main idea is to make the observations uncorrelated from the sensitive data. Adding some (uncorrelated) noise will also help to reinforce the countermeasure. The lower the signal quality is, the more difficult the attack is. In the following, we present some countermeasures that have been most discussed in the state of the art. In general, there are purely algorithmic versions, which induce a partial or a complete modification and re-designing of the implementation such as masking and blinding. Other countermeasures are based on empirical techniques aiming at re-ordering the computations, by randomly permuting some instructions or by inserting fake operations.

### 2.3.1 Hiding

Hiding countermeasure consists generally of randomizing the internal operations of the algorithm, when the order of execution does not matter, as suggested in [42, 43]. In the case of symmetric algorithms, the execution of the different steps can be exchanged. Moreover, the processing of each sub-data of the current state can be done in a completely random manner. This allows to reduce the SNR and thus, makes the attack more difficult. In the case of AES block encryption, randomization can be applied at function level between $SubBytes$ and $ShiftRows$, and internal state level by randomizing the processing of each byte for $AddRoundKey$, $SubBytes$,

$ShiftRows$, and at column level for $MixColumns$. This countermeasure is generally combined with more powerful ones such as masking (see section 2.3.2).

For Hardware implementations, the logic gates can be re-designed to make the power consumption independent from the processed data. It is based on Dual-rail logic (DRL) gates [44, 45], which consist on a complementary logic that switches only once whatever the performed computation. This countermeasure can also be combined with algorithmic ones (like masking) to increase the resistance [46]. However, a significant area overhead is quickly reached.

### 2.3.2 Masking

The most common countermeasure discussed in the state of the art is masking [47, 48, 49]. It aims at protecting the cryptographic implementation against vertical attacks. As a reminder, vertical attacks assume constant consumption for the same input data. It therefore, becomes obvious that if the inputs of the operations targeted by a DPA are random, the leakage model will no longer match the physical leakage. The principle of masking consists in dividing the secret into several shares, and performing the equivalent calculation by manipulating only the shares. This "sharing" depends on the structure of the algorithm. In an algebraic Boolean structure, a Boolean masking is preferred, in a multiplicative group, a multiplicative masking ($aka$ blinding) is used.

Boolean masking of an AES implementation is considered in chapter 4 and chapter 5, to evaluate the robustness of such countermeasure, but also to identify other flaws due to either a mis-integration or glitches.

#### 2.3.2.1 Boolean masking

An intermediate secret data $X$ can be written as:

$$X = \bigoplus_{i=1}^{n} X_i \tag{2.3}$$

and each share $X_i$ is used by a function $\varphi_i$. According to the hypothesis of eq. (2.1), the power consumption $P$ of the device becomes:

$$P = \sum_{i=1}^{n} \varphi_i(X_i) + N$$

As a result, the correlation in the broadest sense cannot be established between $P$ and $\varphi(X)$. The level of protection $d$ is related to the number of used shares, and it is generally lower ($d \leq n$).

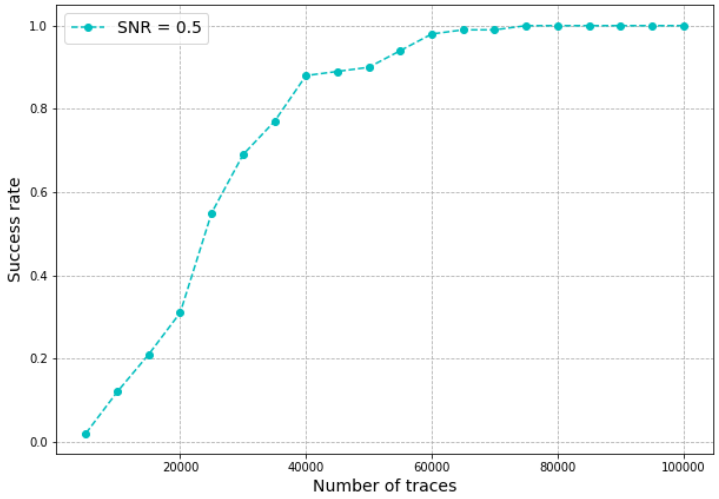



Figure 2.8: $2^{nd}$ order CPA based SR for SNR $= 0.5$.

To break a protected implementation with a masking at order $d$, we need to use High Order DPA (HO-DPA). Therefore, a first order masking ($d = 1$) can be broken using a second order DPA, but it requires more observations (exponential with the order) to recover the secret key compared with an unprotected one [47, 50]. In the particular case, when the masked data and the mask are manipulated at the same time (case of parallel hardware implementation for example), the second order analysis can be performed using the second order moment of the traces:

$$\begin{aligned} P^2 &= \varphi_1(X_1)^2 + \varphi_2(X_2)^2 + N^2 \\ &+ 2 \times \varphi_1(X_1) \times N + 2 \times \varphi_2(X_2) \times N \\ &+ 2 \times \varphi_1(X_1) \times \varphi(X_2) \end{aligned}$$

The relevant term is $2 * \varphi_1(X_1) * \varphi_2(X_2)$, which combines both shares of the secret (multiplicative combining). All other terms can be considered as noise, because they cannot be predicted by the attacker.

To get an idea of the advantage of masking, we simulated traces in the same way as for the unprotected case. The number of required traces to recover the key has greatly increased. We deduce a factor of $640$ between the non-masked (fig. 2.6) and the masked version (fig. 2.8) when the SNR is equal to $0.5 = \frac{\mathbb{V}[2 \times \varphi_1(X_1) \times \varphi_2(X_2)]}{V[N]}$ in both case.

#### 2.3.2.2 Multiplicative masking

Multiplicative masking is generally used in a multiplicative group, such as the case of RSA. We recall that RSA is based on a private key $(d, p, q)$, and a public key $(N = pq, e)$, with $e \times d = 1 \; mod \; \phi(N)$, and $\phi$ is the Euler function. It is much easier to use multiplicative masking in this kind of structure. For example, in the case of a modular exponentiation ($m^d \; mod \; N$), the input message $m$ can be multiplied by a random $r$ ($m' = m \times r$). This makes the intermediate values impossible to predict by an external attacker. The unmasking can be done by multiplying the final result by $r^e$.

Some proposals have also been suggested to mask the AES S-box. As it is composed of two parts, the first being an inversion in the field $\mathbb{F}_{256}$, the second is an affine transformation. Therefore, in the inversion stage, it is possible to multiply the input by a random value $r \in \mathbb{F}_{256}$, perform the inversion which will be masked by $r^{-1}$ and then transform into Boolean masking for the affine part, as described in [51]. Unlike RSA, multiplicative masking of the AES S-box is vulnerable to a first-order DPA as mentioned by the authors in [51] and in [52]. Indeed, the value $0$ is never masked, and therefore allows an attacker to distinguish the zero-input value of the S-box, and finally recover the secret key. To bypass this problem, several suggestions have been discussed and aims to replace the zero-input value with another one [53].

Recently in [54], the authors described a way to deal with the zero-input problem. They use the fact that the zero-input value and the unit-input value are their own inverse. Thus, they replace the zero-input value by one, and compute a $\delta$ function in a shared way, which is used to patch the final result. The value of $\delta$ is added in the conversion step, from Boolean to multiplicative masking. They also give experiment results based on $200$ millions simulated traces, and $50$ millions EM traces acquired from a Field Programmable Gates Array (FPGA). This implementation should also (and designed to) prevent the problem related to glitches encountered in a pure Boolean masking schemes.

### 2.3.3 Boolean masking in presence of glitches

The problem related to glitches is mainly critical when implementing and designing non-linear functions. Indeed, linear functions only need to process the different shares independently to perform the equivalent computation. On the other hand, a non-linear realization (masked $AND$ gate for example) must combine several shares of the same variable. In the case of the multiplier of [2], the two shares ($a$ and $m$) of the same variable are joined on the same $XOR$ gate that computes $i_3$ (see fig. 2.9).

As [55] has mentioned, the number of transitions at this gate is correlated to the sensitive value ($x = a \oplus m$).

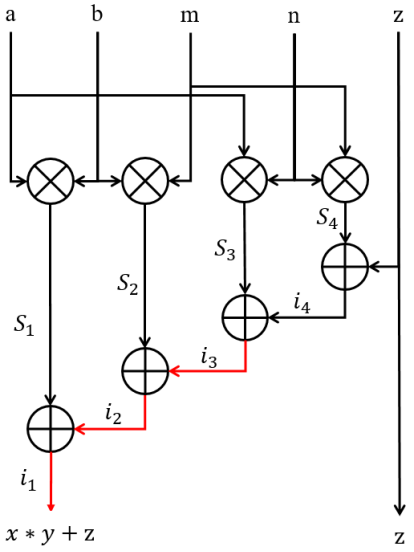



Figure 2.9: Masked AND gate of [2].

#### 2.3.3.1 Canright masked S-box

Canright proposed in [56] a very compact implementation of the AES S-box. The proposed implementation uses the sub-field of $\mathrm{GF}_{2^8}$ (also called "tower field") for the S-box computation, as previously presented in [57]. He showed that his version is about $20\%$ smaller than the initial version of the state of the art [58]. In a second paper [59], on the topic of AES S-box, Canright proposed a protected version as a countermeasure against SCA. The countermeasure is based on a first-order masking [60]. He showed how to compute the non-linear ($\mathrm{GF}_{2^8}$ inverter) part of the S-box in a masked manner.

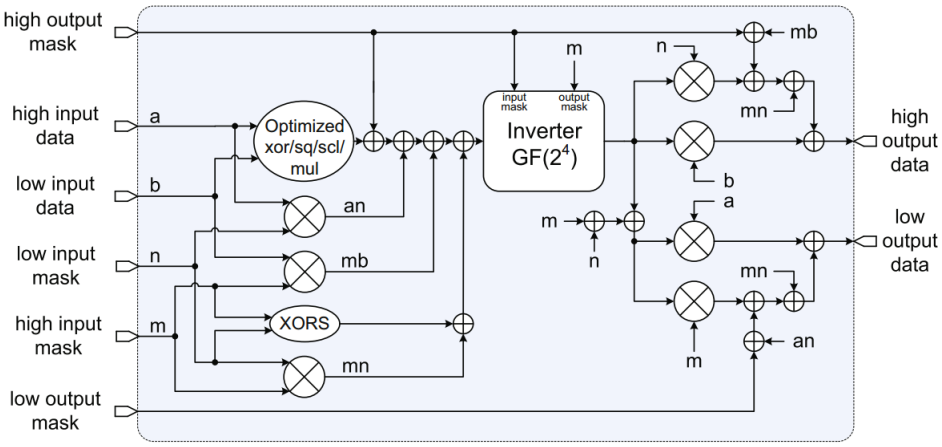



Figure 2.10: Canright masked inversion of $\mathrm{GF}_{256}$ elements [59] and analysed in [36].

Theoretically, such a countermeasure should be robust, at least assuming the gates eval-

uate in the adequate order, *i.e.*, only once all inputs have arrived (RTL case). In [36], the author targeted an AES implementation that instantiates this masked version of the S-box. He managed with first-order analysis to retrieve the secret key. He also found that the number of needed traces is only increased by a factor of six ($\times 6$) compared to a non-protected implementation, which is not enough from this kind of countermeasure. An earlier study on other masked S-box has already been studied in [61, 62]. The results confirmed the existence of a leakage in the non-linear sub-functions of the S-box. A dependency between the total number of transitions and the clear (unmasked) value has been clearly substantiated. He showed that this leakage was due to the absorbed transitions by the second $XOR$ gate of the multiplier. We can notice that at this gate, the calculation involves both shares of the same variable $(a, m)$. In the following, we present two fundamental state-of-the-art approaches to fix this problem.

#### 2.3.3.2 Threshold implementation

In [63, 64], Nikova et al. have proposed a way to implement a non-linear function secure at first order even in presence of glitches. It is based on three main properties:

- Non-Completeness: It is the most important property of Threshold Implementation (TI). It assumes that each gate does not process all shares of the same variable. In other words, each gate should be independent at least from one share.
- Uniformity: The distribution of the shares is uniform.
- Correctness: The sum of the result should be the expected one.

The authors demonstrated that if those three properties are verified, then the circuit will be secure against glitches. They also proposed a first order secure multiplier based on a sharing of order $3$, and a $\mathrm{GF}_{16}$ inverter based on a sharing of order $5$. Both verify the non-completeness property. The TI masked $AND$ between $x = \bigoplus_{i=1}^{3} a_i$ and $y = \bigoplus_{i=1}^{3} b_i$ can be computed as follows:

$$
\begin{aligned}
f_1 &= a_2 * b_2 \oplus a_2 * b_3 \oplus a_3 * b_2 \\
f_2 &= a_3 * b_3 \oplus a_1 * b_3 \oplus a_3 * b_1 \\
f_3 &= a_1 * b_1 \oplus a_1 * b_2 \oplus a_2 * b_1 \\
x * y &= f_1 \oplus f_2 \oplus f_2
\end{aligned}
$$

We can notice that each expression $f_i$ is free from $a_i$ and $b_i$.

Based on these principles, Moradi et al. in [65] have implemented a full AES S-box. It is divided into four phases with four levels of registers. To ensure the global uniformity, the registers behind the multipliers are remasked, with fresh random. This allows to reduce the combinatorial complexity, while satisfying the three conditions of TI for each block. A more

compact version of about $30\%$ with lower latency was then presented in [66]. It computes the result in two cycles instead of four. However, the number of input shares is four, and the output shares is three.

In addition, there are other approaches which aim to completely avoid glitches [67]. The principle was to activate the combinatorial computation recursively, once all signals are arrived, and thus avoid the propagation of non-necessary transitions. The proposals which aim to equalize the delay of arrival of the signals does not allow to eliminate the leakage, but to reduce it only for instance [68].

In [69], the authors extended the TI notion to high order masking. They also presented a $1^{st}$, $2^{nd}$ and $3^{rd}$ TI implementation of a small S-boxes. To be able to check the TI properties on hardware implementations, [70] presented an automated tool which takes an RTL design as input, generates a netlist with Design Compiler (Synopsis), and checks the different TI properties up to order three ($d = 3$). The tool is open-source and available on github: `https://github.com/vmarribas/VerMFi`.

#### 2.3.3.3 Domain oriented masking

The Domain Oriented Masking (DOM) [71] comes with a very similar approach as TI, which consists in separating each mask domain, and optimizing the number of necessary registers and fresh random. The proposed multiplier needs two registers and one fresh random. The computation is performed in three main steps:

- Calculation: This step is similar to the first stage of $AND$ between shares as [51].
- Resharing: In this stage the output of each $AND$ gate is registered and remasked with a new fresh random. Thus, the result will be uniform and independent from the other shares.
- Integration (or compression): This step consists in reducing the number of shares from four to two.

The author in [71] has shown a full implementation of the AES S-box. It is composed of four stage of registers, that stops glitches at the output of each multiplier. This version was $40\%$ smaller than the one presented in [65] and $13\%$ smaller than [66].

| S-Box | Area (GE) | Latency (Cycle) | Fresh random (bits) |
|---|---|---|---|
| Moradi et al. [65] | 4244 | 4 | 48 |
| Bilgin et al. [66] | 3003 | 2 | 44 |
| Gross et al. [71] | 2600 | 4 | 28 |

Table 2.1: Comparison of some glitch-resistant state-of-the-art implementation of the AES S-box.

We show in table 2.1, some state-of-the-art existing implementations of the AES S-box that resist to glitches.

As we can see, there are some similarities between DOM approach and TI. In [72], Reparaz et al. have described the similarities between the different implementations, and how the scheme of [73] can transformed to a secure version against glitches as TI, by pointing out the critical parts that should be treated carefully.

We address this topic more deeply in chapter 5. We explore different ways to secure masked gates, and how to build more compact and secure functions even in presence of glitches.

## 2.4 High-order DPA

The idea behind high-order attacks is to combine multiple time samples, where the masks and the masked data are manipulated [74]. In the case of a first order Boolean masking, the second order DPA can be performed on the absolute difference of the traces. This attack was initially described in the mono-bit-HW power consumption model [75]. For a bit of the mask $m$, and a bit of the masked data $a$, the secret bit value $x$ can be computed also by: $x = |a - m|$.

If the two instants $t_1$ and $t_2$ correspond to the moment when $m$ and $a$ are manipulated with a respective power consumption $P_m$ and $P_a$ then, the consumption of the secret can be inferred from: $P_x = |P_m - P_a|$. It is shown in [76] that this attack is very effective against software implementation on smart-cards. This analysis was also extended to multi-bit-HW power consumption model. Indeed, even when $m$ and $a$ are multi-bit variables, $HW(x)$ still correlated to $|HW(a) - HW(m)|$. Similarly, in [77] the authors presented an attack on parallel hardware masked implementation. They showed that the variance of the leakage depends on the secret key.

In [38], Prouff et al. analysed the different possible combinations of the leaking points, namely, the absolute difference and the multiplicative one, and thus deduced the optimal way in each case. They also explained the relationship between the two leakage models when the noise is high. In [50], the authors presented a study about the influence of noise in the case of a multiplicative and arithmetic combining of leaking points. They also studied the case where $t_1 = t_2$ (the masked data and the mask are manipulated at the same time), where the computation of the second order moment is equivalent to a multiplicative combining when $t_1 \neq t_2$.

## 2.5 Evaluation methods

### 2.5.1 Probing model

To evaluate the security of an implementation against SCA, in particular, the masked implementation, [73] introduced the notion of private circuit. This approach is based on the d-probing model. In this model, the attacker is allowed to place $d$ probes and record the value of $d$ wires. If the secret cannot be found with these $d$ probes, the circuit is considered secure at order $d$. In the same paper, the authors have built secure gadgets (with respect to this model) like the non-linear operation $AND$ secure at any order $d$. It is an extension of the first proposition of a the masked $AND$ of [2].

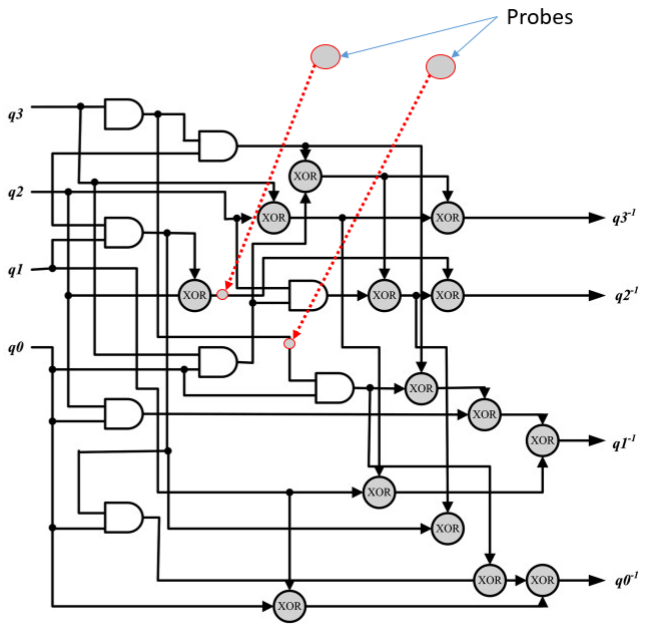



Figure 2.11: Illustration of probing attack model on a circuit. Probing two signals to see if their distribution is in dependent from the secret.

As shown in fig. 2.11, the circuit is secure at order $d = 2$, only if all combinations of two signals are independent from the secret. This first approach is purely algorithmic, and does not wrap all the physical parameters that can induce SCA leakage. In particular, the leakage linked to the power consumption or EM radiation is more correlated with transitions than with the value of the manipulated data.

As already mentioned, in [55], the authors pinpointed a first order leakage in the masked $AND$ gate of [2], which is supposed to be 1-probing secure according to [73]. In fact, this leakage is due to extra-algorithmic transitions (or glitches).

### 2.5.2 Formal verification methods

#### 2.5.2.1 Software implementation

The verification of software implementations is done at instruction level, either at source code level or at assembly (ASM) level. Two models are generally adopted:

- Model in term of value;
- Model in term of transition.

The leakage detection aims to identify dependency with any secret value. In [78], the authors presented a method allowing to mask at compile time a software implementation at the first order, by tagging the different types of the entries as public or secret. They show examples on an AES implementation, where the instructions involving the key are masked, either by adding instructions or by recomputing tables like the S-box. In its initial version, the tool assumes that the instructions are independent and therefore, does not take into account the distance leakage model (or transition). It is based on a specific language, and the designer is asked to add some specific instructions for the compiler.

In [79], the author presented a tool (named Sleuth) capable of dealing directly with Low-Level Virtual Machine (LLVM) code. The evaluator should also specify public and secret variables as well as a leakage model. The tool tracks and checks if all variables are masked and independent from the specified leakage model. It detects the two types of leakages, either linked to the value or to the transition. There is other versions of masking verification tool like [80], which supports more advanced verification, like the uniform distribution of variables, and high order masking schemes.

#### 2.5.2.2 Hardware implementation

The main difference between a hardware and a software implementation is the possibility of parallelising the calculations and carrying out several (algorithmic) instructions within the same cycle. In addition to the different constraints encountered when evaluating a software implementation, the verification of a hardware should ensure that all intermediate calculations performed in the same cycle are independent from the secret values, including extra-algorithmic transitions, such as glitches.

Concerning formal security analysis in presence of glitches, there are few studies. In 2017, Bertoni et al. presented in [81] a methodology to analyse the combinatorial part of a masked circuit. They adopted the concept of transient signals and described an empirical and exhaustive way to evaluate non-linear functions against any type of transitions. To track the origin of the vulnerability, they use a pair referencing the transition and the variable that induces the

transition. Thus, any transition which reveals sensitive data could be detected. They also presented an example on masked *Keccak* function at order two ($n = 2$), and showed how to avoid glitch leakages by designing a sharing at order ($n = 3$), by adopting the TI principles.

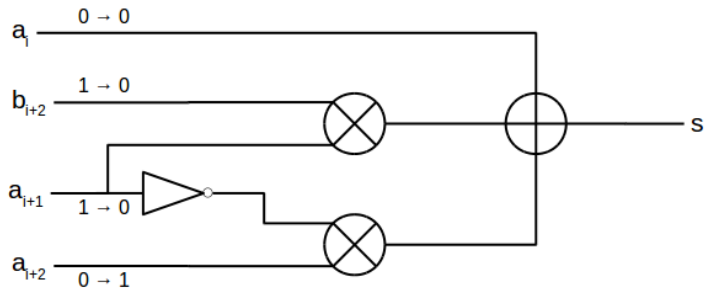



Figure 2.12: Part of masked Keccak. A vulnerability is detected when the transition $(0110 \rightarrow 0001)$ is seen by the last $XOR$ once.

A part of the masked Keccak multiplier at order $n = 2$ is presented in fig. 2.12. The shared value is $r_k = a_k \oplus b_k$. By evaluating all possible transitions, a leakage is detected when the inputs change from $(0, 1, 1, 0)$ to $(0, 0, 0, 1)$ (respectively for $(a_i, b_{i+2}, a_{i+1}, a_{i+2})$). This leakage is visible only if the last $XOR$ evaluates the impact of this transitions arriving from the $AND$ gates at (almost) the same time. Hence the activity at the output will depend on $r_{i+2} = b_{i+2} \oplus b_{i+2}$ which is the unshared secret value.

In 2018, Bloem et al. introduced in [82] a formal method to analyse a masked circuit at any order ($d \geq 2$). They used the Fourier coefficients of the $XOR$ and $AND$ gates, and then, deduced a fast way to propagate the leakage created at the output of each gate. Therefore, it allows checking whether each signal satisfies the property of d-probing secure. This method relies on three main principles:

- **Labelling** Each signal is tagged according to the public and secret values involved in its calculation;
- **Propagation** The output signal is tagged according to the non-zero Fourier coefficient for each variable;
- **Verification** This step checks if each signal (and the circuit) is secured at order $d$.

The propagation rules are derived from the Fourier transform. The labels with a non-zero coefficient are the only ones which are propagated.

In the case of stable signals, the rule of each gate is applied as it is. For the transient signals, to take glitches into account, the authors unified the propagation rules for the $XOR$ and $AND$ gates. Ultimately, this method covers (by overestimating) the leakage that could be generated by a glitch. An example is shown in fig. 2.13. The first one (fig. 2.13a), shows the

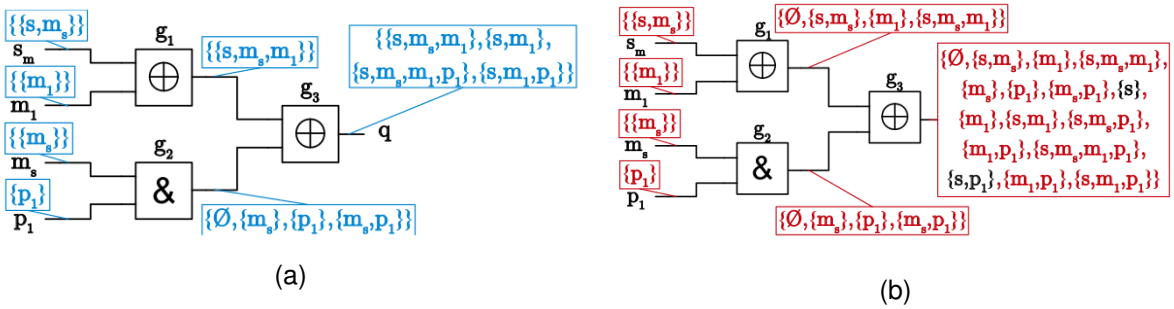



Figure 2.13: Propagation of labels for a small circuit [60]. (a) only stable signals, (b) with transient signals.

propagation of labels without considering glitches, while the second one (fig. 2.13b), where the transient labels are shown in red, are related to the potential leakage that may be created because of glitches. The secret variable $s$ is shown in a single label (in black), without being protected by any mask, thus the circuit is not secure. When registers are inserted, the transient labels are reset to the set of stable ones.

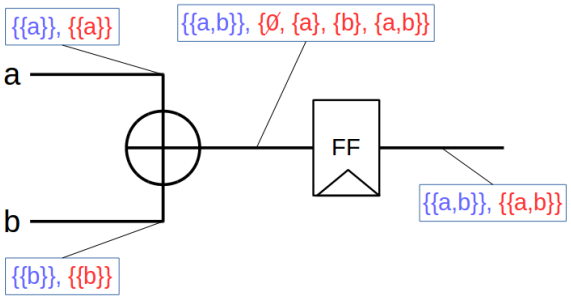



Figure 2.14: Propagation of stable signals and transient signals for a $XOR$ gate. The transient labels are reset to the stable signal after a register (FF) [60].

As shown in fig. 2.14 for a simple example on a $XOR$ gate, the transient labels are stopped, as expected when inserting a register (FF). To achieve the verification phase, a SAT solver is used. The authors demonstrated in some small circuits the effectiveness of this methodology, like masked $AND$ gates of [2], TI and DOM. However, we notice that for a relatively big circuit like AES S-box, the tool takes a long time ($\approx$ 10h). A recent version of the same approach is extended in [83].

In 2019, Barthe et al. in [84] proposed a new approach combining more generalized properties than the classical d-probing model, namely Non-Interference (NI) and Strong Non-Interference (SNI) [85]. Their tool was particularly much more efficient in terms of analysis time compared to [82], and allows a very fast analysis of relatively complex design (like masked AES S-box at order 1 in few minutes). The tool takes as input a synthesized netlist (with yosis - an

open-source synthesis tool). The input netlist should then be annotated to specify the secret and the public variables. Based on a symbolic execution of each instruction, the tool builds a simplified image of the leakage and deduces if each signal is protected or not. They show a simple example on the DOM multiplier, as presented in table 2.2.

Table 2.2: Symbolic execution of the DOM instructions. The leakage is built using the shares of each signal [84]. (a) secure version of DOM multiplier. (b) modified version of DOM multiplier.

| Instruction | Leakage |
|---|---|
| $t_0 \leftarrow b[1] \times a[0]$ | {b[1], a[0]} |
| $r \leftarrow_{\$} \{0,1\}$ | { r } |
| $t_0 \leftarrow t_0 + r$ | {b[1], a[0], r} |
| $t_2 \leftarrow_{ff} t_1$ | {r} |
| $t_3 \leftarrow b[1] \times a[1]$ | {b[1], a[1]} |
| $c[1] \leftarrow t_3 + t_2$ | {b[1], a[1],r} |
| $t_4 \leftarrow b[0] \times a[1]$ | {b[0], a[1]} |
| $t_5 \leftarrow t_4 + r$ | {b[0], a[1], r} |
| $t_6 \leftarrow_{ff} t_5$ | {r} |
| $t_7 \leftarrow b[0] \times a[0]$ | {b[0], a[0]} |
| $c[0] \leftarrow t_7 + t_6$ | {b[0], a[0], r} |

(a)

| Instruction | Leakage |
|---|---|
| $t_0 \leftarrow b[1] \times a[0]$ | {b[1], a[0]} |
| $r \leftarrow_{\$} \{0,1\}$ | { r } |
| $t_0 \leftarrow t_0 + r$ | {b[1], a[0], r} |
| $t_2 \leftarrow_{ff} t_1$ | {r} |
| $t_3 \leftarrow b[1] \times a[1]$ | {b[1], a[1]} |
| $c[1] \leftarrow t_3 + t_2$ | {b[1], a[1],r} |
| $t_4 \leftarrow b[0] \times a[1]$ | {b[0], a[1]} |
| $t_5 \leftarrow t_4 + r$ | {b[0], a[1], r} |
| – | – |
| $t_6 \leftarrow b[0] \times a[0]$ | {b[0], a[0]} |
| $c[0] \leftarrow t_5 + t_6$ | {b[0], a[0], a[1], r} |

(b)

As we can see, the DOM multiplier is secure even in presence of glitches as shown in the expression of the leakage in table 2.2a. In the modified version of table 2.2b, a vulnerability is detected when a register is removed. Both shares $(a[0], a[1])$ of the variable "$a$" are involved in the expression of the symbolic leakage. We notice that this observation is also equivalent and linked to TI principle. For high order evaluation, they provide a large set of masked implementations and a comparison with the tool of [82].

We make a comparison of the implementation issued from those approaches in chapter 5, where we propose a different way to model glitches and verify that each transition is independent from the secret, without involving too strong notions, that may lead to more complex design, unlike NI and SNI properties that may be too strong as mentioned in [83].

### 2.5.3 Pre-silicon security verification

To get closer to a real circuit, several studies aim to simulate traces of power consumption, either with a digital [86, 87, 88] or electrical simulator [68], or by rewriting the algorithm to estimate a side-channel leakage [89, 90]. In the following, we detail the different stages of design of an integrated circuit (IC), and the contribution of each level from a side-channel analysis point of view. An IC has a long-life cycle before being packaged into the end-user product. The part of lifecycle happening before circuit fabrication is called pre-silicon stage.

The starting point is the specification of the IC design. It is a document that describes the entire structure of the design and its pseudo code. From that point, a security checking can start. In fact, a static analysis of the code can be performed to detect potential nodes (conditional branching, unprotected registers, etc.) that can be exploited by a timing or differential analysis. Then, according to the classical design flow, all conception levels can be considered from the security viewpoint:

- **RTL Level** or behavioural level that is very important in detecting the major part of security vulnerabilities in the design. In fact, it allows detecting the leakage based on the common models that can be built directly from the knowledge of the target algorithm. Moreover, it allows an easy validation of most leakage models set after having properly reviewed the code. The RTL level is not dependent on the technology target which allows a more generic evaluation.
- **PS Level** or *netlist* level that regards the state of the code after synthesis (i.e. PS netlist). It allows the detection of specific leakages related directly either to a bad automatic synthesis due to bad simplifications and optimizations; or to a bad implemented combinatorial countermeasure like masking. Moreover, this level is mapped to the technology and provides timing information regarding delays propagation with the design gates. It is noteworthy that those delays might be behind glitches-based leakage.
- **PR level** that regards the state of the design after place and routing process (i.e. PR netlist). It allows the detection of leakages behind a bad routing. It is mapped to the technology and represents the almost final image of the design. It provides timing details regarding delays propagation within the routing of the design instances.
- **Post Layout level** that is the final image of the design when integrated within the chip just before its fabrication by the foundry. It is a 3D representation taking into account the different metal layers of the chip. In term of pre-silicon security evaluation, an FIB analysis can be performed to evaluate the robustness against probing attacks.

In the context of secure implementations, the RTL should be more faithful to the algorithmic description of the countermeasure. Hence, the designer may check the functionality aspect of the countermeasures. When the implementation is mapped to a given technology, some other parameters should be taken into account, such as the propagation time in logic gates. In fact, the input and the output signals are not synchronised. When many gates are in cascade, it will generate a lot of glitches. The effect of such phenomena should be evaluated at an early stage of the design lifecycle, before the fabrication of the final circuit.

## 2.6 SCA & performance issue

When a side-channel evaluation has to be performed with a very high number of traces, the question of performance becomes paramount. The naive calculation of a CPA for example, becomes very greedy in RAM memory. There is a way to do the equivalent computation iteratively, or to reduce the number of traces with light pre-processing without losing essential information. As the power consumption is supposed to be the same when the device manipulates identical data, one can perform a classification pre-processing with respect to, either the sensitive value (S-box output at the first round), or the equivalent value under the bijection assumption (ciphertext or plaintext).

The author in [91] describes how optimizations can be made on the CPA and LRA distinguishers. It classifies the leakage traces before any relevant computation. This approach is based on two sound hypotheses:

- The device leaks the same power when manipulating the same data;
- The same public cryptographic parameter leads to the same sensitive value.

Indeed, the classification is made on either the plaintext bytes or the ciphertext bytes, hence only $256$ traces are handled. As the key is supposed fixed, the input (resp. the output) of the S-box at the first round (resp. last round) are identical, when the plaintext or the ciphertext is the same. The result of the analysis is still equivalent, or even more efficient, namely for the LRA.

However, this optimisation cannot be performed when the leakage model depends on more than one state, like the HD model at the last round. To allow the same optimization, the traces should be classified on two different bytes. It leads to $65536$ classes, which is relatively huge. To bypass this problem, the classification should be made on the leakage model output. Thus, we keep only $256$ traces, which is the cardinal of the possible input bytes. This optimisation has been detailed in [92]. This issue is taken into account when analysing the side-channel traces issued from our implementations, analysed in chapter 4 and chapter 5.

## 2.7 Fault analyses

The objective of fault injection is to disrupt the electronic device with physical means, to corrupt the calculations and try to deduce sensitive information. In this context, the attacker exploits the faulty data and the correct data to break the key using a Differential Fault Analysis (DFA).

### 2.7.1 Fault model

The injections of faults can be performed by disturbing the power supply, the clock, laser or EM pulse [93]. The impact of an injection varies depending on the source. The injections on the power supply or the clock have rather a global effect. The whole circuit will be affected by this kind of disturbance. Laser injections are the most precise in terms of locality, space and time. With a spot size that can reach a precision of a micrometre, it is possible to target SRAM cells of $65\ nm$ technology, to perform bit-flipping, bit-set or bit-freeze.

The fault models are derived from the expected impact depending on the type of injection. Global effect attacks like voltage disturbance and clock glitches have an overall impact on the circuit, and all (or much of) the computation can be altered. For local attacks, like laser, the impact can be modelled on a small area of the circuit, either by a random modification, a freeze or a set of the signal value.

Depending on the algorithm, the number, the precision and the locality of the faults may differ. This is linked to the DFA methods. We detail this relationship in the following sections.

### 2.7.2 DFA on asymmetric algorithms

Since the publication of Bellcore attack in [94] by Boneh, DeMillo and Lipton on an RSA-CRT, researches have been multiplied to explore several models of faults on different implementations. In the RSA-CRT version, a single injection in one modular exponentiation is enough to find the secret key. When $S$ and $S'$ are respectively the correct and the faulted signature, a secret factor $p$ of the RSA modulus $N$ can be retrieved using the Greatest Common Divisor (GCD). Thus, we have: $p = gcd(N, S - S')$.

As one (and only one) modular exponentiation is faulted (suppose that is the one mod $q$), we get $S'\ mod\ q \neq S\ mod\ q$ and $S'\ mod\ p = S'_p$. Thus, the difference $S' - S$ is a multiple of $p$. For other implementations like ECC many faults should be injected to recover the key.

### 2.7.3 DFA on symmetric algorithms

For symmetric algorithms such as DES and AES, several methods have been proposed to detail the way to exploit cipher errors at the last round. In [95], Biham and Shamir described a way to extract a DES key. They showed that the secret key can be recovered with less than $200$ faulted ciphertexts.

Giraud presented in [96] an alternative way to attack an AES key, which was more complicated than DES because of the strong diffusion-confusion property of the AES. First, he described an attack based on a single bit error. By analysing the distribution of the resulted faults,

the attacker can extract the right key byte with less than five $(5)$ observations. For a 128-bits key size, the whole key can consequently be retrieved with less than $50$ faulty ciphertexts.

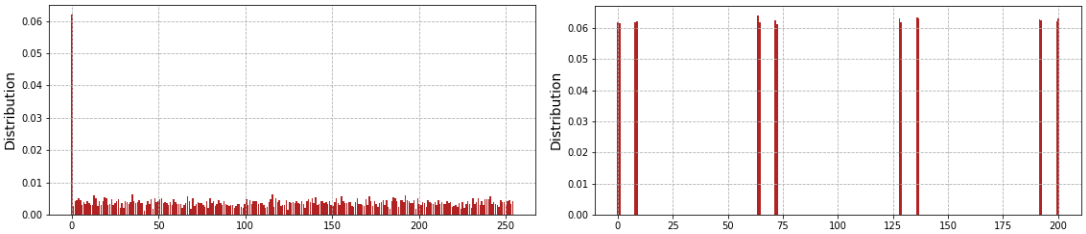
This attack needs to inject a signal fault on each byte of the state register, which leads to sixteen different positions. When attacking the round $9$, we need to inject fault only at four different positions. Because of the diffusion property of *MixColumns*, one faulted byte at round $i$ will lead to four faults at round $i+1$. Hence, less positions are required to retrieve the secret key [96, 97, 98]. When an error $e$ is injected to the state $M_9$ (input of the last round), the equation of the error satisfies:

$$\begin{aligned} e &= sbox^{-1}(C \oplus K_{10}) \oplus sbox^{-1}(D \oplus K_{10}) \\ &= sbox^{-1}(sbox(M_9)) \oplus sbox^{-1}(sbox(M_9 \oplus e)) \end{aligned}$$

where $C$ and $D$ are the correct and faulted ciphertext respectively, and $K_{10}$ is the last round key. When performing a DFA, the predicted $e_K$ based on the key hypothesis $K$ satisfies:

$$e_K = sbox^{-1}(sbox(M_9) \oplus K_{10} \oplus K) \oplus sbox^{-1}(sbox(M_9 \oplus e) \oplus K_{10} \oplus K)$$

When the right key hypothesis is guessed, the distribution of $e_K$ will be the same as $e$. For a non-uniform injected error, the right key can be extracted with very few faulty ciphertexts. In fact, when $K \neq K_{10}$, the distribution of $e_K$ is almost uniform (if we exclude the distribution of the zero value).




(a) (b)

Figure 2.15: Error distribution of the error for different key hypotheses. (a) wrong key hypothesis. (b) Right key hypothesis.

An example of the distribution of the error is shown in fig. 2.15. For a wrong key hypothesis (fig. 2.15a), the distribution of the error $e_k$ is almost uniform (except for value zero). For the right key (fig. 2.15b), the distribution of $e_k$ is uniform over its support, but not over the integers $\{0, \cdots, |C| - 1\}$, which makes it distinguishable from the wrong-key-error distributions.

We note that when the injected errors have no effect $(e = 0)$, it will result on a non-faulted ciphertext $(D \oplus C = 0)$. This event arrives with probability $\frac{1}{|e|} = \mathbb{P}(D \oplus C = 0)$. When the injected error is different from zero, we have:

$$D \oplus C = x \neq 0$$

and

$$\mathbb{P}(D \oplus C = x | e \neq 0) = \frac{1 - \frac{1}{|e|}}{|C| - 1}$$

As an example if the faulted bits are $\{7, 6, 3, 0\}$, thus $|e| = 16$, which gives $\mathbb{P}(D \oplus C = 0) = 6.25\%$, and $\mathbb{P}(D \oplus C \neq 0) \approx 0.37\%$, which matches the values of the histograms given in fig. 2.15.

This kind of DFA is presented and intensely studied in [99] and known as Non-Uniform Error Value Analysis (NUEVA). This condition is very relaxed compared with a single-bit fault injection model.

In [100], the authors resumed the different existing techniques that an attacker can use for EM injection to induce fault on a circuit. They also give some details about faulting analog and digital logic, by the mean of harmonic or EM pulses respectively.

In chapter 6, we study a protected implementation against fault injection at pre-silicon level. We see how the synthesis phase can impact the result of the error detection rate, using some DFA metrics.

## 2.8 FIB for probing

The micro-probing attack can be performed in practice using a FIB station. The attacker may target buses to read the memory content, or combinatorial signals to read an intermediate sensitive values. There are two major countermeasures used to protect against this kind of attack.

The first one consists on implementing a masking scheme, where the attacker needs to combine $d$ wires to retrieve the secret [73]. The principle is to share the secret into several parts, so the attacker must probe more signals to be able to reconstruct the secret, which makes the attack more difficult.

The second one is based on an active shield [101]. It is integrated into the chip itself on metal layers. The goal is to detect any physical intrusion by activating an alarm, when a shield wire is cut (cf. fig. 2.16). The orange path activates the alarm, and the intrusion is detected (because the milled hole had cut a shield wire) where the blue path does not detect the intrusion, as it is milled with a high aspect ratio FIB, which prevents a complete cut of the shield wire.

This approach is a race between the precision of the FIB (or performance) and the characteristics of the shield. The most important parameters for the latter are; the wire width and the spacing. The denser it is, the more efficient the shield is to detect intrusions.

The FIB performance depends on several parameters. From an attacker's perspective, it is the resolution of the spot that is decisive. It depends on the technology of the FIB, the voltage

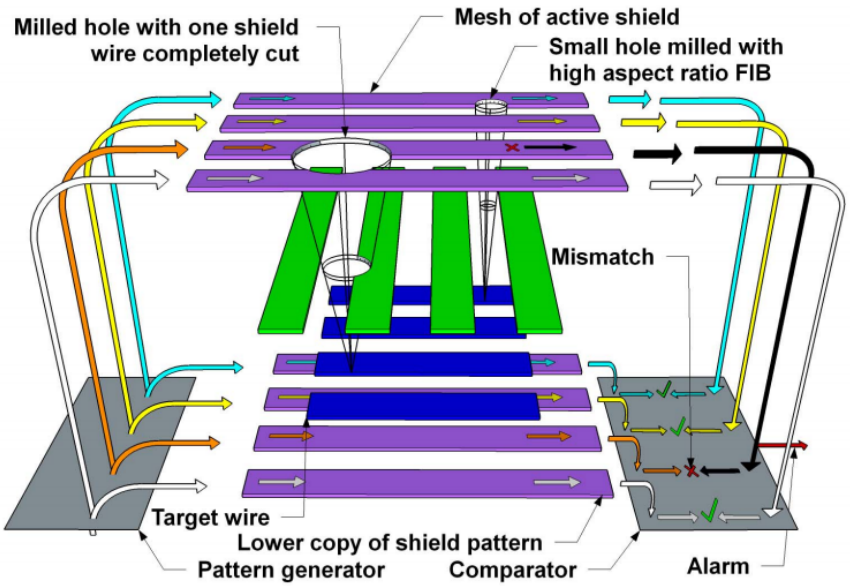



Figure 2.16: Probing a protected design with an active shield (from [3]).

and the current limits. With the size and the shape of the spot, we can model the hole as a cone [102], hence the ratio of the FIB, which is also the ratio between the diameter and the depth of the hole.

Several experiments have shown that for holes with a diameter higher than $100\ nm$, a ratio of $10$ can be achieved. For diameters lesser than $100\ nm$, the ratio decreases to $1$, and even at lower levels [103]. This decrease is due to the shape of the hole. When the diameter is small, it becomes difficult for the extracted particles from the sample to come out. Therefore, it would be more difficult to increase the depth without increasing the diameter [103]. To enhance the ratio, Helium ion ($He^+$) beam can be used instead of Gallium ion beam ($Ga^+$), which gives a high resolution to the ion beam.

### 2.8.1 FIB - Brief description

FIB is a scientific instrument, widely used in the semiconductor and integrated circuit domains. It consists of a focused beam of ions accelerated to a certain energy ranging from $1\ k$ to $50\ k$ electro-volts (eV), with a current between of few pico to some nano Amperes. The liquid usually used is Gallium (Ga), but we can also find sources of Helium, allowing a better resolution. The ions are extracted from the liquid using a high electric field.

Like an electron microscope, FIB can be used for high-resolution imaging up to $5\ nm$, using a low current (a few $pA$), or for milling with a higher current on the order of some nano Amperes. The voltage and the current are controlled by two apertures placed in series (shown in green in

fig. 2.17). By keeping the second aperture constant, the current can be varied by adapting the first one, thus only a portion of the projected ions passes through.

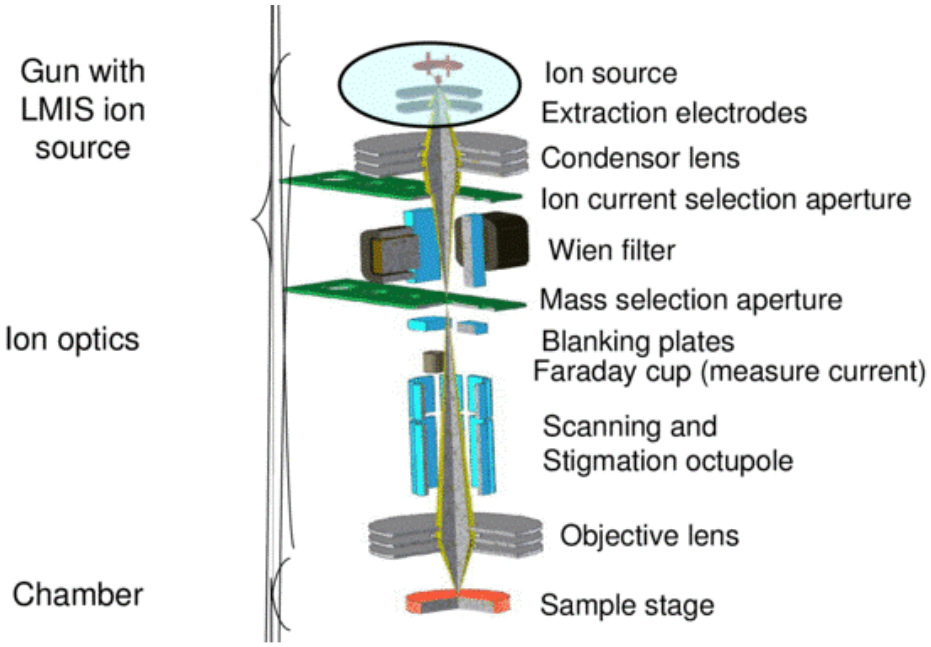



Figure 2.17: Different components of the ion beam column.

The ion beam is focused with the help of electrostatic lenses to a small point, which gives the resolution of the beam, known as spot size. A FIB station is equipped also by an electron microscope for a better and non-destructive visualization of the sample, and a gas injection column to clean the surface during milling process. The performance of a FIB is determined by the following parameters:

- **Ion Beam** it depends on the voltage $V$, the current $I$ and the aperture of the ion column;
- **Electron Beam** used for imaging.

Those two parameters determine the resolution and the performance of the FIB station [104, 105]. For example, at $30\ kV$ and $1\ pA$, the resolution of the ion beam, or the spot size may reach $7\ nm$. The distribution of the ions follows a Gaussian Probability Density Function (PDF) [106]. It is the main factor involved in the milling process to access sensitive signals [107].

The authors in [102] provide a mathematical model for the ion beam profile and different equations to estimate the diameter, the depth and the dwell time. It is also important to mention that the smaller the diameter is, the lower the sputtering yield is. This can be explained by the fact that among the sputtered particles, some of them are redeposited on the substrate, which leads to a lower hole ratio [108].

The milling step can be enhanced to achieve a higher aspect ratio as presented in [109], by activating the Electron Beam (EB) to reduce the Coulomb interaction, and fix to a very low current the ion beam. In [110], the authors show different techniques to achieve a high aspect ratio and sub-micro diameter holes. By fixing the dwell time to $0.1\ ms$ and the current at $48\ pA$, they achieved an absolute depth of $1.8\ \mu m$ with a relative diameter less than $300\ nm$, which gives a ratio of six ($R_{FIB} = \frac{depth}{diameter} = 6$).

### 2.8.2 Micro-probing attack

A micro-probing attack consists of several stages; reverse engineering, pads creation (connections) and the extraction of the secret.

**Reverse Engineering** The reverse engineering is the most difficult and constraining step to perform. It consists on exploring the circuit using a clone device and trying to build a 3D-image of the layout, or a specific part of the circuit.

This step allows the attacker to identify sensitive signals and those which allow an optimal secret key extraction, with a low number of observations and low number of probes.

**Pad Creation** This phase is carried out on the target device, and based on the previous step to create connections with the sensitive signals, without altering the functions of the circuit.

**Key Extraction** Once the connections are completed, the attacker can run the target and record the signal values. The attacker will then be able to eliminate some of the key hypotheses, which do not match the observed values from the sensitive signal.

Examples of attacks on a micro-controller are described in [111]. In [112], the authors presented also the different means allowing to extract an AES-128 key, with lesser number of probes, and to determine the number of observations necessary in various scenarios.

In [113], the authors described the theoretical complexity of a probing attack on some known algorithms, such as DES, RSA and RC-5. They also showed that the number of observations required is very low. To retrieve $6$ key bits of DES, only six encryptions are required.

This topic is addressed in chapter 7. Based on the different characteristic of a FIB, and the different parameter involved in a probing attack. We propose a full and an automated methodology to evaluate protected implementation with a shield against probing attack. Not only this approach is demonstrated on an existing state-of-the-art implementation of AES, but we show how the security of such a design can be enhanced. Our recommendations agree with the expected results and the pre-existing studies on this topic.

# Part I

# Passive attack and countermeasures

# Chapter 3

# Pre-silicon to Post-silicon Analysis

Contents


3.1 Introduction . . . . . 41
3.2 Leakage and security level . . . . . 42
3.3 End-to-end security evaluation . . . . . 43
3.4 Discussion . . . . . 47
3.5 Conclusion . . . . . 48


## 3.1 Introduction

According to the criteria of security evaluation against side-channel attacks, the number of observations needed to break an implementation is the most important parameter. Indeed, for the same implementation, the number of observations can vary from one target to another. This is linked, among other things, to the quality of the acquired signal, the tools used to exploit the leakages, but also the nature of the targeted operation.

The power consumption is usually estimated as a combination of the number of changes in the circuit, and a current leakage (when the circuit is in a stable state). The latter is often defined as a static leakage. The purpose of this chapter is to show, from a high-level point of view, how a link can be established between a virtual target and a real target of the same implementation. Starting from a simple knowledge of the real target, we will extrapolate and determine the number of observations needed to find the secret key of an AES implementation. To answer this question, we need to define some basic concepts to allow us to derive a metric of quantification of the number of traces needed and thus, the security level.

## 3.2 Leakage and security level

To determine the security level of a cryptographic implementation against side-channel attacks, the SR metric is usually used as already presented in section 2.2.3. This metric gives the probability of finding the secret key, for a given number of traces. It can be translated as the probability of distinguishing the right $k^*$, among all possible keys $\mathcal{K}$ [26, 27, 30, 31]. This probability is generally related to a distinguisher, and can be estimated empirically.

In real acquisition (like EM), we cannot actually define directly which part is the signal and which part is the noise. It can only be estimated using the total variance, conditional average and variance. In case of vertical analysis, an attacker tries to measure how the amplitude of the traces ($Y$) varies in terms of some intermediate value ($X$). In this case, we can estimate for some $X = x$, the signal part $S_x$ as: $\hat{S_x} = \mathbb{E}[Y|X = x]$, and the noise part $B$ as: $\hat{B_x} = \mathbb{V}[Y|X = x]$, thus we get:

$$\hat{SNR} = \frac{\mathbb{V}_x[\hat{S_x}]}{\mathbb{E}_x[\hat{B_x}]}$$

As explained above, the determining factor for a SCA is the SNR. In the following, we present a comparative study between a virtual and real analysis, based on power consumption traces, and the CPA. To do so, we use the success rate metric described in [25]. The author gave a formulation using the correlation value $\rho_r$, to estimate the number of needed traces $N_r$ to find the secret key:

$$N_r = 3 + 8\left(\frac{Z_{1-\alpha}}{\ln(\frac{1+\rho_r}{1-\rho_r})}\right)^2 \tag{3.1}$$

where $Z_{1-\alpha}$ is the quantile at $(1-\alpha)$ probability of the centred Gaussian distribution. This result was derived from the Fisher Z-transformation of the Pearson correlation coefficient. Besides, $\rho_r$ refers to the correlation between the leakage model $M$ and the leakage traces $L$. We can divide the leakage trace into two parts, the signal $S$ and the noise $B$: $L = S + B$. The noise $B$ is generally considered to be independent from the manipulated data ($\rho(B, M) = 0$). By rewriting the correlation between $L$ and $M$ we get:

$$\rho(M,\ L) = \rho(M,\ S + N) = \frac{\rho(M,\ S)}{\sqrt{1 + \frac{1}{SNR}}} \tag{3.2}$$

we can identify the correlation of the signal $S$ with the leakage model $M$, and the SNR. According to our definitions, the leakage detection metrics presented in [39], known as NICV, verifies:

$$NICV = \frac{1}{1 + \frac{1}{SNR}}$$

In the following section, we will verify these estimations on virtual and EM traces, by considering different levels of SNR.

## 3.3 End-to-end security evaluation

### 3.3.1 Experimental observations

To perform our experiments, we have synthesised a hardware AES-256 on a SmartFusion2 FPGA for EM measurements, and we generated virtual traces using digital simulations at RTL. In both experiments, we summarised the evolution of the CPA and SNR. We used the rank filter metric to compare the effective number of needed traces to recover the right key and the extrapolated one from eq. (3.1).

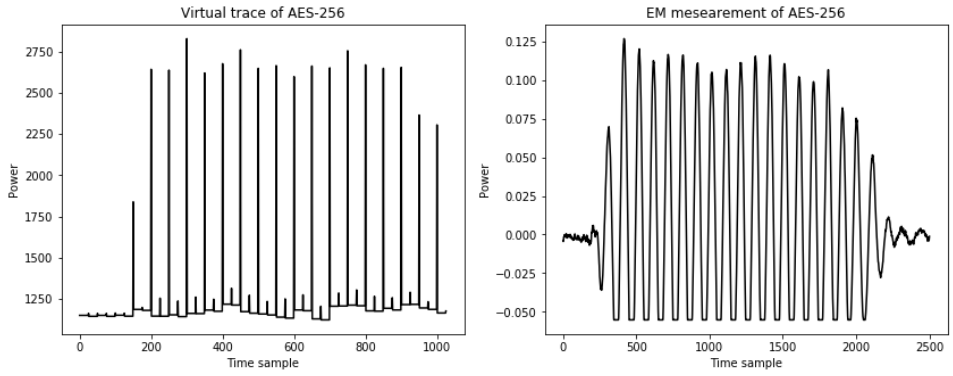



Figure 3.1: Power traces. Left: virtual trace, right: EM measurement.

In fig. 3.1 we show an example of real EM and virtual trace. Using 10,000 traces in both experiments, we have performed a CPA to obtain, on one hand the value (good approximation) of $\rho_r$ and on the other hand, the number of required traces to recover the secret key. The result is shown in fig. 3.2.

The leakage model is based on the HD at the last round. We target 8 bits at a time, and we use the ciphertext bytes $\{c_i\}_{i=0,\ldots,15}$ to recover the key byte $k_i$. The leakage model is computed in two steps:

- For a key hypothesis $k_i$, compute: $r = \text{S-Box}^{-1}(c_i \oplus k_i)$
- Compute $M(k_i)$, the HD between $r$ and $c_{ShiftRow(i)}$.

The best key hypothesis $k_i^*$ is recovered by:

$$k_i^* = \underset{k_i}{\operatorname{argmax}}(\rho(L, M(k_i))$$

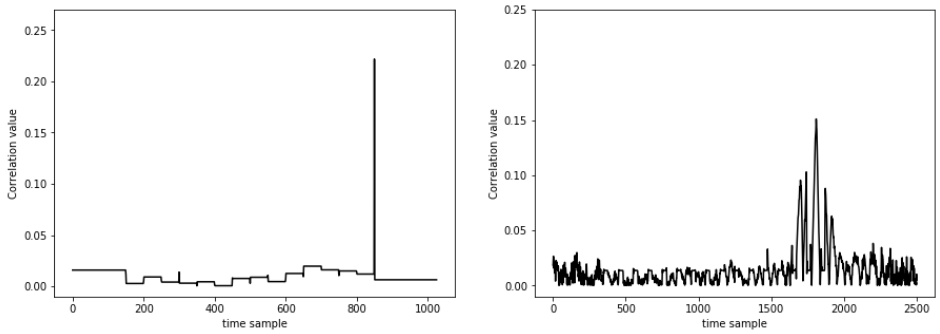



Figure 3.2: CPA using 10,000 traces Left: Virtual trace, Right: Real EM measurement

From eq. (3.1) we deduce the average (Av.) of the theoretical number of required traces and the effective one over all bytes. We give also the maximal and minimal bounds of $N_r$ found over bytes (denoted Max. and Min.). We chose $\alpha = 0.5\%$ (so $Z_{1-\alpha} = 99.5\%$).

From table 3.1, we realise that the estimated number of virtual traces $N_r$ is the closest to the

Table 3.1: Experimental and theoretical number of traces

| Campaign | $N_r$ | | | Required traces (SR) |
|---|---|---|---|---|
| | Min. | Max. | Av. | |
| Virtual-Probing | 310 | 430 | 355 | 420 |
| Virtual-Power | 1350 | 2642 | 2105 | 2200 |
| Real-Probing EM | 500 | 1040 | 730 | 1100 |

true number of required traces.

Firstly, in the virtual case we distinguish:

- *Probing only sensitive signals*. This is equivalent to a power acquisition where we focus our analysis only on the signals that involve sensitive data. So, we expect the attack to be fast;
- *Probing all the design*. This is equivalent to a power acquisition where the whole design is taken into account. Clearly, we need more traces to recover the secret key due to the accumulated noise.

For both probing methods, $N_r$ gives a maximal bound that is close to the real number of traces. In the real acquisition, we have probed the most leaking decoupling capacitors.

In fig. 3.3, we can see that the CPA converges approximately (for most of bytes) after a threshold of $600$ traces in the virtual case and about $1300$ traces for the real one.

After processing $600$ traces (on average) of the virtual traces (resp. 1300 of real EM), the key is recovered with success probability of $99\%$. This corresponds to the threshold of CPA convergence (see fig. 3.4 and fig. 3.5).

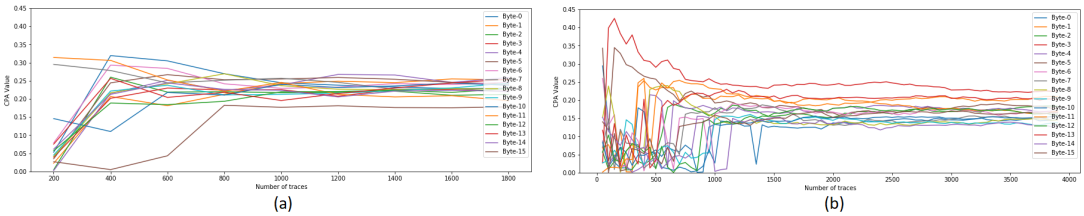



Figure 3.3: CPA convergence, (a) virtual trace, (b) real EM measurement

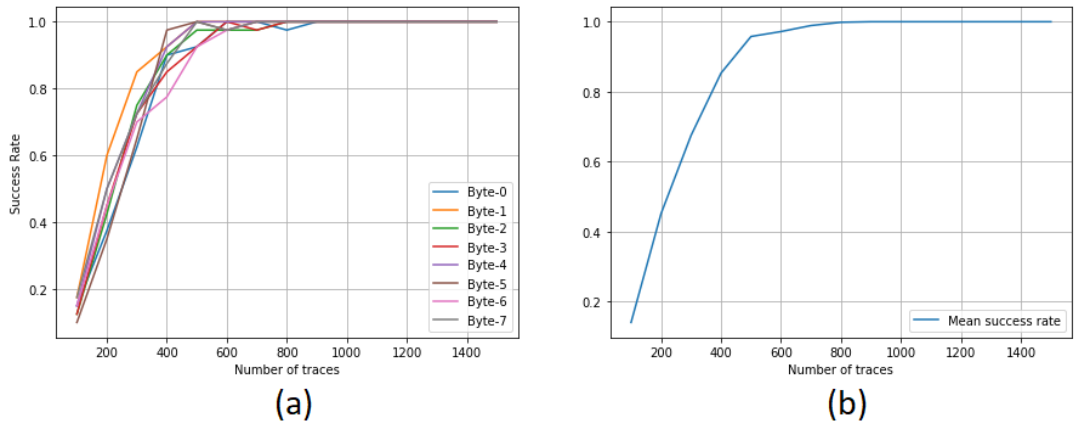



Figure 3.4: SR convergence of virtual traces: (a) SR of the 8 first bytes, (b): the average of the SR over all bytes

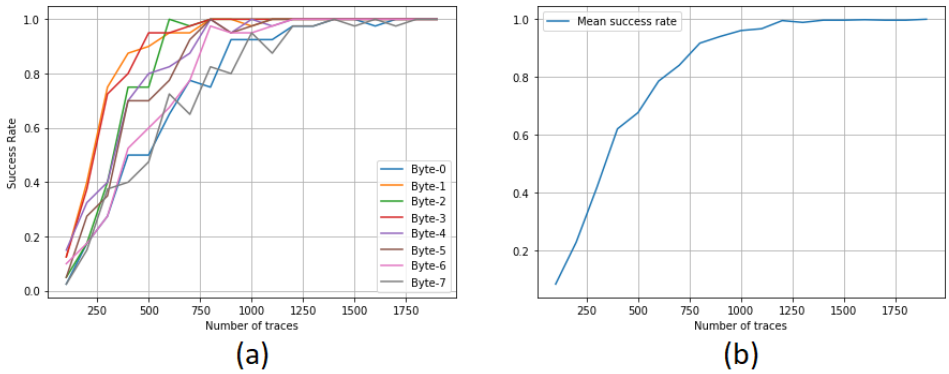



Figure 3.5: SR convergence of EM traces: (a) SR of the 8 first bytes, (b): the average of the SR over all bytes

We note that, when we tried to use the HW leakage model at the last round (using only the result of the S-box$^{-1}$), the attack was unsuccessful in the case of EM traces, and only few bytes are recovered on the virtual case. This is due to the imperfection of this leakage model regarding the power consumption, which depends as explained in section 2.5 on the number of switching bits (signals), rather than the manipulated value.

We see in the next section an example of how to map and predict the results between two

experiments, independently from the setup platform and the used metric.

### 3.3.2 From virtual to real SCA

In the case of virtual traces, we have only the algorithmic noise, so the theoretical SNR ($SNR_{Theoretical}$) can be computed as follows:

$$SNR_{theoretical} = \frac{A_{size}}{128 - A_{size}} = \frac{1}{15}$$

Where $A_{size}$ is the number of bits targeted during the analysis, which is 8 in our case. From eq. (3.2) we can replace the value of $\rho_r$ in eq. (3.1) using eq. (3.2).

In order to find a way to compare the complexity of the attack between two scenarios, we can build a metric $F$ that extracts the number of traces from the value of SNR. We can replace in eq. (3.1) the value of $\rho_r$ defined by eq. (3.2), and assuming that the measured information (signal part) is the same in the virtual case and real case (EM) ($\rho_{ms} = \rho(M, S)$), we will obtain the following relation:

$$F(s) = \frac{\beta}{ln\left(\frac{1+\frac{\rho_{ms}}{\sqrt{1+\frac{1}{s}}}}{1-\frac{\rho_{ms}}{\sqrt{1+\frac{1}{s}}}}\right)^2} \tag{3.3}$$

where $s = \frac{SNR_{theoretical}}{SNR_{Real}}$ and $\beta$ is a normalisation factor such that $F(1) = 1$. eq. (3.3) gives (approximately) the relation between the number of traces in two different conditions of the same implementation. In fig. 3.6, we plot this function in the range $[1, 7]$ to cover the rate of our experiments (the estimated real SNR is close to 0.035), and fixed the value of $\rho_{ms}$ to $0.95$ which is computed from the virtual traces using $SNR_{theoretical}$ and the empirical correlation coefficient.

For example, we have:

- The SNR is close to $0.035$ in the case of real acquisition
- $s = \frac{\frac{1}{15}}{0.035} = 1.905$
- The image of $F$ gives $1.64$

thus, we get a ratio of $1.64$ between virtual and real traces.

Table 3.2: Estimated number of traces based on SNR

| Campaign | Average at 99% | Average at 90% | Using F |
|---|---|---|---|
| Virtual-Power | 2200 | 1900 | 1940 |
| Real EM | 1200 | 900 | 985 |

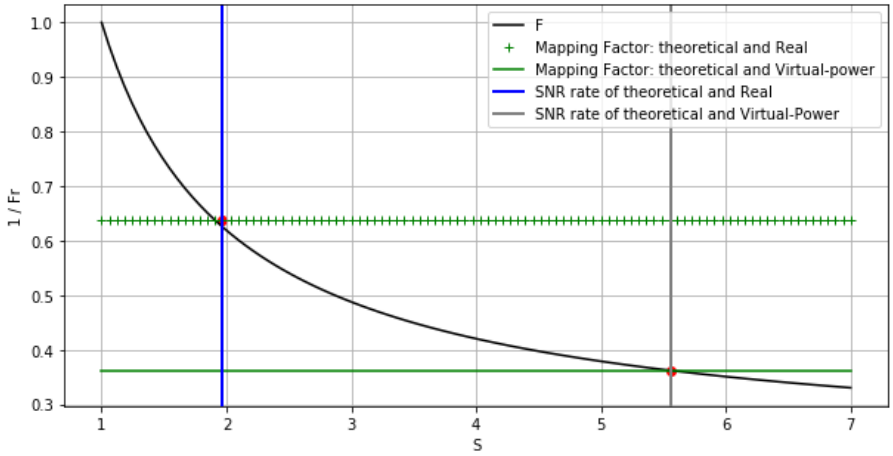



Figure 3.6: The mapping function between SNR and number of required traces comparing with the theoretical one.

In table 3.2 we have summarised the different results obtained over the three campaigns. The third column is calculated by: $600 * F_s$, where $F_s$ is the image of F at the considered SNR level, and $600$ being the convergence threshold of the CPA for traces of the Virtual-1 campaign, used as a reference to calculate the third column.

## 3.4 Discussion

As already mentioned in section 2.2.3, several studies have been able to estimate the success rate of a DPA on a cryptographic target. Those metrics are used to substantiate the level of security that an implementation might have, and the request for encompassing reliable countermeasures, when an attacker has a physical and a privileged access to the target.

Despite the fact that these metrics justify these two essential points, the projection of the security level (number of traces) on a real target remains a fundamental question, because these metrics do not take into account the true instance of the implementation, and a more generic model is usually assumed (such as the non-distinction between measurement noise and algorithmic noise).

With a posterior knowledge of the hardware behaviour, we established closer and more accurate estimation, with more realistic conditions of an attack scenario, such as measurement and algorithmic noise. The latter is characterized by digital simulation, which exhibits the behavioural aspect of the circuit, and allows to estimate its overall activity, thus deduce the intrinsic noise of the implementation.

## 3.5 Conclusion

In this study, we verified a theoretical prediction in presence of relatively low noise ( SNR $\geq 1\%$). The number of traces needed to find the secret key is relatively low ($\leq 2K$). When the same implementation is analysed under several noise conditions, a link can be established through the function $F$ given by eq. (3.3). We should note that this function is now independent from the used leakage model. The only thing that matters is the noise level on the new target. The latter depends only on the measuring environment and the equipment used to mark the difference between a real and pre-silicon analysis. Without significant countermeasures, the difficulty of an attack on a real target is determined by the surrounding noise and radiation level of the chip. The aim of this approach was to present methods for checking the resistance of a circuit in a pre-silicon context. Such an approach will help developers to properly integrate security functions into a system instantiating cryptographic primitives. Countermeasures have been developed against SCA, and the level of leakage has decreased significantly. Nevertheless, the desired security level must be reached with regard to other physical phenomena, such as leakages detected on masked implementations due to glitches.

# Chapter 4

# Post-Synthesis Analysis of a Masked Implementation

Contents


4.1 Introduction . . . . . 49
4.2 Analysis of a masked implementation . . . . . 50
4.3 Discussion . . . . . 59
4.4 Conclusion . . . . . 59


## 4.1 Introduction

To counter SCA, several countermeasures were widely studied in the state-of-the-art. They aim to make the measurable physical leakage independent from the manipulated data. There are several techniques to protect a given algorithm. Depending on the structure and the nature of the considered implementation, some transformations can be adopted more or less easily. The best known techniques rely on masking and shuffling. Masking is the most common studied countermeasure in the state-of-the-art. It consists of performing equivalent calculation by dividing the variable into several shares [47, 48]. This should prevent SCA from revealing sensitive information from power consumption traces. Theoretically, such a countermeasure is considered very reliable. According to the probing-based evaluation model [73], it becomes impossible to find the secret by recording a single signal or variable, when well implemented.

In addition to the algorithmic considerations to ensure the security of the implementation, the design life cycle of the circuit can alter some features, and thus it induces tragic simplifications. As a result of this process, vulnerabilities may appear. For example, some signals may be unmasked, because of a series of optimizations performed by the synthesizer. To avoid this type of unfortunate situations, it is necessary to ensure that at each design step, the circuit

complies with the expected security properties. In particular, at each level of design step, it is necessary to check that all signals remain well masked and independent from the secret data.

This is quite feasible with the help of digital simulations. Each signal can be processed separately and then, by analyzing its distribution, we can deduce if it is independent from the sensitive data. Dependency detection metrics such as correlation, variance analysis or mutual information, can be applied to each signal. If a bias is detected, then the signal will be tagged as vulnerable. We can perform these steps iteratively, in order to detect vulnerabilities on each design step. This allows a full integration into a concrete development environment.

## 4.2 Analysis of a masked implementation

In the following, we look in more details into an AES implementation with a masked S-box (described in section 2.3.3.1). Based on a pre-silicon analysis, we identify the different source of leakages, propose a fix and reiterate the analysis. This assessment covers the most relevant design step of synthesis to prohibit the propagation of any algorithmic leakage.

**Notations** To illustrate the results of the analysis more clearly we define:

- $\mathcal{K}$: the set of possible keys
- $\mathcal{C}$: the set of possible ciphertexts
- $\mathcal{S}$: the set of signals in the target design
- $\mathcal{T}$: the set of power traces

We start in section 4.2.1 with RTL analyses to verify the countermeasure at the algorithmic level, and that all signals are correctly masked, at least when they are evaluated in a proper order. Then, in section 4.2.2 we present the same analysis on timing-annotated netlist, synthesised on FPGA.

### 4.2.1 RTL analyses

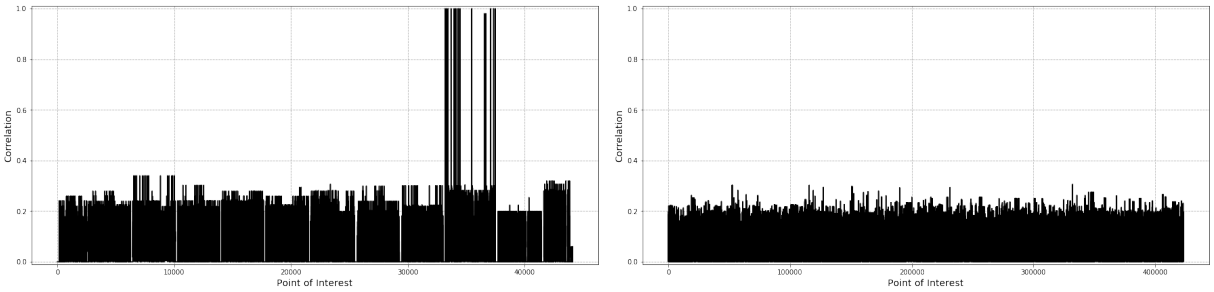
The purpose of the RTL analyses is to verify that the implementation at the algorithmic level respects the expected properties. To do so, we carry out a simulation campaign with a fixed key and random messages. For the detection metric, we use the Pearson correlation. The leakage model $M$ corresponds to the input of the S-box at the last round, therefore, it is calculated as follows:

$$\begin{array}{rcccl} M & : & (\mathcal{K},\mathcal{C}) & \mapsto & \mathcal{C}, \\ & & (k,c) & \mapsto & \text{S-box}^{-1}(c \oplus k), \end{array} \tag{4.1}$$

Each signal $s \in \mathcal{S}$ is then correlated with each bit of $y = \text{S-box}^{-1}(c \oplus k) = (y_7, \cdots, y_0)$:

$$\begin{gathered} \forall (k, c, i) \in (\mathcal{K}, \mathcal{C}, [0; 7]), \\ \rho(y_i, s) \end{gathered} \tag{4.2}$$

If a given signal is unmasked, the correlation will equal $1$. We should note that $s$ is a binary-temporal signal. The correlation is calculated between $y_i$ and each time sample of $s$. In the case of RTL, those samples correspond to the clock edge.




(a) Unmasked implementation. (b) Masked implementation.

Figure 4.1: CPA result for an unprotected and protected implementation at RTL level.

To illustrate the result of this analysis, fig. 4.1 shows a comparison between the correlation coefficient of masked and non-masked implementations. For the masked version, the score for the right key remains drowned in noise and indistinguishable from the wrong keys. The curve shows the results of all signals at all time samples. For each time sample, we have concatenated the active signals. This allows us to affirm that this implementation is correctly designed at the algorithmic level (RTL). We can therefore conclude that this design has no unmasked data compared with an unmasked version.

As already mentioned in section 2.2.3, the number of traces needed to distinguish the secret key depends only on the confusion coefficient of the S-box (which is about $\kappa = 56\%$). In fact, this coefficient gives (on average) the number of key candidates to eliminate at each new observation. If for each new observation we eliminate $56\%$ of remaining keys, then the right key is retrieved after $n$ measurements, such that:

$$256 * (\kappa)^n \leq 1 \Longrightarrow n \geq 10$$

However, these observations should be made with different messages. In our experience, we have acquired $50$ traces to get a significant correlation peak, compared with the noise level, as we can see in fig. 4.1. In fact, when the implementation is not protected, the correlation will be equal to $1$. Increasing the number of traces allows to reduce the level of correlation with the wrong keys until reaching a negligible level.

Even though this analysis shows that the implementation is secure at RTL – which is a necessary condition – it is not sufficient to state that its corresponding implantation on a specific technology will keep the same expected security properties.

For this reason, performing analysis on back-annotated netlist is of utmost importance in this context, in order to have a more reliable characterisation of the design robustness. Hereafter, the obtained results on this abstraction level are exposed and detailed.

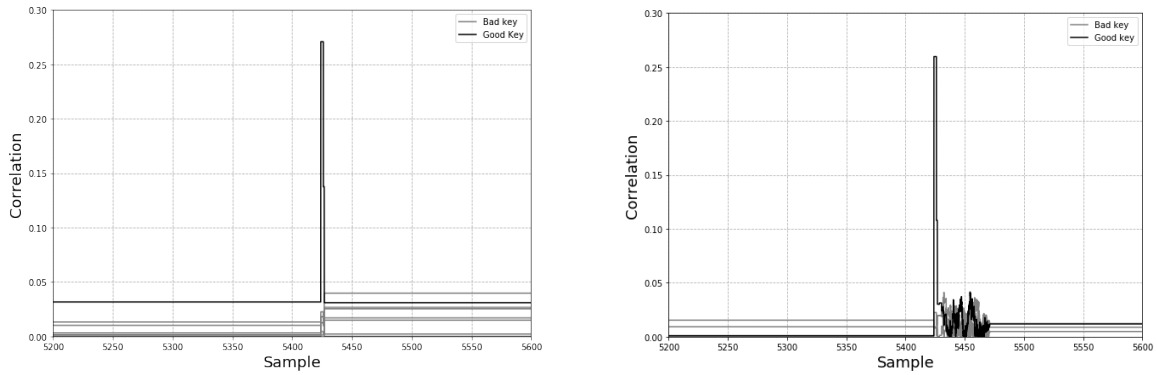
### 4.2.2 PS analyses

To perform analyses at PS level, we generate a netlist for a Xilinx FPGA. The aim is to reflect the behaviour of the circuit concerning the propagation times of logic gates, as explained in section 2.5.3.

#### 4.2.2.1 Mis-integration

To check that the synthesis does not make any simplification that could create a vulnerability, we perform this analysis in two steps. The first step consists in analyzing the netlist without the timing constraints. We force the synthesizer to keep the hierarchy of the design, as well as the internal signals of each module in the description language (Verilog source code). The results are therefore similar to those obtained with RTL simulation. Indeed, in this experience, the results do not indicate any vulnerable signals.

In the second step, when the propagation time information is added to the simulation, the analysis identifies many vulnerable signals, with a significant correlation with the leakage model for the correct key hypothesis. At first glance, this seems to be implausible, and it requires a great deal of investigation to track the source and the reason of this leakage.




(a) S-box input

(b) S-box output

Figure 4.2: CPA result on vulnerable signals. The score of the right key (black) is clearly distinguishable.

We computed the correlation on one input and one output signal of the S-box. The results are illustrated in fig. 4.2. The right key shows a very significant peak (plotted in black), compared to the result of bad key hypotheses (plotted in gray).

In listing 4.1 we have pinpointed the leaking code that caused the leakage. The signal $demask$ takes the value that allows unmasking the final ciphertext only at the last round, which is not supposed to be vulnerable.

Listing 4.1: Vulnerable code

```
unmask <= demaks when round = x"B" else (others => '0');
demasked_round_value <= round_value  xor unmask;
```

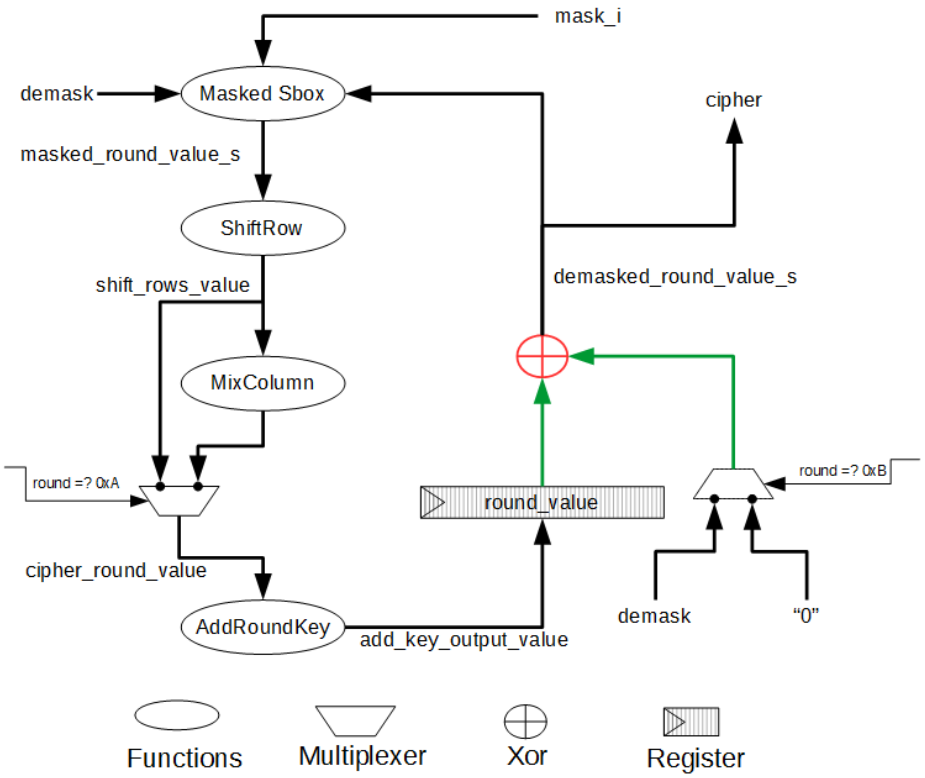



Figure 4.3: Simplified block diagram of the masked AES top module. The vulnerable signal is indicated in red color. The created leakage will be propagated through the next combinatorial functions.

The block diagram of the data-path of the implementation is illustrated in fig. 4.3. This block diagram is annotated with the correct signal name. Due to the delay caused by the propagation time, the signal connected to the register ($round\_value$) still has the previous value (*i.e.* the masked S-box input $S'_9$ ). As a consequence, when the signal $round$ reaches $11$ $(0xB)$, the signal connected to the S-box module will be automatically unmasked (for a short period of time), as illustrated in fig. 4.4 based on a time-annotated PS simulation. As we can see, the

signal $demasked\_round\_value\_s$ is updated several times. The leakage is induced on the first transition.

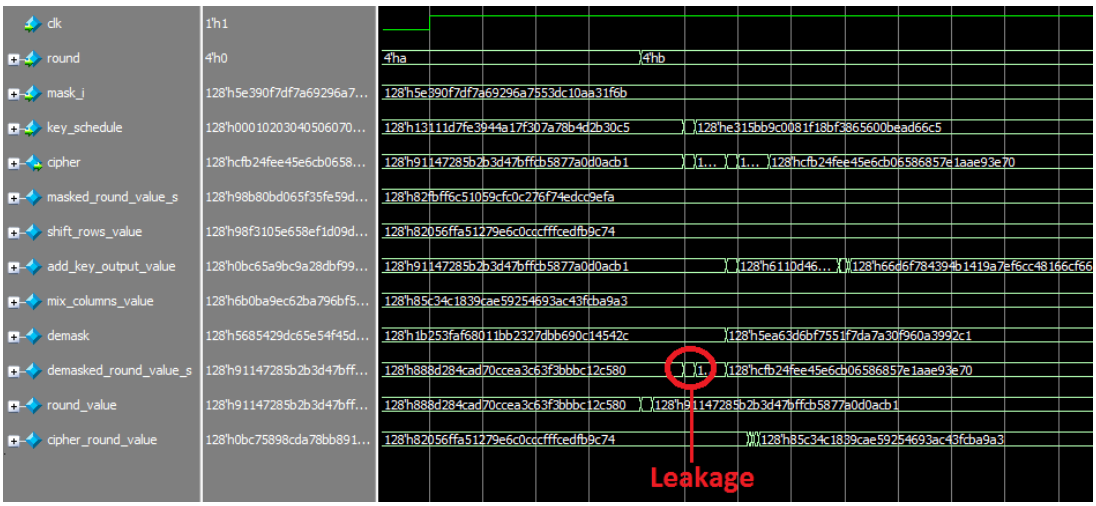



Figure 4.4: Post synthesis simulation illustrating how an unmasked value can be computed due to the propagation time.

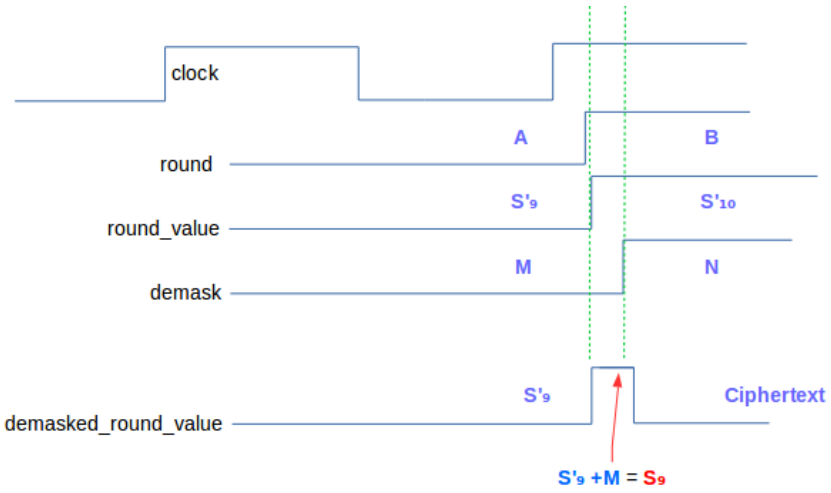



Figure 4.5: Diagram illustrating how an unmasked value can be computed due propagation time.

We illustrate more clearly in fig. 4.5 this leakage. We can observe how the previous value of the state register ($round\_value$) is $xored$ with the previous value of the mask, resulting an unmasked value through the signal $demasked\_round\_value\_s$.

#### 4.2.2.2 Correct integration

To fix this vulnerability, the mask signal of the intermediate rounds, and the mask signal of the last round has to be separated. To be more **rigorous** in our evaluation, we decide **to**

**completely remove the demasking** phase at the last round. Thus, the returned cipher is in fact masked. Since we place ourselves in a white-box evaluation, we can reconstruct the correct cipher knowing the mask. At this point, we can assume that no data can be unmasked internally.

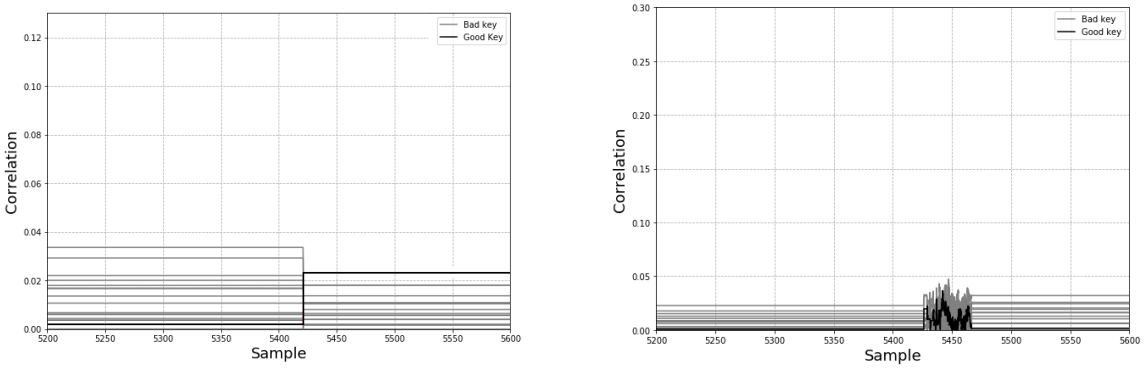
In listing 4.2 we give a way to fix this vulnerability. The mask allowing to unmask the final state is “shift_row_mask_i”. This signal does not allow to unmask the intermediate round values.

Listing 4.2: Fixed code

```
demasked_round_value_s <= round_value;
cipher <= demasked_round_value_s xor shift_row_mask_i
    when round = 0xB
    else demasked_round_value_s;
```

The mask of the intermediate rounds is actually “mc_shift_row_mask_i” which is the output of $MixColumns$ computed as:

$$mc\_shift\_row\_mask\_i = MixColumns(shift\_row\_mask\_i)$$




(a) S-box input (b) S-box output

Figure 4.6: CPA result on a vulnerable signal. The right key is not distinguishable.

Based on a signal level analysis as performed previously, we are able to confirm that no signal takes the sensitive value defined in eq. 4.1. The result of the CPA on the same signal previously identified as vulnerable in fig. 4.2 do not show any peak for the right key hypothesis, as shown in fig. 4.6. This analysis confirms that no signal takes an unmasked value, but does not show whether the design does not have any signal that is dependent on a secret value, or in other words, secure against a first-order analysis.

To go further in our analysis, we generate traces of the power consumption using digital simulations. These traces are constructed by considering the activity of the circuit (transitions) and the static state [114, 115, 116, 117]. This allows to detect the two kinds of leakages that are either related to the value of signals or to the activity. These results will be presented in the next section.

### 4.2.3 High-level leakage assessment

To check that the whole activity is not correlated with the sensitive value defined in eq. 4.2, we first perform a CPA using the HW, HD and mono-bit leakage model at the last round, by predicting the S-box input.

None of these attacks allow to distinguish the right key, and no significant peak is observable up to 25,000 traces. To check that no other leakages are present, we performed a more generic detection analysis with the NICV using the ciphertext. On a properly protected implementation, we expect a single peak corresponding to the ciphertext itself (after unmasking). However, in our analysis we identify two leaking points at two different rounds (10 and 9).

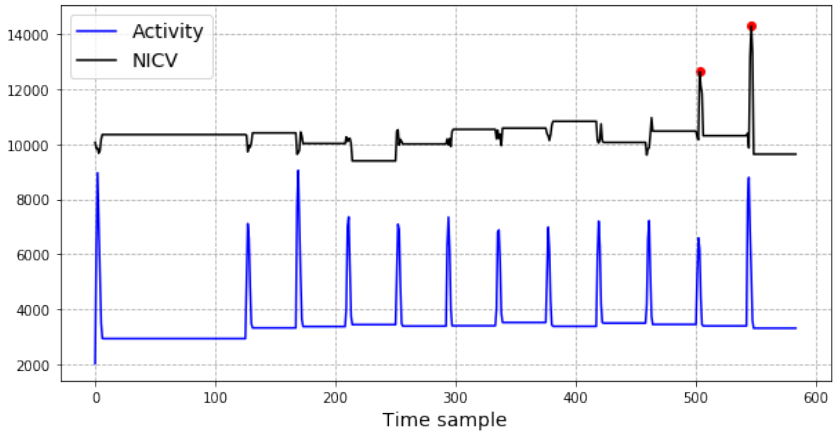



Figure 4.7: Superimposed NICV and a raw trace. A leakage is detected at round 9 and 10 identified by the red points. The NICV is scaled and shifted for clarity.

In fig. 4.7, we superimpose the result of the NICV and a simulated trace. We can notice two leaking points. The last one corresponds to the ciphertext, and the previous one corresponds to the state at round 9, which is therefore vulnerable. In the next sub-section, we will analyse the leakage with more sophisticated distinguisher, that does not make any assumption about the nature of the leakage, but it extract it from the power traces themselves.

### 4.2.4 Leakage exploitation - Collision attack

Collision attacks can be considered as profiling attacks, but without a clone device. As explained in section 2.2.3.2, the attacker can build the leakage model from traces of power consumption, by making a hypothesis on a small part of the key, (8 bits in this case). Besides, it assumes that all the S-boxes consume in the same way (i.e., the consumption of $sbox_i$ is equivalent to the consumption of $sbox_j$ in average). Here, we mean by $sbox_i$, the sub-circuit block that takes as input the byte $i \in \{0, \cdots, 15\}$ of the state.

Firstly, we present an analysis on the simulated traces at PS level, and verify that the attack works well. Secondly, we follow with the real traces (EM). And finally, we mix the two campaigns (simulated and EM) for a simple comparison.

#### 4.2.4.1 Simulated traces

Using the simulated traces at PS level, we compute the expectation of the power consumption of one S-box. For each value $x \in \{0, ..., 2^n - 1\}$ (where $n = 8$ in our case) of the ciphertext byte, we compute the leakage template $\mathcal{L}(x)$ as:

$$\mathcal{L}(x) = \mathbb{E}[\mathcal{T}_{i^*} | x] \tag{4.3}$$

where $i^*$ is the instant where the S-box is computed. We have performed the CPA using $\mathcal{L}$ as a leakage model.

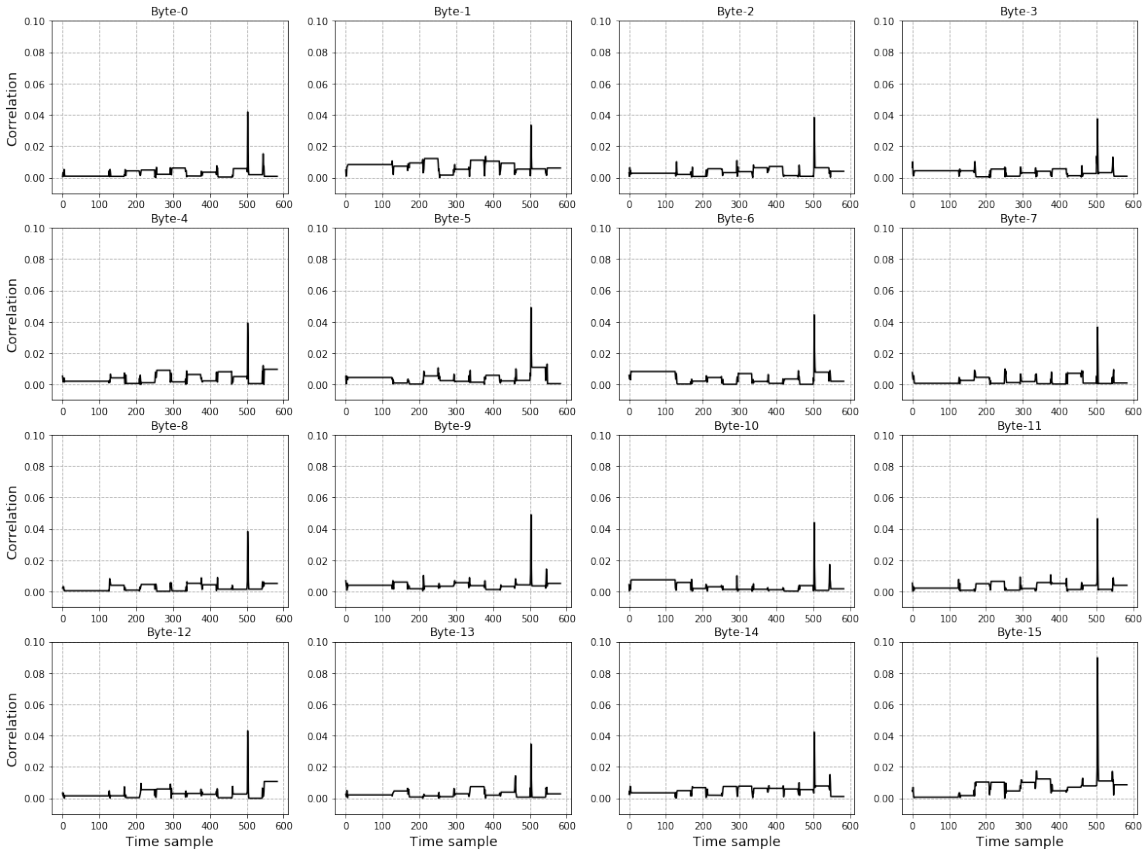



Figure 4.8: Result of the CPA using leakage model given by eq. 4.3. All key bytes are recovered.

Using only 25,000 traces, we are able to recover all bytes of the secret key (cf. Figure 4.8). Actually, this attack is equivalent to collision attack described in [36], where the leakage model is learned from the observations.

We note that to calculate the templates $\mathcal{L}(x)$, we used the byte $15$. This is the reason why it has a better correlation compared with the other bytes.

#### 4.2.4.2 EM traces

In order to make a complete analysis of our target, we implemented the fixed design on SAKURA-G FPGA board, and we acquired one million EM traces. First, we evaluate the robustness of this implementation against CPA, using standard leakage models (HW, HD and mono-bit). All the leakage models that we have tested, targeting the input of the S-box have failed to recover the correct key. We can conclude that the leakage is not correlated directly (linearly) with the sensitive data (at least based on 1,000,000 traces). Thus, we have constructed a new leakage model based on the EM traces, as in eq. 4.3.

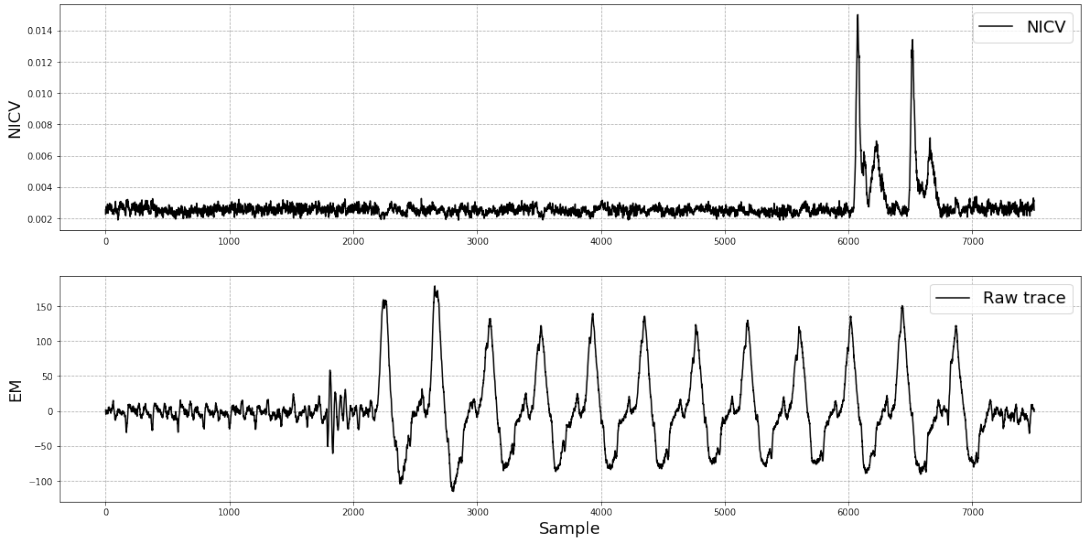
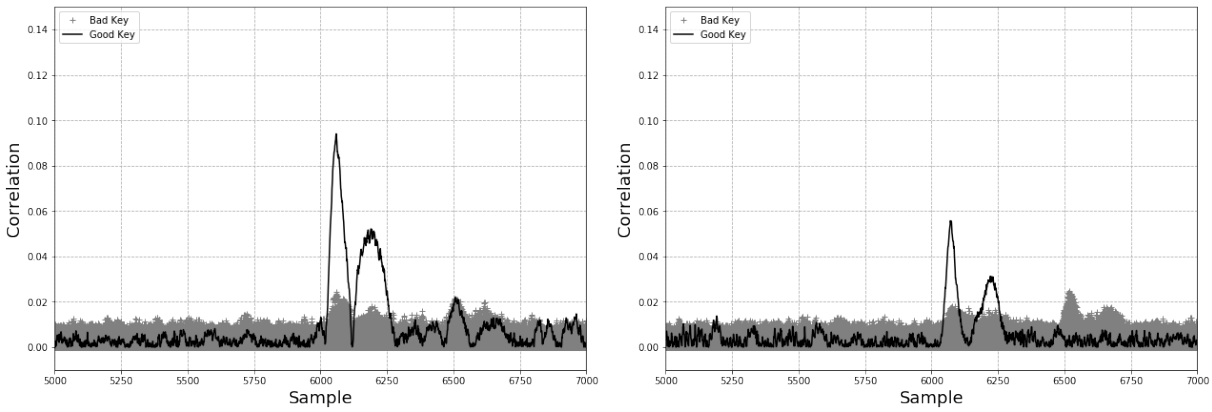



Figure 4.9: Raw trace superimposed with the NICV. Two leaking rounds are identified.

As in the case of simulated traces, we use the NICV to locate the leaking points. The result is shown in fig. 4.9. We also notice a leakage at round 10, and a second leakage on round 9. The latter is used to perform a correlation attack.

Using this leakage model, all bytes are successfully recovered. We have also used the leakage model extracted from the simulated traces, and the result was very similar. In fig. 4.10, we show an example of the CPA based on both leakage models, fig. 4.10a shows the result based on the leakage model extracted from the EM traces, and fig. 4.10b shows the result using the leakage model extracted from the simulated traces. The results are very similar and allow to recover the right secret key.

This observation shows that the simulated activity of the circuit based on the toggle count is correlated with the EM measurements. This shows that the activity of the simulated circuit is




(a) (b)

Figure 4.10: CPA result based on leakage model of eq. 4.3. The right key hypothesis is clearly distinguishable. (a) The leakage model is extracted from EM traces. (b) The leakage model is extracted from the simulated traces.

very close to that observed on the real one. As the design does not present any leakage in the absence of propagation time information, we can deduce that this vulnerability is linked to the extra-transitions (glitches), that are not required for the correct functioning of the circuit.

## 4.3 Discussion

In this work we presented an automated approach that can be integrated into EDA tools in order to perform security analysis at an early stage in the design and development environment. As such, it enables a secure-design approach where security requirements can be tracked and verified at each step of the design, following an interactive verification workflow from low abstraction level (signals and registers) to high abstraction level.

Unlike most of state-of-the-art proposed frameworks described in section 2.5, it also enables an exhaustive coverage of each and every step of the computation, including critical steps such as final unmasking and mask refreshing. In particular, the emphasis has been put on the assessment of the glitch induced vulnerabilities due to propagation delays in logic gates.

This observation puts forward the necessity of evaluating systematically the design against glitch vulnerabilities. This aspect will be addressed further in chapter 5 where we explain and detail an approach for formal evaluation methodology against glitches.

## 4.4 Conclusion

In this work we have presented an end-to-end analysis of a protected implementation with masking scheme. This study allowed us to detect several more or less obvious sources of

leakages. Designers are generally able to implement a masking scheme that conforms to the specification and the security properties at the behavioural level. Unfortunately, this is not enough to ensure the security of such a scheme against physical attacks. In particular, the propagation time in the logical gates prevents the sequential processing of the data. Indeed, this induces extra non-expected and non-essential calculations for the final result, but generates many leakages exploitable by an SCA. We have seen in the first example how a sensitive value of an AES encryption is in clear because of signal delays. From SCA point of view, this leakage is trivial, but for a designer, this computation is more difficult to predict and anticipate. This first example can be considered as a mis-integration, independent from the used S-box block.

In a second example, where the design does not present a trivial leakage at signal level, we did call for a higher level analysis. This analysis is based on an estimation of the power consumption (or activity) of the circuit and the leakage detection metric NICV. The latter makes it possible to identify all types of leakages; linear or non-linear, without leakage model unlike the classical CPA. After identifying the potential leakage samples, we build a leakage model extracted from the (simulated) power consumption traces. It allowed us to find the secret key. The same analysis could be reproduced on EM traces. A correspondence between numerical and real activity was also highlighted.

It is important to mention that the leakage shown in the first example does not depend on the used S-box, but on how the countermeasure is integrated. The reason of the second leakage has not been clarified at this point. We only know that the activity induced due to transitions (glitches) is correlated with the sensitive data. In the next chapter, we will study in depth the impact of propagation time on the circuit activity, and the reason why a first order leakage occurs.

# Chapter 5

# Formal Analysis of a Masked Implementation

Contents


5.1 Introduction . . . . . 61
5.2 Preliminaries . . . . . 62
5.3 Formalization of netlist static analysis . . . . . 64
5.4 Practical case: masked inversion in $GF_{2^4}$ . . . . . 73
5.5 Actual exploitation of vulnerable netlists . . . . . 79
5.6 Conclusion . . . . . 85


## 5.1 Introduction

Masking scheme ensure that sensitive variables are randomized. At hardware level, this property must hold true not only at each clock edge, but also during the signals propagation inside the combinatorial logic. Owing to the race of signals inside the combinatorial gates, intermediate values mixing previous and current states of signals may be computed. This phenomenon is termed “glitching”, and it has the negative property that those transitions depend on the sensitive (secret) value.

State-of-the-art protections against glitches either attempt to remove them to the point that no further leakage occurs through glitches, or to separate the combinatorial gates dealing with the masks and the masked data. Those two strategies ensure the absence of sensitive leakages thought glitches by a (conservative) design methodology [63, 67, 71].

In the present chapter, we will show that these methodologies are overkill. First, we formalize an algorithm to verify the absence of leakage despite glitches in arbitrary netlists. This algorithm checks that for all possible glitches configurations no sensitive information is leaked.

Then, we leverage this algorithm to validate the security of masked netlists which are optimized (with respect to gate count) compared to state-of-the-art glitch-resistant masking schemes. We exhibit examples of netlists smaller than state-of-the-art that do not follow the design principles of the state-of-the-art resistant logic styles. Tools have been proposed to check styles, and obviously, they report a leakage warning on our optimized designs, but we show that those are *false positives*: Our methodology allows for exact verification in such a way that it does not check for sufficient condition, *but it does check that each transition is properly "masked"*.

## 5.2 Preliminaries

In the following we assume that an attacker is able to predict a sensitive intermediate value, which depends on $n$ secret key bits ($n \leq 8$ or $n \leq 16$). In the case of AES, usually $n$ is set to $8$. To check the resistance of recent masked implementations against glitches, the authors characterise the power model based on the S-box input, then they use collision attacks to the key recovery step [118, 65, 66].

### 5.2.1 Notations

- We denote by $(\oplus)$ and $(*)$ the $XOR$ and $AND$ operations on Boolean variables (lowercase) or vectors (uppercase) respectively.
- To indicate the inputs $(A, M)$ of a gate output $S$ that implements a Boolean function $f$, we use functional notations $S = f(A, M)$.
- A delayed value of a signal is indicated by apostrophes ($S'$).
- When the intermediate value depends only on some (delayed) signals, it will be indicated on its arguments, (for example: $S' = f(A', M')$).
- In general, we use $X$ for the secret data, $M$ indicates the masks and $A$ the masked data $A = X \oplus M$.
- We suppose also that the masks are uniformly distributed and cannot be guessed by an attacker.
- The expression of an output gate $S$ can be expressed either with the tuple $(A, M)$ or $(X, M)$. To distinguish both, we index the latter with $X$. Thus, we have:

$$S = f_X(X, M) = f(X \oplus M, M) = f(A, M).$$

The evaluation of a given design will use both notations (or expressions) to determine which variable is leaking, and where the vulnerability is located.

### 5.2.2 Concepts

A formal based approach can be adopted to analyse the netlist by checking that all signals are independent from the secret data:

- For each gate output, deduce the corresponding Boolean expression $f$ from the netlist;
- Use some criterion of independence to ensure that $f_X$ is independent from the secret variable $X$. This criterion can be based on a full formal representation like in [82, 84], or on an exhaustive evaluation of the conditional probabilities $\mathbb{P}(f_X|X)$. This probability must be the same whatever the value taken by $X$.

In terms of value, this is enough to ensure that each signal is independent from the secret. However, in terms of transitions this is not sufficient. The vulnerabilities introduced by glitches are directly related to the leakage introduced by transitions within the same cycle. In our context we consider two sources of exploitable vulnerabilities:

- Value based vulnerability: when a signal value is not independent from the secret value.
- Transition based vulnerability: when the activity (or transitions) of the signal depends on the secret.

To check the first vulnerability, the authors in [119] presented a relation between the Walsh Transform (WT) and the statistical dependency of a Boolean function with its variables. In the following, $A = (a_0, \cdots, a_{n-1}), M = (m_0, \cdots, m_{n-1}), X = (x_0, \cdots, x_{n-1})$ are binary vectors, with $A = X \oplus M$.

**Definition 6** (WT (from [119])). Let $f$ be a Boolean function:

$$X = (x_0, \ldots, x_{n-1}) \mapsto f(X),\ GF_2^n \mapsto GF_2.$$

The Walsh Transform $F = WT(f)$ of $f$ is defined as:

$$GF_2^n \mapsto \mathbb{Z},\ \ F : W = (w_0, \cdots, w_{n-1}) \mapsto \sum_{X \in GF_2^n} f(X)(-1)^{W \cdot X}$$

*where*

$$W \cdot X = \bigoplus_{i=0}^{n-1} w_i * x_i$$

is the standard scalar product.

**Theorem 1** (Correlation immunity [119]). *The Boolean combining function $f$ for $n$ binary variables is $m^{th}$-order correlation immune, where $1 \leq m \leq n$ $iff$ its Walsh transform $F$ satisfies:*

$$\forall\, W \in GF_2^n,\ 1 \leq HW(W) \leq m,\ F(W) = 0.$$

**Corollary 1.** *A function $f_X$ is independent from $X$, if it is independent from each subset of the involved secret variables. In general, if $f_X$ is expressed as:*

$$f_X(x_0, \ldots, x_{n-1}, m_0, \ldots, m_{n-1})$$

*then, $f_X$ is independent from X $iff$:*

$$\forall\, W \in GF_2^{2n}, \quad F_X(W) = 0\ ,\ \textit{where}\ w_n = \ldots = w_{2n-1} = 0.$$

**Definition 7** (Security with respect to value)**.** A Boolean function $f(A, M)$ is secure in terms of value if it is independent from $X = A \oplus M$ (i.e it satisfies corollary 1).

This gives a spectral equivalent version to check if any Boolean function is statistically dependent on any set of secret variables. For example, if there exists $W = (1, 1, 0, \cdots, 0)$ such that $F(W) \neq 0$ then $f$ depends on $(x_0, x_1)$. Nevertheless, theorem 1 cannot be used directly to check if a given function is secure in terms of transitions. To take transitions into account, we need to consider two successive states of a signal.

**Definition 8** (Transition leakage)**.** We define the transition leakage as the distance between two successive values of a function $f$ by $D_{\delta_A,\delta_M}(f, A, M)$ for some $\delta_A$, $\delta_M \in GF_2^n$:

$$D_{\delta_A,\delta_M}(f, A, M) = f(A \oplus \delta_A, M \oplus \delta_M) \oplus f(A, M)$$

This distance is characterized by $\delta = (\delta_A, \delta_M)$, which is a bit-vector such that the bits at one indicate the delayed signals. The delayed variables are then, $A' = A \oplus \delta_A$ and $M' = M \oplus \delta_M$.

## 5.3 Formalization of netlist static analysis

In the following we give some examples to introduce our security verification methodology. Mainly we apply the notions described previously to analyse non-linear functions in presence of glitches. In section 5.3.1, we analyse the impact of a glitch at the netlist inputs, and in section 5.3.2, we extend this approach to netlist logic and give a complete formal model that proves security in presence of glitches. Finally, in section 5.3.7, we give a simple masked $AND$ gate based on this methodology.

### 5.3.1 Motivating examples

**Example 1.** Let $X = (x_0, x_1, x_2)$, $M = (m_0, m_1, m_2)$ and $A = X \oplus M$, and $f$ defined as:

$$f(A, M) = (m_2 \oplus a_0 * m_1) \oplus a_1 * m_0$$

which is a part of the masked $AND$ gate described in [2]. We can easily check that $f$ is uniformly distributed in terms of value and independent from $X$ (i.e $\mathbb{P}(f = 1|X) = \frac{1}{2}$).

However, in the case of a transition when $\delta_A = (0, 1, 0)$ and $\delta_M = (0, 1, 0)$, we get:

$$D_{\delta_A,\delta_M}(f, A, M) = a_0 \oplus m_0 = x_0$$

which depends on $X$. Thus, this implementation is vulnerable in terms of transition, and may leak in presence of glitches. Besides, it leaks $x_0$ only if the timing characteristics of the device are such that the couple $(a_1, m_1)$ arrives later than other signals, and that the transitions $(a_1, m_1) \rightarrow (a'_1, m'_1)$ arrive almost at the same time at the inputs of the last $XOR$ gate (red color).

Another interesting case is when $\delta_A = (1, 1, 0)$ and $\delta_M = (1, 1, 0)$:

$$D_{\delta_A,\delta_M}(f, A, M) = x_0 \oplus x_1.$$

In this case, the leakage is not correlated to the HW of $X$. This is in fact the general form of the leakage created by glitches when the multi-linear polynomial of the gate is of degree 2. In this configuration, it results in the WT of the function $f$ that $F_X(W) \neq 0$ for $W = (1, 1, 0, \cdots, 0)$. The same holds actually for the traces of power consumption (see section 5.5).

**Example 2.** Here we consider another case that involves also both shares of the same variable. We have: $X = (x_0, x_1, x_2, x_3)$, $M = (m_0, m_1, m_2, m_3)$, $A = X \oplus M$, and:

$$f(A, M) = a_0 * (m_1 \oplus m_2) \oplus a_1 * (m_0 \oplus m_3).$$

By analyzing all the possible transitions $\forall\ \delta_A, \delta_M \in GF_2^4$, we have always $f_X$ independent from $X$. We can see that if both $(m_1, m_2)$ (or $(m_0, m_3)$) change, $f$ do not change, so the number of transitions to compute can be reduced. The relevant value of $f'$ are:

$$f(A \oplus \delta_A, M \oplus \delta_M) = (a_0 \oplus \delta_0) * (m_1 \oplus m_2 \oplus \delta_1) \oplus (a_1 \oplus \delta_2) * (m_0 \oplus m_3 \oplus \delta_3)$$

with $\delta_i \in GF_2$, and $\delta_0 = \delta_{a_0}$, $\delta_2 = \delta_{a_1}$, $\delta_1 = \delta_{m_1} \oplus \delta_{m_2}$, $\delta_3 = \delta_{m_0} \oplus \delta_{m_3}$.

If $\delta_A = (0, 1, 0, 0)$ and $\delta_M = (0, 1, 0, 0)$ then:

$$D_{\delta_A,\delta_M}(f, A, M) = x_0 \oplus m_3$$

which is uniform and independent from $X$. We can deduce that this is secure even in presence of glitches, but according to [82, 84] this function is not secure, because $f$ uses all shares of the secret variables $x_0$ and $x_1$.

### 5.3.2 Formal model - Glitch-extension

In a combinatorial block, a glitch can be generated because of a single transition (without a glitch) on a single signal. Indeed, when the output of the block is driven by several gates which take the same input signal, the final output can be impacted several times. To take into account this behaviour, we introduce the notion of transient input.

**Definition 9** (Transient input)**.** The transient input $(\tilde{A}, \tilde{M})$ of a function $f$ is the set of all variables that comes from different combinatorial paths. Thus, the copies of same variable at different places are considered independently. We denote also by $\tilde{f}$ the transient expression of $f$. Thus, we have: $\tilde{f}(\tilde{A}, \tilde{M}) = f(A, M)$.

To understand clearly this notion, it is more appropriate to use the notation based on trees expression, namely the prefix traversal.

**Definition 10** (Prefix traversal)**.** A prefix representation is the expression produced when placing the operator first and the two operands next.

In example 2, $f$ can be expressed as:

$$f(A, M) = \oplus(*(a_0, \oplus(m_1, m_2)), *(a_1, \oplus(m_0, m_3))) \tag{5.1}$$

The advantage of this notation lies in the fact that it preserves the structure of the function instantiation in the physical logic gates of the circuit. For instance, eq. 5.1 can be instantiated equivalently by:

$$f(A, M) = \oplus(\oplus(*(a_0, m_1), *(a_0, m_2)), \oplus(*(a_1, m_0), *(a_1, m_3)))$$

which gives different leakage results when considering glitches. More precisely, each leaf of the tree is considered independently in the case of transitions, thus different $\delta$ could be associated to the same variable.

**Proposition 1** (Security with respect to transition (Glitch-Extended Security))**.** *Let $f$ be the expression of the signal $S$, and $\tilde{A}, \tilde{M} \in GF_2^{\tilde{n}}$ the transient input variables of $S$.*

*$S$ is secure against glitches $iff$ for any $\delta_a, \delta_m \in GF_2^{\tilde{n}}$:*

$$D_{\delta_a, \delta_m}(\tilde{f})(\tilde{A}, \tilde{M})$$

*is statistically independent from $X = A \oplus M$.*

*Proof.* In fact, $D_{\delta_a, \delta_m}(\tilde{f})(\tilde{A}, \tilde{M})$ is a Boolean function therefore, theorem 1 and corollary 1 apply directly. □

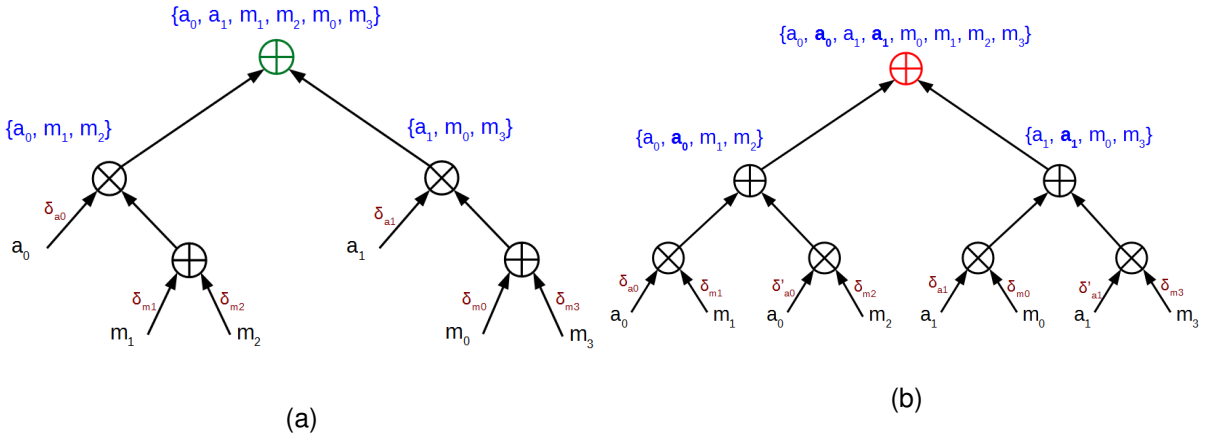



Figure 5.1: Different ways to implement $f$ of example 2. Transient inputs for each gate are shown in blue: in (a) $(\tilde{A}, \tilde{M}) = ((a_0, a_1), (m_0, m_1, m_2, m_3))$, (b) $(\tilde{A}, \tilde{M}) = ((a_0, a_0, a_1, a_1), (m_0, m_1, m_2, m_3))$. For each input we associate different delay ($\delta$).

In fig. 5.1, we show the two different ways to implement $f$ introduced in example 2. In the first case (a), $f$ is not leaking $X$. In the second case (b), $f$ may leak $X$. The reason is that, the variable $a_0$ (resp. $a_1$) may impact the function $f$ differently (at two different time samples).

### 5.3.3 Leakage detection algorithm

Algorithm 1 scans the netlist (input $S$) and first checks whether each node is masked, and then tests whether it is vulnerable to glitches. If a configuration yields an unbalanced distribution, then the algorithm returns the corresponding $\delta$ and the leaking signal. The internal functions work as follows:

- get_transient_inputs: returns the inputs of each gate as instantiated in the design (definition 9).
- get_masked_variables: returns the masked variables of the input gate.
- get_masks_variables: returns the masks variables of the input gate.

If no transition depends on the sensitive variable $X$, then algorithm 1 returns “Secure”.

We insist that this verification methodology is agnostic in the actual quantitative delays within the netlist, because we abstract away the glitching source as an anticipated evaluation anywhere in the netlist. Our threat model is that the netlist is known, represented as a tree of gates, and is immutable. The attacker can well probe a node to artificially load it exaggeratedly (in an adversarial view to create a long delay path), but cannot alter the netlist by cutting wires or disabling gates.

We show in table 5.1 a comparison of our approach with other existing formal analysis methods. In the following sub-section we will build a masked $AND$ gate secured against glitches,

**Algorithm 1:** Security Verification Against Glitches

**Input:** $S$: The design, $A$: List of masked variables, $M$: List of mask variables
**Output:** "Secure" or first leaking signal

**for** $s \in S$ **do** `// For each signal` $s$ `in the netlist`
  $transient\_inputs\_of\_s \leftarrow get\_transient\_inputs(s)$
  $n \leftarrow \|transient\_inputs\_of\_s\|$
  $M_s \leftarrow get\_masks\_variables(transient\_input\_of\_s, M)$
  $A_s \leftarrow get\_masked\_variables(transient\_input\_of\_s, A)$
  $X \leftarrow A_s \oplus M_s$
  $f \leftarrow s(inputs\_of\_s)$
  $f_X \leftarrow f(X \oplus M_s, M_s)$
  $value\_distribution \leftarrow \mathbb{P}(f_X|X)$ `// Security in terms of value`
  **if** *value_distribution is not balanced* **then**
    **return** $s$ `// First order leaking signal` $s$ `in terms of value`
  **for** $\delta \in GF_2^n$ **do** `// Security in terms of transition`
    $f' \leftarrow s(transient\_inputs\_of\_s \oplus \delta)$
    $T \leftarrow f \oplus f'$ `//` $T$ `is the transition`
    $T_X \leftarrow T(X \oplus M_s, M_s)$
    $distribution \leftarrow \mathbb{P}(T_X|X)$
    **if** *distribution is not balanced* **then**
      **return** $s, \delta$ `//` $s$ `being the leaking signal, and` $\delta$ `indicating the delayed signal`

**return** "Secure"

Table 5.1: Comparison with state-of-the-art formal analysis methods

| Analysis method | Leakage location | Value leakage model | Exact transient leakage | Formal leakage expression |
|---|---|---|---|---|
| Barthe et al. [84] | ✓ | ✓ | ✗ | ✗ |
| Bloem et al. [82] | ✓ | ✓ | ✗ | ✗ |
| This work | ✓ | ✓ | ✓ | ✓ |

based on the previous observation made in section 5.3.2. This version does not have a major advantage over TI, but we will use the same principle for more concrete cases where the difference is more significant in terms of area (section 5.4).

### 5.3.4 Analysis of the masked *AND* gate

The shared output of the $AND$ gate needs to combine the different shares of each variable. In the following we show how a leakage is created on more realistic examples. Let's consider the classical multiplier presented in [2] and shown in Figure 5.2. To compute the shared result of

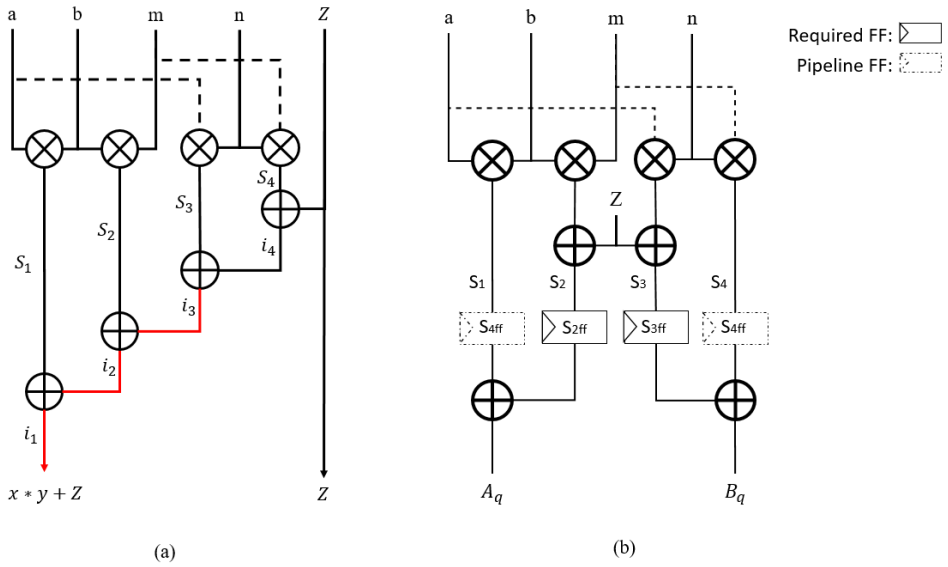



Figure 5.2: Masked $AND$ gates. (a)[2] leaking signals are in red color, (b) First order secure DOM $AND$ [71].

$x * y$, we compute the $AND$ result between each independent shares:

$$s_1 = a * b, \quad s_2 = b * m, \quad s_3 = a * n, \quad s_4 = m * n \tag{5.2}$$

It is easy to check that:

$$x * y = s_1 \oplus s_2 \oplus s_3 \oplus s_4 \tag{5.3}$$

A new fresh random $z$ is required to mask the output, and the result is computed in the following order:

$$((z \oplus s_4) \oplus s_3) \oplus s_2) \oplus s_1) = x * y \oplus z$$

which is secure at the algorithmic level, because each intermediate result is independent from $(x, y)$.

The order of summation is clearly important. If we compute:

$$s = s_1 \oplus s_2 = a * b \oplus a * n = a * (b \oplus n) = a * y \tag{5.4}$$

we have $\mathbb{P}(s|y) \neq \mathbb{P}(s)$ (in other words, the distribution of $s$ is not independent from y). In fact, any computation of $(s_i \oplus s_j)$ with $i \neq j$ will depend on $x$ or $y$.

### 5.3.5 Masked *AND* with propagation time

Even if we consider the circuit at gate level, the result $s_i \oplus s_j$ is never computed. To explain why a leakage is created, we should consider the case where a signal is delayed. For example, if

$m$ arrives with some delay, the circuit will compute the intermediate result $R' = R(a, b, m', n, z)$ with $m'$, and then update with $m$. Formally we have:

$$R' = ((z \oplus s'_4) \oplus s_3) \oplus s'_2) \oplus s_1)$$
$$R = ((z \oplus s_4) \oplus s_3) \oplus s_2) \oplus s_1)$$

In terms of each intermediate value, the dependency with the secret do not hold. However, in terms of transition ($R' \rightarrow R$), which can be modelled as $R' \oplus R$ we get:

$$\begin{aligned} R' \oplus R &= s_4 \oplus s'_4 \oplus s_2 \oplus s'_2 \\ &= m' * n \oplus m * n \oplus m' * b \oplus m * b \\ &= (m' \oplus m) * n \oplus (m' \oplus m) * b = b \oplus n \\ &= y \end{aligned} \tag{5.5}$$

As $m$ is delayed (and we suppose that it changes, $\delta_m = 1$), we have: $m \oplus m' = 1$. Thus, we deduce that the leakage (in terms of transition) gives exactly $y$.

We can consider other scenarios and deduce the formal expression of the leakage in terms of $x$ or $y$. For example, if $n$ is delayed then, the leakage will give $x$. We note that the leakage can be modelled as in eq. (5.5) only if we suppose that the transitions $s'_2 \rightarrow s_2$ and $s'_4 \rightarrow s_4$ arrive almost at the same time (within the propagation time of the corresponding $XOR$ gate). In [55], this was referred as the absorbed transitions. As we can see in the formal expressions of the intermediate results, the two shares of $y$ ($b$ and $n$) were multiplied by $m$. Thus, if $m$ is delayed, the circuit will leak exactly $(n \oplus b = y)$.We illustrate in fig. 5.2 the masked $AND$ gate of [2], presented in section 2.3.3 and the DOM version from [71]. The leaking signals $\{i_1, i_2, i_3\}$ are highlighted in red color.

Unfortunately, it is not the only way for this multiplier to leak. If we suppose that all signals $\{a, b, n, m, z\}$ arrive at the same time, the multiplier may still leak. Let's consider the intermediate signal $i_3$ that compute: $i_3 = (z \oplus s_4) \oplus s_3)$. In the case where $i_3$ is evaluated twice with the same value $Z$, the transition will depend on $y$ if the new values of $s_3$ and $s_4$ affects the signal $i_3$ at the same time.

In fact, the transition $i_3(a', n', z) \rightarrow i_3(a, n, z) \equiv i'_3 \oplus i_3$ is not balanced in terms of $y$ (we have, $\mathbb{P}(i'_3 \oplus i_3 | y = 0) = \frac{1}{3}$). We can deduce that the new fresh random $z$ has no protection effect against transitions. In general, the summation of $s_i \oplus s_j$ leaks in terms of value, but also in terms of transition.

### 5.3.6 Securing the masked *AND* gate

In the first case, the cause of the leakage was the delay of the signal that is multiplied by both shares of the same secret. The impact of the generated glitches is seen from the gate that manipulates both shares. In this case, the circuit leaks in $50\%$ of the time ($\mathbb{P}(m \oplus m' = 1) = \frac{1}{2}$). The second leakage is related to the structure of the function itself. We have mentioned before, that any summation of $s_i \oplus s_j$ will leak the secret data (in terms of value and also in terms of transition). The value of each signal $i_k$ can be written as:

$$i_k(s_k, s_{k-1}, i_{k-2}) = s_k \oplus s_{k-1} \oplus i_{k-2}$$

If $i_k$ is evaluated twice with the same value $i_{k-2}$ and the transition $(s'_k, s'_{k-1}) \rightarrow (s_k, s_{k-1})$ is seen almost at the same time by $i_k$, the leakage will be equivalent to the case where $s_k \oplus s_{k-1}$ is computed separately. In this case, the circuit leaks less than the first case, as more inputs should be changed (at least two inputs should change i.e. 25%).

The masked $AND$ gate can be secured with different ways:

- Keeping four shares for the output results [120]: This will increase exponentially the number of needed gates in a concrete circuits. In [63], the author proposed the (TI) structure based on three shares to make a first order secure multiplier.
- Insertion of delay elements: This should prevent the evaluation of each $XOR$ gate with two new $s_i$ and $s_j$ at the same time. If the circuit evaluates the result for each $s_i$ sequentially, each transition will be independent from $x$ and $y$, and the circuit will be secure against first order analysis. This is relatively easier than equalizing the time arrival of signals. An evaluation of this countermeasure is exposed in appendix A.1 on simulated traces at PS level.
- Insertion of registers: Even after registering the signal $s_i$ we should not sum them directly for the same reason mentioned in section 5.3.6. Thus, we need to remask each registers with a new random values $z_i$ for each $s_i$, then it can be securely summed and the output mask will be:

  $$z = \bigoplus_{i=1}^{4} z_i$$

  However, this requires a lot of fresh random bits. The DOM approach allows the usage of only one new fresh random to remask only two signals $s_i$, but the output masks cannot be controlled as it involves different shares of the inputs.

### 5.3.7 Our glitch-resistant masked AND gate

In this sub-section, we will present another way to secure the masked AND gate. This is based on the results of the preliminary study given in section 5.3.

Let $a_i = x_i \oplus m_i$ and $b_i = y_i \oplus n_i$ for $i \in \{0, 1\}$. In table 5.2, we give the different steps

Table 5.2: Masked implementation of $AND$ gate. $z_i$ are fresh random. Left: we have $x_0 * y_0 = i_4 \oplus z_0$; Right: we have $x_0 * y_0 = T_1 \oplus T_2 \oplus T_3$.

| Vulnerable masked $AND$ [2] | Our secure masked $AND$ |
|---|---|
| $s_1 \leftarrow a_0 * b_0$ | $s_1 \leftarrow (n_0 \oplus z_0)$ |
| $s_2 \leftarrow a_0 * n_0$ | $s_2 \leftarrow (m_0 \oplus z_1)$ |
| $s_3 \leftarrow b_0 * m_0$ | $s_3 \leftarrow a_0 * s_1$ |
| $s_4 \leftarrow m_0 * n_0$ | $s_4 \leftarrow b_0 * s_2$ |
| $i_1 \leftarrow z \oplus s_1$ | $i_1 \leftarrow b_0 \oplus z_0$ |
| $i_2 \leftarrow i_1 \oplus s_2$ | $i_2 \leftarrow a * i_1$ |
| $i_3 \leftarrow i_2 \oplus s_3$ | $i_3 \leftarrow b * z_1$ |
| $i_4 \leftarrow i_3 \oplus s_4$ | $T_1 \leftarrow s_2 \oplus s_4$ |
| | $T_2 \leftarrow i_2 \oplus i_3$ |
| | $T_3 \leftarrow m_0 * n_0$ |

for implementing the masked $AND$ gate. The left one is not secure. The right one satisfies our security model against glitches, and also in terms of value. To check that, let's consider a non-linear function $f$, defined as:

$$f(A, B, M, N, Z) = \oplus(*(a_0, \oplus(n_0, z_0)), *(b_0, \oplus(m_0, z_1))). \tag{5.6}$$

We can see that both shares of the secret $(x_0, y_0)$ are manipulated by $f$. We have seen in section 5.3.2 that $f$ is secure according to algorithm 1. Particularly, as a case of comparison with the classical masked $AND$ gate when:

$$(a_0, b_0, m_0, n_0) \rightarrow (a_0 \oplus 1, b_0 \oplus 1, m_0 \oplus 1, n_0 \oplus 1)$$

we have:

$$(i'_1) \rightarrow (i_1) \equiv x_0 \oplus y_0$$

and for $f$ in eq. 5.6 we get (with $f_0 = a_0 * (n_0 \oplus z_0)$ and $f_1 = b_0 * (m_0 \oplus z_1)$):

$$(f'_0 \oplus f'_1) \rightarrow (f_0 \oplus f_1) \equiv x_0 \oplus y_0 \oplus m_1 \oplus n_1,$$

which is not vulnerable. Whatever the considered transition, either the result depends only on one share of $(x_0, y_0)$, or it is remasked by $n_1$ or $m_1$. In other words, each transition is masked at least with one mask $n_i$ or $m_i$.

Thus, TI Non-completeness Property (TINC) is not necessary. Based on this result, we can implement a masked $AND$ gate (without any resharing of the inputs) using two fresh random $z_0$ and $z_1$ (we can reuse masks of other variables to reduce the usage of randomness):

$$\begin{array}{rcl} T_1 & = & a_0 * (n_0 \oplus z_0) \oplus b_0 * (m_0 \oplus z_1), \\ T_2 & = & a_0 * (b_0 \oplus z_0) \oplus b * z_1, \\ T_3 & = & m_0 * n_0 \end{array} \tag{5.7}$$

The output result is $x_0 * y_0 = T_1 \oplus T_2 \oplus T_3$. Incidentally, we can see that $T_1$ satisfies proposition 1 (cf. example 2), $T_2$ and that $T_3$ satisfy the TINC property.

## 5.4 Practical case: masked inversion in $GF_{2^4}$

We now design a complete implementation of a $GF_{2^4}$ inverter, based on the Canright version of the AES S-box masked at first-order. In section 5.5, we give the full implementation of our S-box, integrating our $GF_{2^4}$ inverter. In the same section, we compare our formal results with the results of digital simulation at RTL and PS levels.

For the sake of clarity, we focus our analysis on the $GF_{2^4}$ inverter. The results can be transposed to the operations performed in $GF_{2^8}$ inverter. Based on the simulation results and the formal expression of each signal, we will explain how the leakage is created, propose a possible fix, and constantly check the security of the design until no leakage is reported.

### 5.4.1 Canright AES S-Box

As already presented in section 2.3.3, Canright proposed an optimized instance of the AES S-box [121] based on standard CMOS gates $XOR$, $NOR$ and $NAND$. The inversion is computed based on the Tower Field representation of the element of $GF_{2^8}$. The inversion of an element in $GF_{2^8}$ can be reduced to one inversion in $GF_{2^4}$, some multiplications and additions in $GF_{2^4}$ and $GF_{2^2}$.

This implementation takes the masked input, the input mask, and the output mask, 8 bits each. We can find symmetry in the operations performed in $GF_{2^4}$ inside the $GF_{2^8}$ inverter, and those performed in $GF_{2^2}$ inside the $GF_{2^4}$ inverter. Thus, the $GF_{2^4}$ inverter takes, 3 inputs of 4 bits (masked input, input mask and output mask).

### 5.4.2 Formal based evaluation of Canright inverter

If we explicitly write the expression of the inputs of $csa$ gate, as illustrated in fig. 5.3, we get (for one bit, namely bit number 1):

$$
\begin{aligned}
an_1 &= (a_1 * n_1) \oplus ((a_0 \oplus a_1) * (n_0 \oplus n_1)) \\
mb_1 &= (m_1 * b_1) \oplus ((m_0 \oplus m_1) * (b_0 \oplus b_1)) \\
cst_1 &= a_1 \oplus b_1 \oplus a_1 * b_1 \oplus (a_1 \oplus a_0) * (b_1 \oplus b_0) \oplus N_3 \\
csa_1 &= cst_1 \oplus an_1, \\
csb_1 &= csa_1 \oplus mb_1
\end{aligned}
$$

where $N_3$ is a fresh mask (one bit of the output mask). These equations are also represented as a netlist in fig. 5.3.

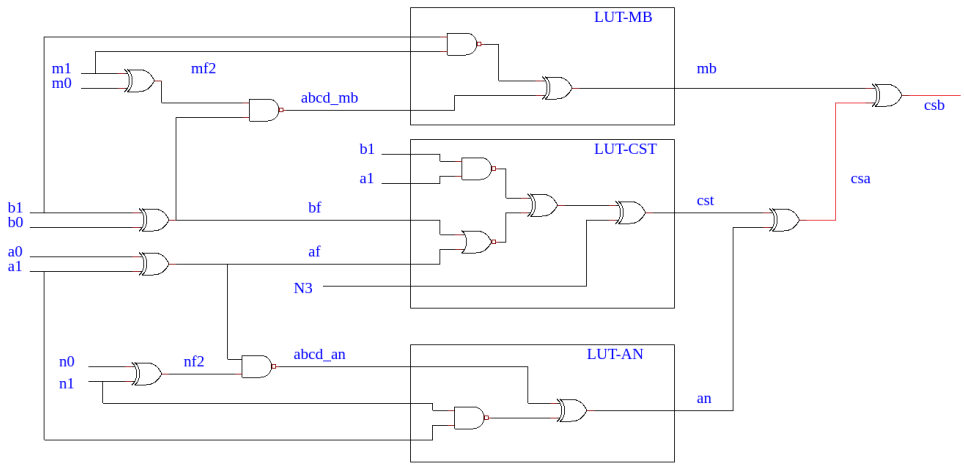



Figure 5.3: Masked circuit computing $csb_1$. The leaking signals (red color) are $csa_1$ and $csb_1$.

The order of summation is also important, if $an_1$ and $mb_1$ are summed together, the result will depend on $X$:

$$
\begin{aligned}
an_1 \oplus mb_1 = &(x_2 * n_0 \oplus x_2 * n_1 \oplus x_3 * n_0) \\
&\oplus (x_0 * m_0 \oplus x_0 * m_1 \oplus x_1 * m_0) = S_{ab}
\end{aligned}
$$

Obviously, $\mathbb{P}(S_{ab}|X) \neq \mathbb{P}(S_{ab})$, particularly for $X = 0$, we have $S_{ab} = 0$ with probability $1$. Now, let us consider the case where all signals are summed in the right order. For example, the signal $csa$ ($csa = cst \oplus an$). For the first bit, we have:

$$
csa_1 = a_0 * (b_0 \oplus b_1) \oplus a_1 * b_0 \oplus a_1 \oplus b_1 \oplus a_1 * n_0 \oplus a_0 * (n_0 \oplus n_1)) \oplus N_3.
$$

In terms of value, the result is protected (at least) by the fresh mask $N_3$. However, in terms of transition in presence of propagation time, $a_0$ can arrive with some delay and the transition ($csa'_1 \rightarrow csa_1$) will depend on $X$ and leak ($x_0 \oplus x_1$), according to algorithm 1 when $\delta_{a_0} = 1$:

$$\begin{aligned} csa'_1 \oplus csa_1 &= (a'_0 \oplus a_0) * (b_0 \oplus b_1 \oplus n_0 \oplus n_1) \\ &= (a'_0 \oplus a_0) * (x_0 \oplus x_1) \\ &= x_0 \oplus x_1 \end{aligned}$$

This depends on $x_0$ and $x_1$, hence **the Canright design is not secure**. Note however, that this leakage model is not a conventional. Only a though analysis and dedicated attacks can exploit this kind of leakage, such as collision or template. Actual exploitation of this first-order flaw is detailed in section 5.5.1.

### 5.4.3 Our $GF_{2^4}$ inverter - Compact and provably secure

As we have seen previously, to secure this implementation against glitches, it is necessary to redesign mainly the non-linear functions. In the following, we show how the inversion can be achieved within only one cycle. Then, using the observation of eq. 5.7, we reduce the number of needed registers (FF).

#### 5.4.3.1 Inversion in $GF_{2^4}$

First, we express the equations of the inverse $y = (y_0, \ldots, y_3)$ of any element $x = (x_0, \ldots, x_3) \in GF_2^4 \simeq GF_{2^4}$:

$$\begin{aligned} y_0 &= x_1 * x_2 * x_3 \oplus x_0 * x_2 \oplus x_0 * x_3 \oplus x_1 * x_3 \oplus x_2 \\ y_1 &= x_0 * x_2 * x_3 \oplus x_0 * x_3 \oplus x_1 * x_3 \oplus x_2 \oplus x_3 \\ y_2 &= x_0 * x_1 * x_3 \oplus x_0 * x_2 \oplus x_1 * x_2 \oplus x_1 * x_3 \oplus x_0 \\ y_3 &= x_0 * x_1 * x_2 \oplus x_1 * x_2 \oplus x_1 * x_3 \oplus x_1 \oplus x_0 \end{aligned}$$

The masked result can be deduced by replacing $x_i$ by $a_i \oplus m_i$. For the first bit $y_0$ we get:

$$y_0 = S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus S_5 \oplus S_6 \oplus S_7 \oplus S_8 \tag{5.8}$$

with,

$$
\begin{aligned}
S_1 &= a_2 * a_3 * a_1 \oplus a_2 * a_0 \oplus a_3 * a_1, \\
S_2 &= a_2 * a_3 * m_1 \oplus a_2 * m_0 \oplus a_3 * m_0 \\
S_3 &= a_2 * a_1 * m_3 \oplus a_0 * m_3 \oplus a_2, \\
S_4 &= a_2 * m_3 * m_1 \oplus a_3 * m_1 \\
S_5 &= a_3 * a_1 * m_2 \oplus a_0 * m_2, \\
S_6 &= a_3 * m_2 * m_1 \oplus a_3 * a_0 \oplus m_2 \\
S_7 &= a_1 * m_2 * m_3 \oplus a_1 * m_3 \oplus m_3 * m_0, \\
S_8 &= m_2 * m_3 * m_1 \oplus m_2 * m_0 \oplus m_3 * m_1
\end{aligned}
$$

We can see that each result $S_i$ respects the TINC. Moreover, as each monomial of degree 3 cannot be combined with any other monomial of degree 3, the minimal number of shares that respect TINC will be $8$. To achieve the inversion in one cycle, $8$ FFs and $8$ fresh random are needed to remask each $S_i$. We note that each $y_i$ can be expressed in the same way as eq. 5.8. See appendix A.2 for more details about the complete masked $GF_{16}$ inverter.

We describe in the following, how those equations can be compressed with fewer registers.

#### 5.4.3.2 Reducing the number of registers

To reduce the number of needed FFs, we need to optimize the masked computation of monomials of degree 3. For $y_0$ we have:

$$
\begin{aligned}
x_1 * x_2 * x_3 &= (a_1 \oplus m_1) * x_2 * x_3 \\
a_1 * x_2 * x_3 &= a_1 * (a_2 * a_3 \oplus a_2 * m_3 \oplus a_3 * m_2 \oplus m_2 * m_3) \\
&= a_1 * (a_2 * (m_3 \oplus z_0) \oplus a_3 * (m_2 \oplus z_1)) \\
&\oplus a_1 * (a_2 * (a_3 \oplus z_0) \oplus a_3 * z_1) \oplus a_1 * m_2 * m_3.
\end{aligned}
$$

The same thing holds for $m_1$. Thus, we reduce the number of needed FFs to 6. Finally, the masked computation of the LSB of the inverse in $GF_{16}$ is implemented as:

$$
y_0 = S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus S_5 \oplus S_6 \tag{5.9}
$$

with,

$$S_1 = a_1 * (a_2 * (m_3 \oplus z_0) \oplus a_3 * (m_2 \oplus z_1))$$
$$S_2 = m_1 * (a_2 * (m_3 \oplus z_0) \oplus a_3 * (m_2 \oplus z_1))$$
$$S_3 = a_1 * (a_2 * (a_3 \oplus z_0) \oplus a_3 * z_1 \oplus a_3) \oplus a_0 * (a_2 \oplus a_3) \oplus a_2$$
$$S_4 = m_1 * (a_2 * (a_3 \oplus z_0) \oplus a_3 * z_1 \oplus a_3) \oplus m_0 * (a_2 \oplus a_3)$$
$$S_5 = a_1 * (m_2 * m_3 \oplus m_3) \oplus a_0 * (m_2 \oplus m_3)$$
$$S_6 = m_1 * (m_2 * m_3 \oplus m_3) \oplus m_0 * (m_2 \oplus m_3) \oplus m_2$$

Note that each signal $S_i$ satisfies corollary 1 and proposition 1 and hence, algorithm 1 returns "Secure" for each $S_i$ (but $S_1$ and $S_2$ do not satisfy TINC). To ensure a secure compression, each $S_i$ needs to be remasked with a fresh mask and stored into a register ($S_{i_{ff}} \leftarrow S_i \oplus z_j$). At most, 8 new fresh masks are needed. For each bit $y_i$, the positions of the masks $z_j$ can be changed such that the output mask of each bit would be different. The number of possible output masks is: $\binom{8}{6} = 28$.

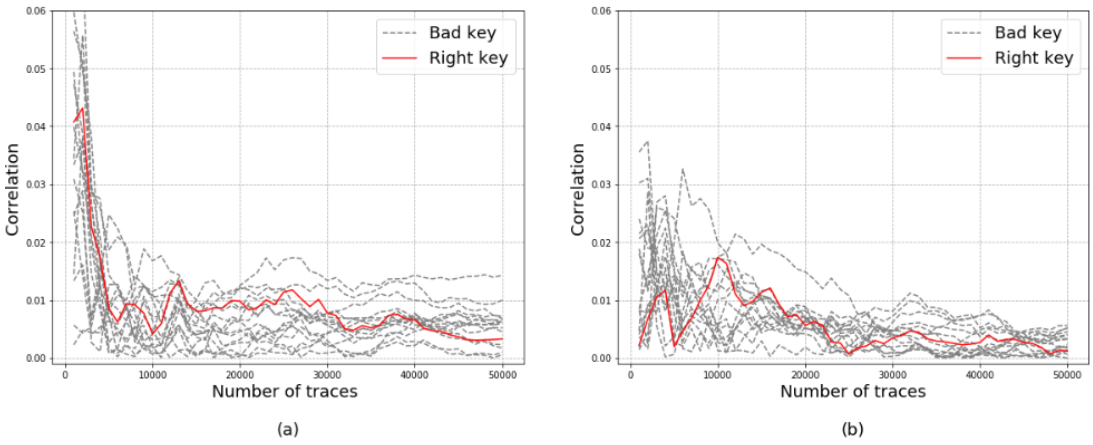



Figure 5.4: First (a) and second (b) order CPA on $S_1$ of eq. 5.9. The right key correlation is not distinguishable

In fig. 5.4 we show the result of the first and second order CPA based on simulated traces at PS level. The results of the first order confirm that the secret key is indistinguishable (see eq. 5.10). Besides, even the second order CPA is not effective for instance. The reason is that there is no configuration (a couple $(\delta_a, \delta_m)$) where a given mask $\{m_i\}$ can leak alone (in terms of transition). If the expression of the leakage involves one mask $m_i$ it also involves one mask $z_i$. Thus, the combination of the leakage cannot depend on $x_i$ because of the extra fresh mask $z_i$.

The architecture of a one-bit inversion is shown in fig. 5.5. Each $S_i$ is remasked with a new fresh mask before registration (green registers M-FF). We have proven by netlist traversal

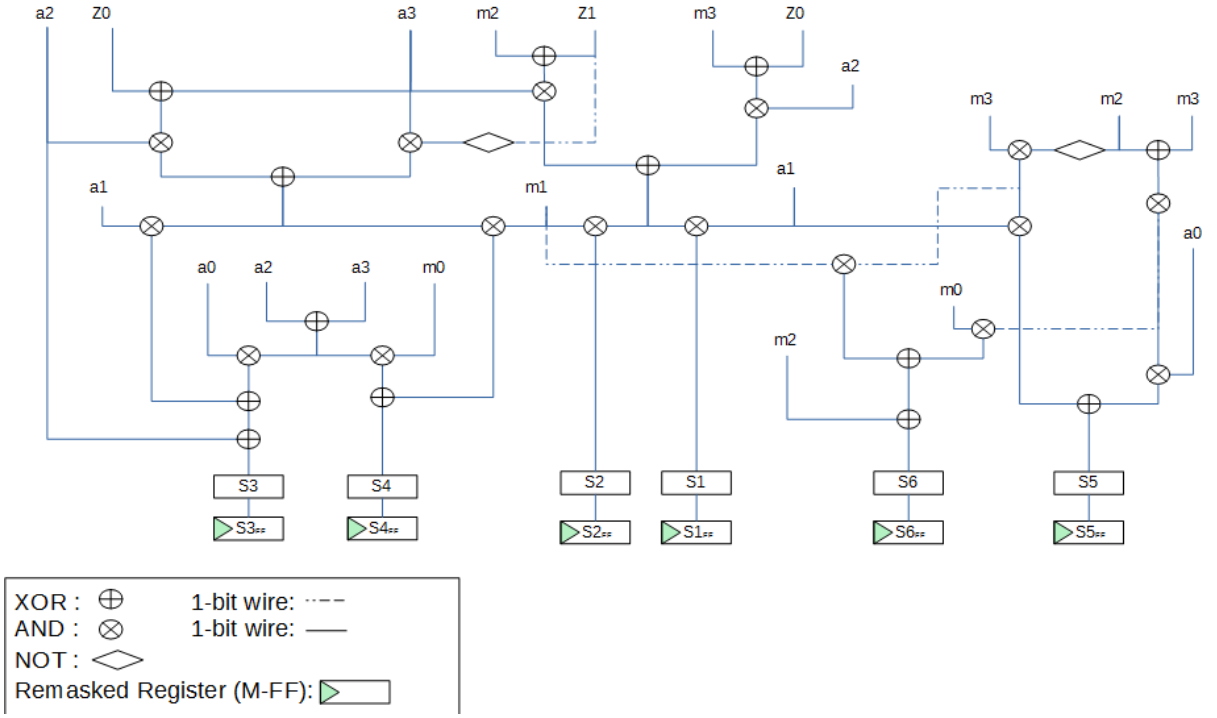



Figure 5.5: Our new design of one bit $GF_{2^4}$ inversion - Formally proven to resist against glitches.

algorithm (algorithm 1) that each signal in the design verifies corollary 1 for security in terms of value, and proposition 1 for security in terms of transitions (glitches).

We synthesized the $GF_{2^4}$ inversion, using the Cadence GSCLIB045 standard cell demonstration library, without any timing constraint. The comparison metric is the Gate Equivalent (GE) relative to the NAND2X1 cell of the library.

Table 5.3: $GF_{2^4}$ inverter - Comparing areas (GE)

| Implementation | Area (GE) | | #Cycles | First-order security | |
|---|---|---|---|---|---|
| | Logic | Sequential | | Value | Glitch |
| Canright Simple [122] | 153 | 0 | 0 | ✓ | ✗ |
| DOM [71] | 358 | 144 | 2 | ✓ | ✓ |
| TI [66] * | 618 | / | 1 | ✓ | ✓ |
| This work (eq. 5.9) | 296 | 127 | 1 | ✓ | ✓ |

*: The given logic area includes sequential logic.

The reference design is a part of the simple Canright from [122]. As shown in table 5.3, the combinational area roughly double, and 30% of more area is added for the registers. Compared with the DOM version (without pipeline), our version is 19% smaller. The TI implementation from [66] takes much more area. The number of GE which is taken from the publication is not issued from the same library, but it still bigger than DOM and our version.

To test the robustness of our design, we will perform an empirical evaluation based on digital simulation, and EM traces acquired on an FPGA target.

## 5.5 Actual exploitation of vulnerable netlists

In the following, we demonstrate attacks on netlists which have been demonstrated to contain vulnerabilities. First, in section 5.5.1, we show an attack based on virtual (simulated) traces as describe in 2.5.3. Second, in section 5.5.2 and section 5.5.2.2, we demonstrate attacks based on measured EM traces on FPGA.

### 5.5.1 SCA evaluation of Canright inverter - Digital simulation

Firstly, we have analysed the Canright RTL code based on digital simulation. We have confirmed that all intermediate results are correctly masked and independent from the secret data. Secondly, the same analysis was performed on a synthesized netlist using a SAKURA-G FPGA target (without timing), and no leakage has been reported. Once again, all combinatorial signals are independent from the secret data, and the synthesizer did not make any optimisation that may unmask the secret data. This is consistent with our constraints: we have forced the synthesizer to keep all intermediate signals and the hierarchy of each module, using the attribute *"keep"*.

Finally, when we added the timing information to the netlist, the tool has reported several leaking signals. For instance, the first level was at the compression step of the multipliers outputs, similarly to the case of the classical multiplier. The first reported leakage in the design was the signal $csa$ (see fig. 5.3) as already discussed in subsection 5.4.2. This signal is the result of a $XOR$ of the output of two non-linear functions that deal with some identical shared data.

Based on the simulation results, we were able to explicitly specify the timing information on the Standard Delay Format (SDF) file. To determine the reason for this leakage we therefore removed all the timing information excepted those of $a_0$. As expected from the study presented in section 5.4.1, the leakage was correlated to $(x_0 \oplus x_1)$. In fig. 5.6 we show the result of the CPA using the leakage model returning $(x_0 \oplus x_1)$, the red curve shows the result of the right key. We get the same result based on CCA.

The leakage model $\mathcal{L}$ is computed for any key hypothesis $K$ and the (known) output $C \in GF_2^4$ as follows:

$$\begin{aligned} X &= (c \oplus K)^{-1} \in GF_2^4 \\ \mathcal{L}(C, K) &= x_0 \oplus x_1. \end{aligned} \tag{5.10}$$

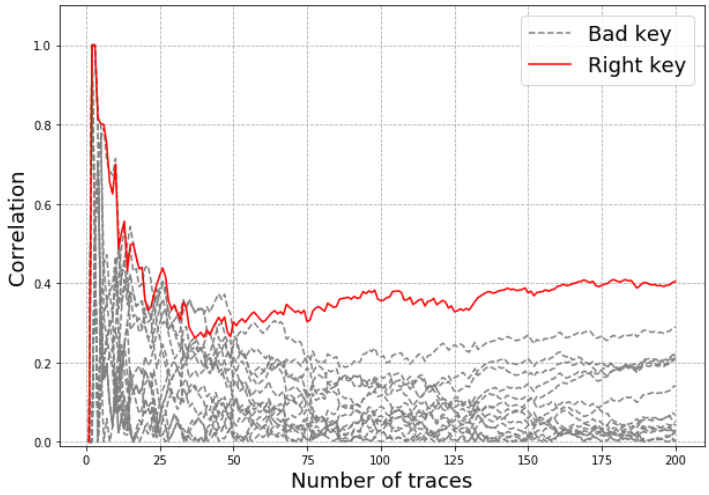


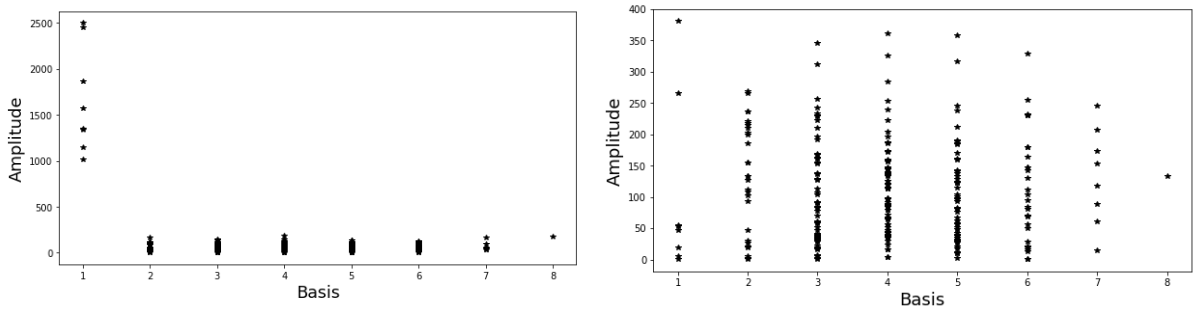

Figure 5.6: CPA on $csa_1$ activity. Only 75 traces are sufficient to recover the secret key

We recall that, by convention, the inverse of $0$ is mapped to $0$ itself. This leakage is created only if the change induced by $a_0$ would impact the gate $csa$ at the same time (within the propagation time of the $XOR$ LUT).

### 5.5.2 SCA evaluation of Canright inverter - EM Acquisition

To complete the analysis on a real target, we present different results based on EM measurement. We implemented different versions of the $GF_{16}$ inverter; with and without registers as presented in section 5.4.3.

First, we give an EM analysis only on small design involving two $GF_{16}$ inverters. Then we analyse the full AES S-box. To mark the difference between a leakage characterized by the HW of the manipulated data, we applied the WT function to the EM traces.




(a) HW leakage model case

(b) Glitches leakage case

Figure 5.7: WT applied to EM traces. $Basis = HW(W)$.

The results are displayed in fig. 5.7. In fig. 5.7a, the amplitude of the leakage is more significant when the weight of the base is equal to one ($\exists!\ w_i \neq 0$). This kind of traces can be

exploited by a HW leakage model (unmasked implementation). On the other hand, when the leakage is due to glitches, the amplitude of all the bases is almost equivalent. So the leakage is in fact a mixture of bits, like the one identified in section 5.3.

#### 5.5.2.1 Small substitution function

To perform a real evaluation with a best characterisation of the leakage, we implemented a small substitution function that we note $Sbox'$ using two $GF_{16}$ inverters, hence, we get a small block encryption, that we note $AES'$.

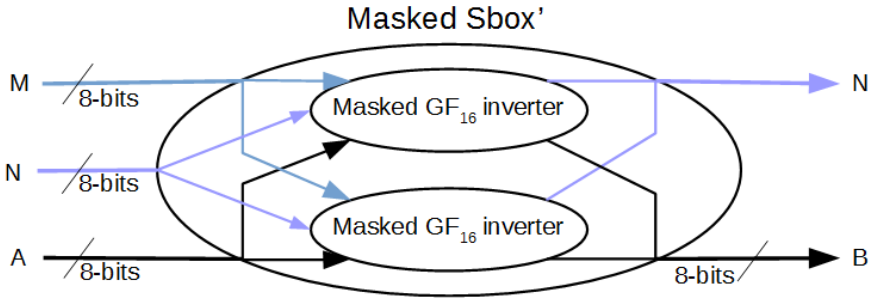



Figure 5.8: Masked design of S-box'.

The design of the masked S-box' is illustrated in fig. 5.8. The unmasked computation of this substitution function is given by:

$$Sbox'(x_7, \cdots, x_0) = ((x_7, \cdots, x_3)^{-1}, (x_3, \cdots, x_0)^{-1})$$

To have a more precise quantification of the leakage, we have implemented three masked versions of the $GF_{16}$ inverter, which are supposed to have different levels of leakages:

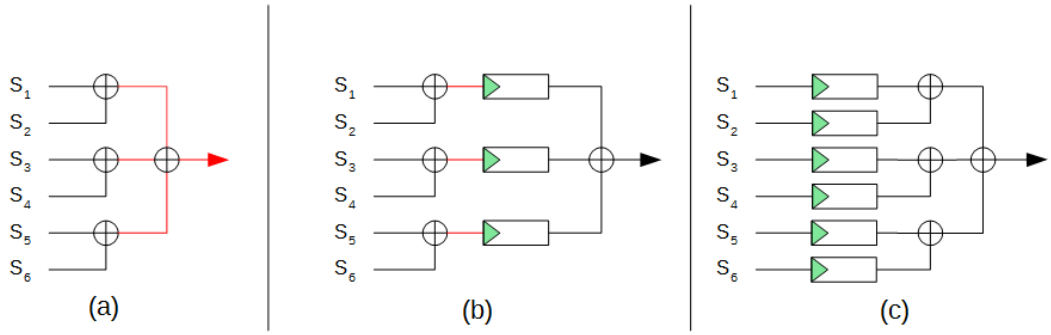
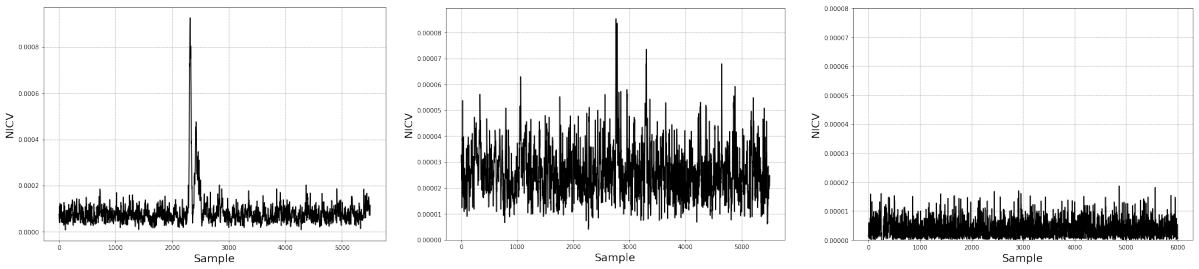



Figure 5.9: Different implementations of the $GF_{16}$ inverter. Only one-bit output is shown. The leaking signals are highlighted in red. Green FFs are remasked with fresh random.

- $AES'_0$: no register in the $GF_{16}$ inverter presented in fig. 5.9-(a) (the analysis is shown in fig. 5.10a). The design is leaking.
- $AES'_1$: only 3 registers are used instead of 6, we register only $(S_i + S_{i+1})_{i=0,2,4}$ as presented in fig. 5.9b. The leakage is still visible (fig. 5.10b).
- $AES'_2$: implementation of section 5.4.3 fig. 5.9c. Please, refer to eq. 5.9 for more understanding. No leakage detected (fig. 5.10c).




(a) 200,000 traces (b) 600,000 traces (c) 1,000,000 traces

Figure 5.10: NICV based on an unmasked intermediate state of AES'

The results of the NICV between the sensitive value (the unmasked output of the S-box') and the EM traces are presented in fig. 5.10. We can notice that $AES'_1$ is less vulnerable than $AES'_0$, and $AES'_2$ does not present any visible leakage (using 1,000,000 traces). When inserting registers with fresh masks, the leakage is progressively removed. We note also that the leakage created by the first level of the $XOR$ gates is stopped. The outputs of the registers are independent, thus no more leakage is created in the second stage of compression.

We implemented in Verilog our novel Boolean equations for the inversion in $GF_{2^4}$. The nets not to be simplified have been constrained to be kept in the netlist, using the *keep* attribute. Otherwise, we let the synthesizer (Cadence Encounter) optimize the netlist by merging common sub-expressions. The resulting netlist is displayed in fig. 5.11. We have verified formally that each node fulfils the requirements of corollary 1 and proposition 1. The combinational gates are as usual, and rectangle symbols represent the 24 DFFs.

#### 5.5.2.2 AES S-Box

For the full AES implementation, we have added registers at the output of each $GF_{16}$ multiplier in the S-box design (see figure fig. 5.13). The implementation without registers is shown to be very leaky. So, we have used only the two last versions of the inverter for the full AES:

- $AES_1$: only 3 registers are used instead of 6, we register only $S_i + S_{i+1}$
- $AES_2$: use 6 registers as for AES'. The full masked S-box is illustrated in fig. 5.13.

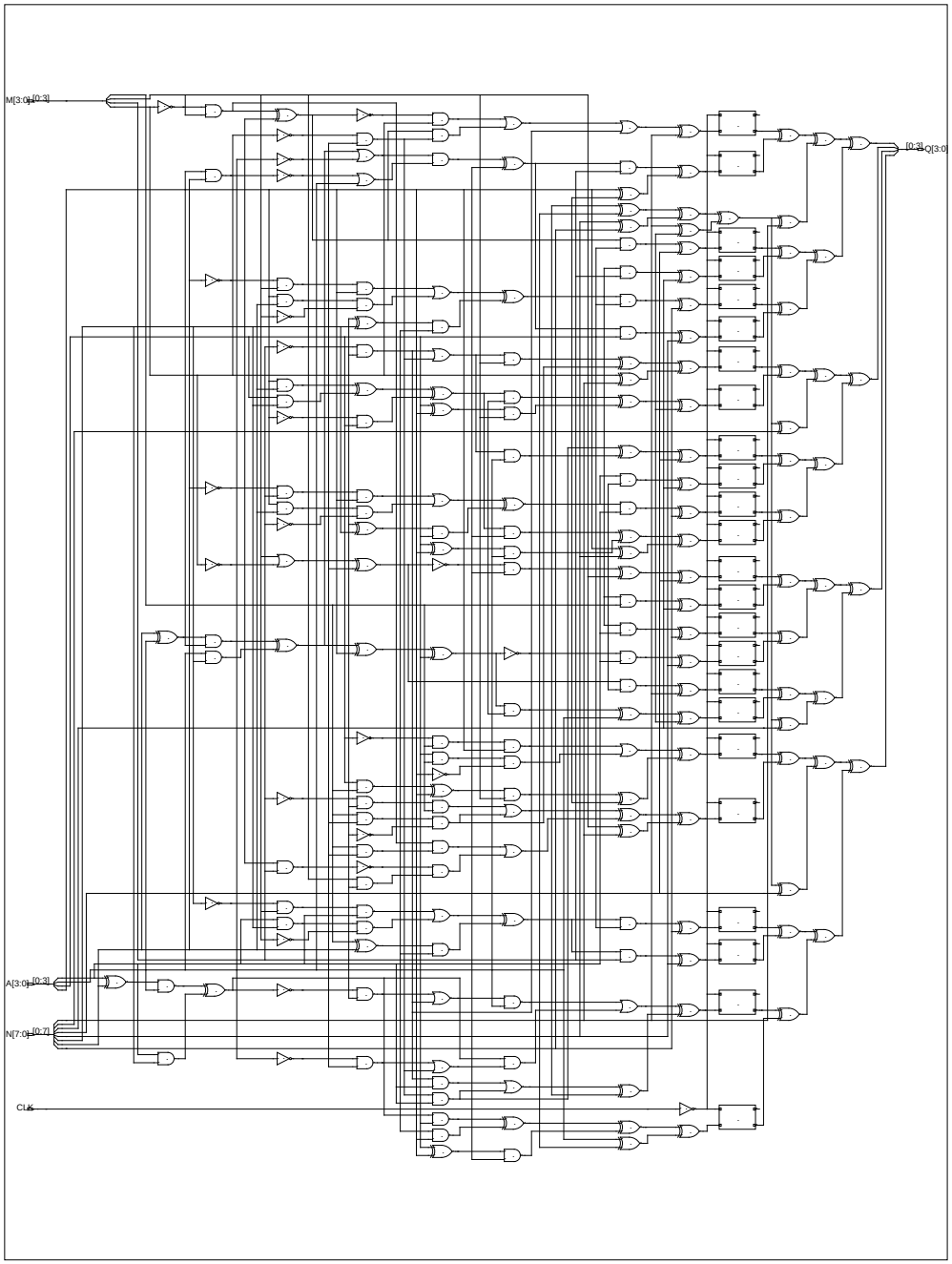
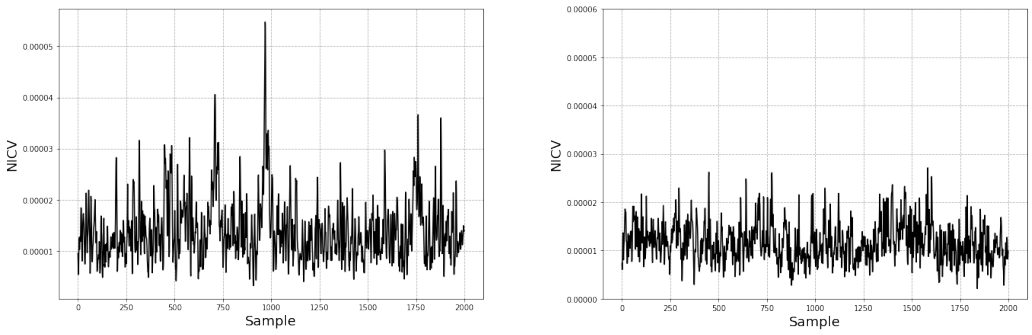



Figure 5.11: Masked $GF_{2^4}$ inverter synthesis result. The synthesizer did not optimize the design. All signals are correctly kept.




(a) $AES_1$ using 1,000,000 traces (b) $AES_2$: using 1,200,000 traces

Figure 5.12: NICV based on an unmasked intermediate state of AES

The results of the NICV for both implementations are presented in fig. 5.12. The result of the first implementation $AES'_1$ is given in fig. 5.10. The leakage is comparable to the one identified in fig. 5.10b, and 10 times smaller than $AES'_1$, because of the extra activity (noise) of the rest of the AES S-box. The last implementation ($AES_2$) does not present any leakage using 1,200,000 traces, which is consistent with the result obtained on the $AES'_2$ implementation.

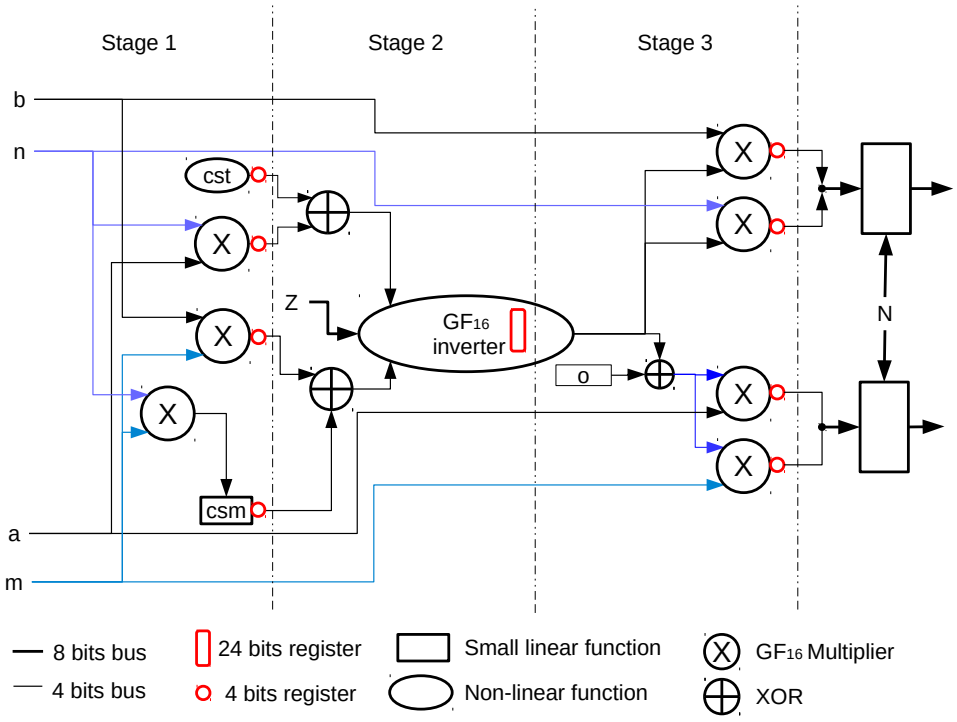



Figure 5.13: AES S-box scheme using our secure $GF_{2^4}$ inverter

The whole design of our S-box is presented in fig. 5.13. The names of the signals are the same as the original implementation given in [122]. The input signals are as follows:

- $A = (a_3, a_2, a_1, a_0, b_3, b_2, b_1, b_0)$: 8-bits masked input

- $M = (m_3, m_2, m_1, m_0, n_3, n_2, n_1, n_0)$: 8-bits input mask
- $N$: 8-bits output mask
- $Z = (N, z_1, \cdots, z_{18})$: 26-bits fresh random including $N$

Also, we can notice that the stage $1$ and $3$ are identical to stage $1$ and stage $4$ of the DOM S-box. As the different output bits of the inverter $GF_{16}$ are $xored$ together at the $GF_{16}$ multipliers level, we masked the 24 FFs with different masks to avoid any transition resulting from a delayed register output with an identical mask. Indeed, we need $18$ bits of fresh random bits ($Z = (N, z_1, \cdots, z_{18})$).

## 5.6 Conclusion

In this chapter we have evaluated the security of hardware masked implementations against SCA vulnerabilities in presence of glitches. We have detailed the form of the leakage and exposed the different ways to prevent information leakage.

Namely, we presented an algorithm to check exactly for leakage in terms of values and transitions in masked netlists. It is subsequently possible to design more compact and optimized functions. Indeed, our algorithm allows to check the security of netlists implementing logic using gadgets which are less constrained and more compact than the conservative methodology required by TI or DOM.

We have given more understanding about the leakage on masked non-linear gates based on an in-depth analysis in terms of transition based power consumption. Thus, we have identified the critical parts on the non-linear gates that should be treated carefully. In addition to a formal security proof, our results are argued based on empirical verification on simulated synthesised netlist and EM traces, as it was expected from the formal analysis in presence of propagation time.

# Part II

# Active attack and countermeasures

# Chapter 6

# Fault Injection Analyses Assisted by Simulation

Contents


6.1 Introduction . . . . . 89
6.2 Fault model . . . . . 90
6.3 Fault detection on protected implementation . . . . . 92
6.4 Conclusion . . . . . 98


## 6.1 Introduction

To perform a Fault Injection Analysis (FIA), an adversary needs to induce errors into the target device. Using some tampering means, which can be accomplished in several ways, is extensively discussed in literature [123]. In general, tampering means or fault injection techniques are classified into two broad categories, i.e., *global* and *local*. Global fault injections [124] are, often, low-cost techniques which create disturbances on global parameters like voltage, clock or temperature. The resultant faults are more or less random in nature, and the adversary might need several injections to find required faults. On the other hand, local techniques (e.g., clock glitch, optical/electromagnetic injections, body bias injection [125]) are more accurate in terms of fault location and model. However, this precision comes at the expense of costly and bespoke[1] equipment. The kind of injected fault can be defined as fault model which has two important parameters, namely *location* and *impact*. Location means the spatial and temporal location of fault injection during the execution of target algorithm. Depending on the type and precision of the technique, the impact can be at bit level, set of bits (variable) or completely random. Coming to the impact of fault, it is the effect on the target data. Commonly known

[1] In Common Criteria parlance.

fault injection impacts on target data can cause stuck-at, bit-flip, random-byte, or uniformly distributed random value.

## 6.2 Fault model

### 6.2.1 Clock-glitch injection

The principle of the Clock-Glitch injection consists in precisely modifying the period of one or more clock cycles of the target design during the execution. When the modified clock period is much shorter than what is expected in the normal clock, it shall create setup violation faults [126]. In a case of cryptographic implementations, these faults can be exploited to retrieve the secret key.

Since the modification of the clock frequency at RTL level is meaningless, we can perform clock glitch injections only with back-annotated gate-level descriptions (i.e., at post-synthesis level or at place and route level). To this end, we synthesized an AES core using an ASIC 65nm CMOS technology and used nominal PVT (Process, Voltage, Temperature) conditions for timing information extractions. After that, we have configured the fault model to perform a clock glitch on a specific cycle during PS simulations taking into account the gate and wire delays (e.g., SDF file). The configuration consists in defining some parameters needed to set the stimuli for simulations and the clock glitch parameters, in particular, the cycle target and the glitch duration. In our case, the main configuration was as follows:

- Target cycle: last round of the AES execution;
- Glitches duration: from 4 ns to 7 ns with steps of 100 ps;
- Number of simulations: 310.

Figure 6.1 shows a cartographic view of the effects of clock glitches in terms of erroneous bits observed in the final output. Based on such information, the evaluator can easily identify the minimal glitch duration that would lead to a final output error for a given cycle.

Simulation results can be used to apply a set of DFA, which exploit differences between correct and faulty outputs to recover the key, such those presented in section 2.7. One example is the NUEVA metric [99] which measures the uniformity of error values injected before the last S-box operation in order to find the key. Another example is the AES-128 DFA using Giraud metric [127]: This fault analysis requires single-bit faults at the input of the last S-box operation. As shown in Fig. 6.2, the key is recovered entirely with only 126 simulations, using DFA of Giraud. A few more simulations are required to perform the full analysis with the NUEVA technique.

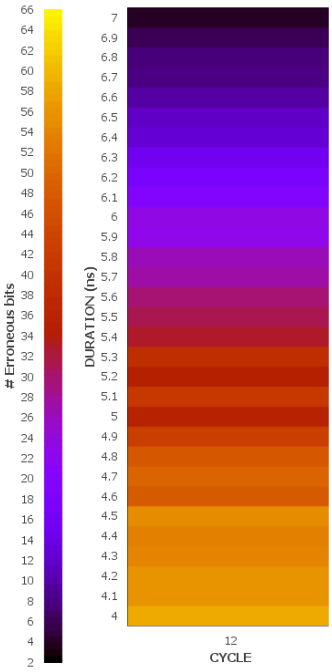



Figure 6.1: Erroneous bits according to the glitch duration.

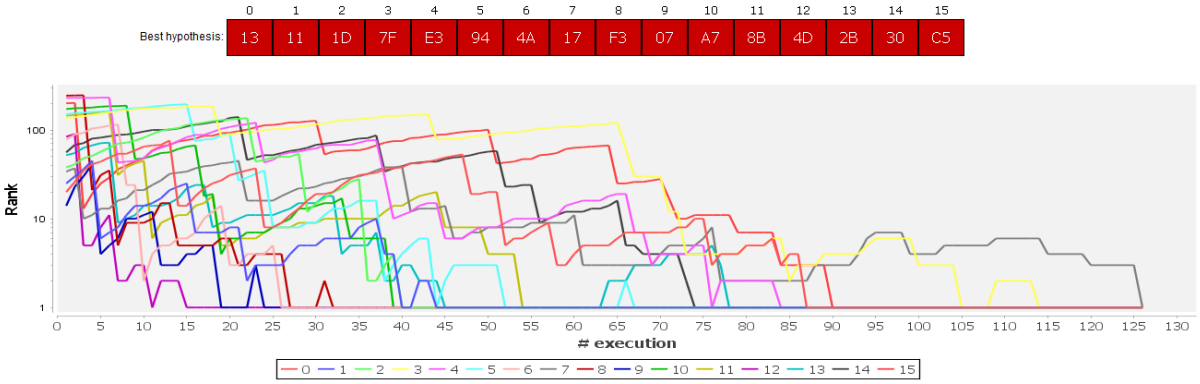



Figure 6.2: Analysis of clock-glitch injection results using DFA AES-128 Giraud metric.

### 6.2.2 Laser injection

Laser fault injection falls into optical fault injection methods which consist in exposing the device to an intense light for a brief period of time. The injection can be performed either through the front-side or the backside of the target chip [128, 129, 130]. Laser attacks can be used to inject faults characterized by high locality and timing accuracy. The laser injections can be modelled not only at gate-level but also at functional level (i.e., RTL) by configuring parameters such as the fault type (e.g., permanent/transient), the fault model (e.g., bit-flip, bit-set, bit-reset, stuck-at-0/1), the fault location (e.g., wires, registers) and the fault time.

In this experiment, we have performed our analysis at RTL level with the following configu-

ration:

- Fault time: last round cycle of the AES execution
- Fault location: inputs of the S-box module
- Fault model: bit-flip model
- Number of simulations: 100

Figure 6.3 illustrates the results of the analyses completed using DFA metrics already presented in the previous section. We can see that all key bytes are broken using only few simulations, in this case 10 with the DFA based on Giraud metric.

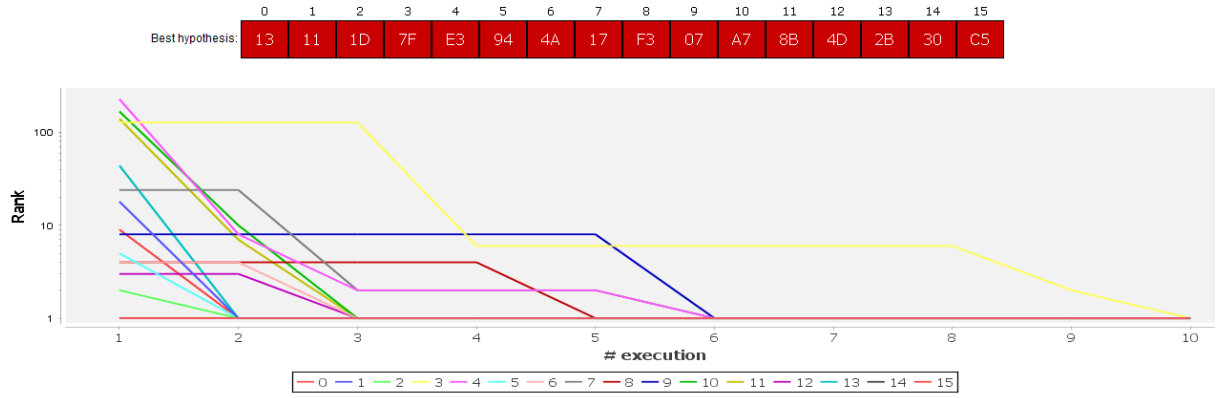



Figure 6.3: Analysis of laser injection results using DFA AES-128 Giraud metric.

## 6.3 Fault detection on protected implementation

The attacks based on malicious injection of faults can degrade seriously the security of a cryptosystem. Faults injected into the cryptographic modules during the encryption (or decryption) operation will very likely result in a number of errors in the encrypted/decrypted data. Such faults must be detected before their spread to avoid the transmission and use of incorrect data. Fault detection techniques represent, therefore, a possible countermeasure against fault injection attacks and a desirable property for preventing malicious attacks, aimed at extracting sensitive information from the device, like the secret key.

### 6.3.1 AES-EDC implementation

For the AES block cipher, two main approaches have been proposed for achieving fault detection. The first one is based on temporal or spatial redundancy; in temporal redundancy, the

same hardware is used to repeat the same process twice using the same input data. This technique uses minimum hardware overhead. Yet, it entails time overhead. In spatial redundancy, two copies of the hardware are used concurrently to perform the same computation on the same data. After each computation, the results are compared and any difference is reported as a fault. The advantage of this technique is that it can detect all kinds of faults. However, it requires an important hardware overhead.

The second approach is concurrent error detection using Error Detecting Code (EDC). It employs circuit-level coding techniques, e.g, parity schemes, modular redundancy, etc., to produce and verify results after each computation.

From a security point of view, designers have to verify the effectiveness of a given implemented countermeasure and be sure that it prevents against fault analysis. Remark that all countermeasures detect faults only to some extent (*e.g.*, up to a certain order, that is to say, up to a certain bit-wise multiplicity).

For this purpose, we present our results based on the countermeasure presented by Bertoni et al. [131] which targets the datapath of the AES encryption module. This countermeasure uses a $4 \times 4$ parity matrix. Each bit corresponds to one byte of the state, and at each round the matrix is predicted and then can be compared with the computed one from the state. This countermeasure can detect all odd errors and some even errors. The hardware overhead is less than many other countermeasures (e.g., [132]) where a computation redundancy is required (2 times overhead).

We designed an AES-128 encryption module implementing this countermeasure for the datapath (see fig. 6.4). The control unit is also protected by computing the parity of the rounds counter.

In this implementation we can distinguish two fault detection blocks:

- The first (1) is used to check the integrity of the register. If a fault is injected into the register, then the comparison between its current parity and the predicted one (16-bit register) returns a non-zero result (assuming the number of flipped bits within the same byte is odd).
- The second (2) is used to check the integrity of the intermediate calculations.

Different scenario of injection are illustrated in figure fig. 6.5, we restricted the detection to the output of the S-box, but it is equivalent to check the parity at the *MixColumn* output as in the whole design fig. 6.4.

The functions of fig. 6.4 are:

- SBOXWITHPARITY: The 128-bit output corresponds to the standard output of the AES S-box function. The 16-bits output corresponds to the parity of the 128-bits output. This

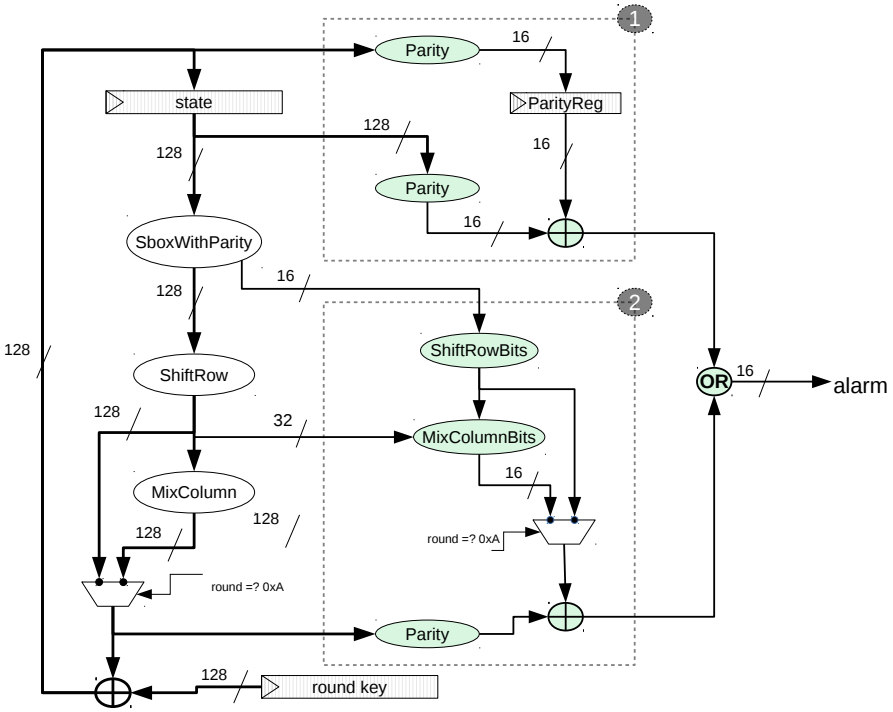



Figure 6.4: Datapath of the AES parity check implementation against fault injection.

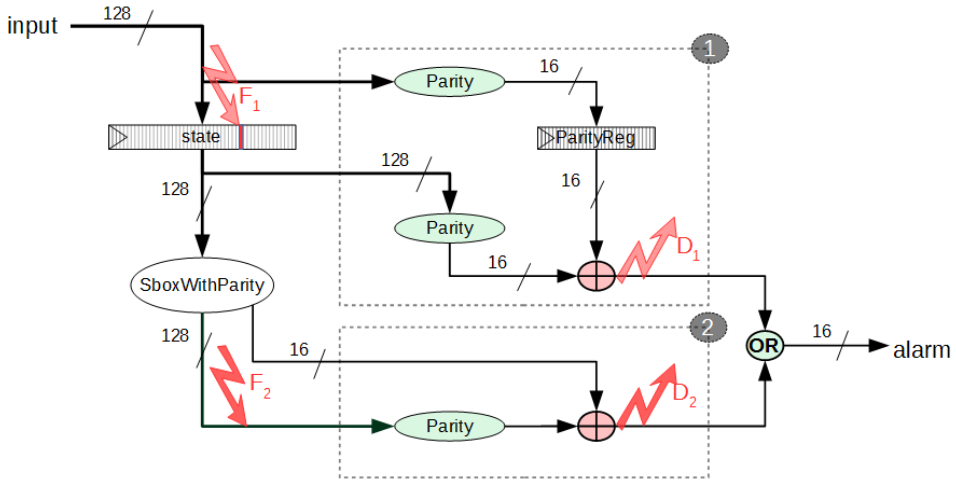



Figure 6.5: Datapath of the AES S-box parity check against fault injection. When flipping one bit of the state register ($F_1$) it will be detected in the first block ($D_1$). When flipping the S-box output ($F_2$) it will be detected in the second block ($D_2$).

step is calculated using a look-up table containing $256 \times 9$ bits; 8-bits for the standard output, and one parity bit;

- SHIFTROWBITS: Applies a rotation to the parity bits to align them correctly according to the SHIFTROW output;

- MIXCOLUMNSBITS: Predicts the parity of the output of MIXCOLUMNS. In addition to the parity matrix, this step requires some bit of the output of SHIFTROW to predict the parity of MIXCOLUMNS;
- PARITY: compute the parity of each byte of the 128-bits state.

### 6.3.2 Design evaluation

We have performed several simulation-based fault injection campaigns at RTL level in order to evaluate the fault coverage of the proposed parity-based EDC scheme. One hundred thousand injections are performed, where plaintexts and keys are selected randomly. The fault model is a single bit-flip at the last round of the encryption operation. The obtained results show that the detection rate is equal to 100% as shown in [131]. Then, we launched the logic synthesis on a Virtex-V Xilinx FPGA as technology target in order to perform the same fault injection campaigns but at Post-synthesis level (PS) (i.e., the post-map netlist is used during simulations). As expected, the detection rate is equal to 100%.

#### 6.3.2.1 Analysis with synthesis optimization

We re-synthesized the same RTL code but with different logic synthesis options to optimize the logic and to improve timing and design performances. As a matter of fact, the Xilinx Synthesis Technology (XST) synthesis tool allows designers to configure several options and properties that are taken into account during the synthesis process. These options target possible optimizations for area, speed or power consumption.

An extract from the Xilinx synthesis settings dialog box is shown in Figure 6.6. In our case, we activate some options to optimize the design such as the $-logic\_opt$ option which optimizes timing-critical connections through restructuring and resynthesizing, followed by incremental placement and incremental timing analysis. Previous injection campaigns are performed based on the obtained netlist. However, results are not the same because the detection rate decreases from 100% to 18.75%. More precisely, only faults injected on the AES control unit are detected. All faults into the datapath are no longer detected due the synthesis tool optimizations. The resulting architecture of the synthesis phase is shown in fig. 6.7.

The countermeasure logic on the datapath was completely removed after the logical synthesis to optimize the design for area by reducing the total amount of logic used for design implementation. With obviously less gates, an equivalent functionality is obtained, albeit with a lesser security. Indeed, the S-box is left unprotected, simply because the synthesizer has been smart enough to eliminate some combinational schemes considered to be equivalent. Functionally speaking, there is no alteration. However, from a security standpoint, the complete S-box transformation is left unprotected.

| Switch Name | Property Name | |
|---|---|---|
| -ol | Placer Effort Level | High |
| -xe | Placer Extra Effort | None |
| -t | Starting Placer Cost Table (1-100) | 1 |
| -logic_opt | Combinatorial Logic Optimization | ✖ |
| -register_duplication | Register Duplication | Off |
| -global_opt | Global Optimization | Power |
| -equivalent_register_removal | Equivalent Register Removal | ✖ |
| -x | Ignore User Timing Constraints | ☐ |
| -ntd | Timing Mode | Performance Evaluation |
| -u | Trim Unconnected Signals | ✖ |
| -ignore_keep_hierarchy | Allow Logic Optimization Across Hierarchy | ☐ |
| -cm | Optimization Strategy (Cover Mode) | Area |
| -detail | Generate Detailed MAP Report | ☐ |
| -ir | Use RLOC Constraints | Yes |
| -pr | Pack I/O Registers/Latches into IOBs | Off |
| -c | Maximum Compression | ☐ |
| -lc | LUT Combining | Auto |
| -bp | Map Slice Logic into Unused Block RAMs | ☐ |
| -power | Power Reduction | ☒ |
| -activityfile | Power Activity File | |
| -mt | Enable Multi-Threading | Off |
| | Other Map Command Line Options | |

Figure 6.6: Extract from the XST synthesis options for Xilinx FPGAs.

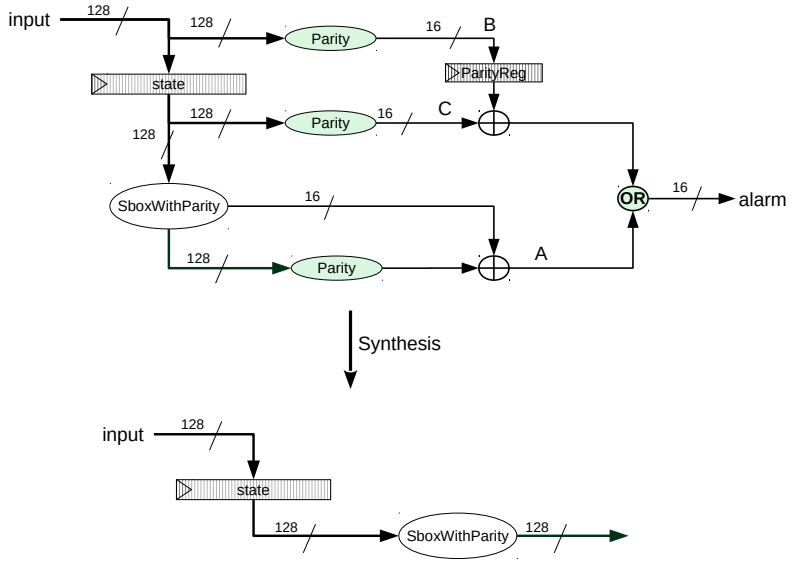



Figure 6.7: Total simplification of fault detection logic upon synthesis.

For the optimization prevention of signal $B$ in fig. 6.7, we use the `DONT_TOUCH` attribute. This attribute prevents optimizations where signals are either optimized or absorbed into logic blocks. It instructs the synthesis tool to keep the so tagged signal, and that signal is placed

in the netlist. Logic synthesis and fault injections are remade with the same options used during the previous experimentation. Results indicate that the detection rate increases from 18.75% to 56.43%. Indeed, the synthesis tool has partially simplified the fault detection logic as shown in fig. 6.8 by eliminating the combinational block producing $C$ signal. Consequently, only faults injected on the state register are detected, which opens a large door for successful fault injection attacks within the combinational logic.

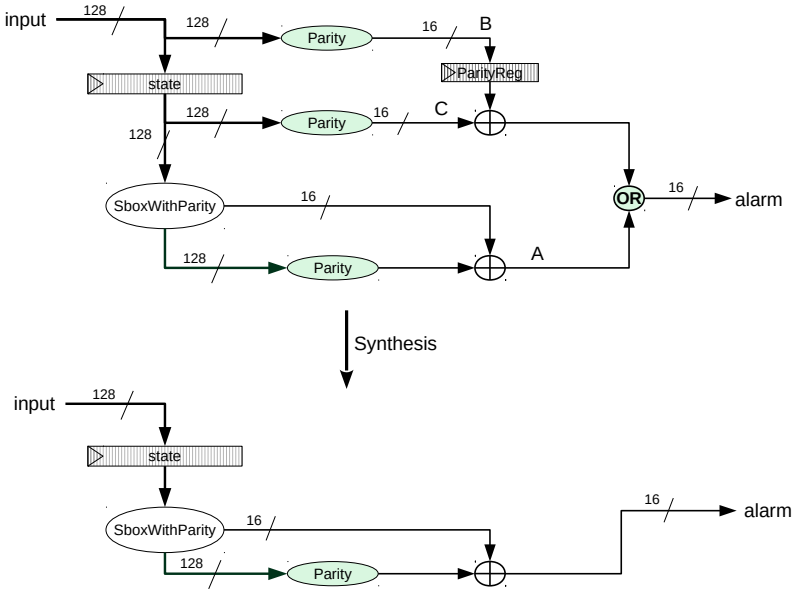



Figure 6.8: Partial simplification of fault detection logic upon synthesis.

Table 6.1: Fault detection rate for RTL and post-synthesis levels.

| Level | RTL | PS (default options) | PS -logic_opt = true -xor_collapsing = true | PS -logic_opt = true DONT_TOUCH attribute |
|---|---|---|---|---|
| Detection rate | $100\%$ | $100\%$ | $18.75\%$ | $56.43\%$ |

Table 6.1 summarizes the fault detection rate according to the analysed level. We can conclude that the protection can be removed altogether during logical synthesis, thereby causing a security regression. This kind of mis-integration may happen in real case, where designers do not check the security evolution of their design at each stage of synthesis. Therefore, robustness of hardware cryptographic modules against fault injection attacks should be evaluated at each abstraction level in the design conception flow.

Another reason for designer's attention to be deflected from security is the requirements for testability. Clearly, in fig. 6.8(a), the alarm signal is not testable. Indeed, it is consistently

equal to '0'. Therefore, in a view to achieve DFT (Design For Test) requirements, some test logic to address independently the registers driving signals A, B & C, shall be added. But in the meantime, the designer might shift its focus so conscientiously that he might forget about the need for setting `DONT_TOUCH` attributes. Hence the need for an automated verification as an independent third-party verification tool.

## 6.4 Conclusion

FIA are serious threats to cryptographic algorithms [123]. Countermeasures have been developed against such attacks. Still, it is non-obvious how to implement such protections at source-code level. There are many options to configure the synthesis tools. Hence exploring their combinatorics is exponential. In practice, users select a few options. Some options can lead to total or partial simplification of the countermeasure. Using a simulation-based methodology, we manage to detect such alterations, and we quantify the amount of degradation. In addition, we precisely pinpoint the residual leakage samples.

We also emphasized the need to verify the functioning of the countermeasures at each stage of the design. Indeed, some parts can be simplified and thus, compromise the implemented protections and the security of the device.

# Chapter 7

# Evaluation Against Focused Ions Beam for Probing Attack

Contents


7.1 Introduction . . . . . 99
7.2 Probing model . . . . . 100
7.3 FIB for probing attack . . . . . 100
7.4 Methodology of FIB for probing . . . . . 101
7.5 Study-case on AES . . . . . 108
7.6 Conclusion . . . . . 113


## 7.1 Introduction

Probing attack is considered to be one of the most powerful attack used to break the security and extract confidential information from an embedded system. This attack requires different bespoke equipment and expertise. However, there is no methodology to evaluate theoretically the security level of a design or circuit against this threat. It can be only realised by a real and certified evaluation laboratory. For the design house, this evaluation can be expensive in term of time and resources.

In this chapter, we introduce an innovative methodology that can be applied to evaluate the probing attack on any design at simulation level. Our method helps to extract the sensitive signals of a design, emulate different Focused Ions Beam technologies used for probing attacks, and evaluate the accessibility level of each signal. It can be used to evaluate precisely any probing attack on the target design at simulation level, hence reducing the cost and time to market of the design. This methodology can be applied for both ASIC and FPGA technology. A use-case on an AES-128 shows the efficiency of our methodology. It also helps to evaluate the efficiency of the active shield used as a countermeasure against probing attack.

**Outlines.** We give an end-to-end methodology to evaluate a circuit against front-side FIB probing attacks. Based on a full pre-silicon model of the circuit, we give an automated evaluation of sensitive signal identification, location and complexity access given a FIB configuration. Our main contributions are:

- Automatic identification of sensitive signals;
- Improved method for exposed area detection [133];
- An adapted metric for evaluating the security in term of exposed area.

The sensitive signal identification is based on NICV metric [39], that we apply to each signal individually, using the critical parameters of the implementation. Only a few knowledge of the target IP is required, which allows testing third-party IPs, since the layout file description (Library Exchange Format (LEF) and Design Exchange Format (DEF) files) are provided.

In section 7.4, we describe the different step of our methodology about sensitive signal identification, location and evaluation against probing attacks. In section 7.5, we give some results on protected implementation using a shield, and we discuss how the security can be improved by inserting new (virtual) shield.

## 7.2 Probing model

As already explained in section 2.5.1, in the probing model scenario, the attacker is allowed to probe $d$ signals [73]. It is said to be secure at order $d$ if no information about the secret can be learned up to $d$ probes. If we consider a powerful attacker who can record a given signal of the circuit, the number of needed measurements to recover the key depends on the function that computes this value [113].

For example, if we probe the value of the *AddRoundKey* output, we can recover only one bit of the secret key. The attacker needs to probe each bit to recover the whole key (which is very complex and time consuming). The best way to minimise the number of measurements is to probe a non-linear function [113]. In the case of AES or DES, we probe the S-box output (or the input if we target the last round) [112].

## 7.3 FIB for probing attack

To achieve a real probing attack, a FIB workstation is required. The attacker need to follow three main steps, as already described in section 2.8.2. The complexity of the probing attack depends on many parameters. Mainly, the step of reverse engineering is the most complex one. The attacker should identify each block and the vulnerable signals of the implementation

[134]. This process is highly dependent on the performance of the workstation. We refer the reader to section 2.8.2 for more details.

In [133, 3], the authors described a methodology allowing to analyse a hardware implementation protected by an active shield against probing attack. They showed on a protected implementation with an active shield, the optimal ratio necessary to bypass the shield, or conversely, deduce the ratio for which the shield remains effective. Their methodology aims to find the exposed areas, by excluding the zones where other wires cross the image of the target wire on the layer above.

Our approach is similar but complementary in the sense that we are looking for all exposed areas according to a maximum authorized angle (by the FIB or by the attacker), which thus allows us to take into account exposed area with an angle. With this method, more exposed area can be identified.

## 7.4 Methodology of FIB for probing

As described in the previous section, FIB probing is an advanced, complex and extremely expensive attack. Therefore, there are just few entities that can realize a FIB testing on their circuits. For this reason, we propose a new methodology to simulate the FIB attack at an early stage of the design life cycle. With this methodology, the designer can simulate and correct all vulnerabilities that can be exploited by the attacker using a FIB. The new methodology is composed of the following steps that we detail in the sequel:

1. Sensitive signals identification
2. Sensitive signals location
3. Exposed signals

The global workflow of our approach is presented in fig. 7.1. In term of FIB attack, we can address three main types; by-pass attack, re-routing attack and disable shield attack.

When an implementation is protected by a shield, the easiest way for an attacker is to avoid cutting its wires, which is the first attack (by-pass attack). The last two attacks require more effort on the attacker side. They require more investigation for the reverse engineering step, and the routing of certain wires. This increase the attack time and its complexity. In the following, we address only the by-pass attack, which do not require circuit edition.

### 7.4.1 Sensitive signals identification

The FIB allows probing and monitoring the internal signals of the circuit during its operation. With the retrieved data, the attacker can recover the sensitive information hidden inside the

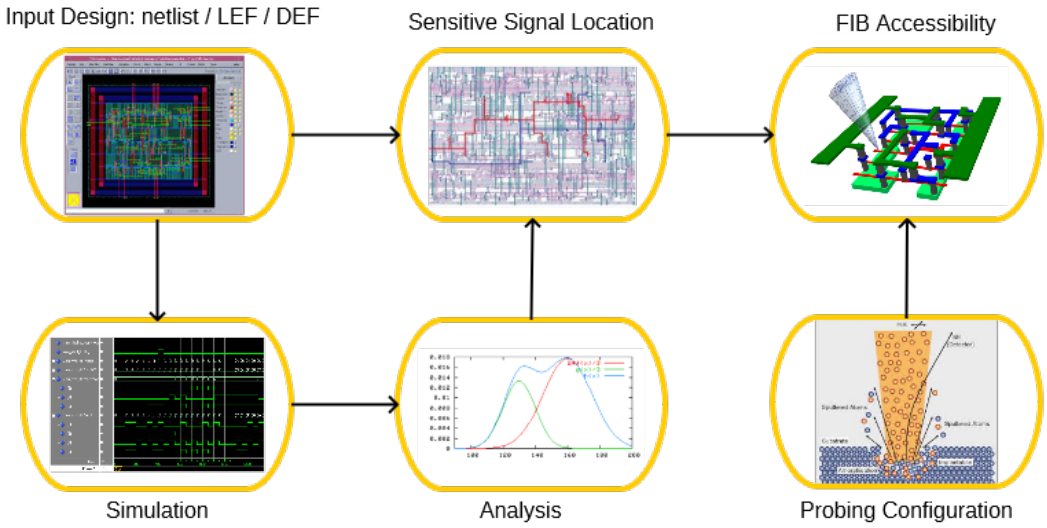



Figure 7.1: Global workflow for probing evaluation threats

circuit. The question is which signal the attacker needs to probe. In a complex circuit, with thousands of internal signals, he cannot probe them all. For this purpose, the first step of our methodology consists in creating a method to select a group of sensitive signals that could be interesting for a FIB attack. The workflow of our method is the following:

- Tag the critical parameters
- Create the testbench
- Launch the logic simulation
- Create the simulated traces
- Analysis

The first step of our method consists in tagging the critical parameters. In this step, the designer needs to define all critical parameters that he wants to protect against the FIB attack. For example, they could be the value of the secret key, plaintext or masks of cryptographic IPs. The second step takes the critical parameters to create an appropriate testbench. A test process is added to randomize these parameters. It is used to evaluate the propagation of these values into the design.
The third step consists in launching the simulation of the new testbench using a digital simulator. During the simulation, all internal signals states are stored and used for the evaluation.
In the fourth step, we use the simulation results to generate the activities traces of each signal. At the end, we use the NICV as a metric for the evaluation. This metric allows detecting the dependency of each simulated signal with the sensitive parameters which are defined above by the designer. This metric is applied for each internal signal and each sensitive parameter.

At the end, we obtain the NICV coefficient of each signal for each time sample. Then, we can apply a threshold to select the signals where the NICV is greater than this selected threshold. It means that these signals are correlated with the sensitive values that the designer wants to protect. Hence, by probing these signals, an attacker can retrieve these sensitive values. At the end, a list of sensitive signals for each sensitive value is obtained.

### 7.4.2 Sensitive signal location

Once the sensitive signals are identified, we need to know if these signals are accessible. First, we need to identify the physical location of these signals in the layout. It is done using a layout parser. This parser is able to analyse all kind of layout (ASIC or FPGA design) and extract the location of each physical segment of the signals. It will allow identifying how many segment a specific signal (or net) has, on which metal they are located and their corresponding coordinates. The procedure of this parser is the following:

1. Take the layout file as input;
2. Find the information related to the technology (number of metal layers, wires width, Vias etc.);
3. Parse the name of all wires used by the devices (including the power wires);
4. For each wire, retrieve the following information:
    - The different segments;
    - The metal layer related to each segment;
    - Different Vias of the layer;
    - The metal layers related to each Via.

At the end of the parsing step, we get the whole information of each wire. All this information will be stored in a database. Then, a customized program is used to select the desired signal and show all this information. It gives the information of both sensitive and non-sensitive wires (signals). The information of non-sensitive wires is also important. It will help us to determine the real sensitive areas for probing attack. More details about the sensitive areas will be presented in the next section.

### 7.4.3 FIB probing model

A FIB is composed of different components that allow scanning and milling specimens. An electronic microscopy is used to scan the surface of the sample, and an ion beam for milling and lamellae preparation. In the case of milling, a flow of ions are emitted with specific current **I** $(5\ nA; 30\ nA)$), accelerated at a specific voltage U $(5\ kV; 30\ kV)$, and focused into a point of the sample. The ions hit the surface of the target and weaken the focused zone and tear atoms

from the sample. The depth and the diameter of the left hole depend on the Dwell time (fixed time at single point), the beam current and the voltage. Another factor which depends on the sputtered yield is the incidence angle to the surface. Experiments show that the maximum yield is reached when the angle is between $65^{\circ}$ and $85^{\circ}$. The spot size of the beam is obviously the most important parameters which define the FIB resolution. The best knows resolution is about $5\ nm$ [104].

The purpose of probing attack is to be able to access to some sensitive signals of the circuits. To access these signals, we need to identify an appropriate area, that optimizes the milling step. This can be defined as the dimension of the cone that we must make to achieve that, and decide if a such cone is feasible with a given FIB.

### 7.4.4 FIB access methodology

In the circuit layout, we have different layers that contains the targeted signal. For a given signal at position $X = (x, y, z)$ (or a list of positions of wires), we try to access this signal without damaging the circuit (or with minimal damage). We describe our method applied to a wire, which can be seen as a list of positions at different layers. The principal idea of this method is a bottom-up process, which is based on two principle steps:

- Projection: The wire will be projected recursively to the layers above;
- Delimitation: This step consists in eliminating the region that is crossed with other wires, or select the one that has the less number of wires (minimal damage).

We start from the wire position and give the area from where it can be accessed. Note that in this method, we assume that all wires have either $0^{\circ}$ or $90^{\circ}$ with respect to the $X$ axis.

In algorithm 2, we give the projection and delimitation steps that give us the list of all areas allowing to access any sensitive wire.

#### 7.4.4.1 Projection

A wire can be seen as a list of positions in a given layer. Here, we describe the whole process for one segment of the wire (for the whole wire, we apply the same method for each segment). The normal projection of the wire gives its image at the top layer, and by varying the projection angle $\theta$ from $[0, \theta_{max}]$ along $x$ and $y$ axes from the normal angle, we get a rectangular image which represents the zone from where the targeted wire can be reached from the layer above. If the segment is determined by two positions $(x_0, y_0)$ and $(x_0, y_1)$ (here we suppose that is vertical), then the boundaries of the rectangle can be computed as follows:

$$r = z \times \tan(\theta_{max})$$
$$R = \{(x_0 - r, y_0 - r), (x_0 - r, y_0 + r), (x_0 + r, y_1 + r), (x_0 + r, y_1 + r)\}$$

**Algorithm 2:** Projection and delimitation process

**Input:** Design: (LEF, DEF files) , Signal target: S
**Output:** Accessibility paths

```
1  Segments ← shape(S)
2  for segment ∈ Segments do                    // For each segment in Segments
3      current_layer ← get_layer_index(segment)
4      layer_above ← current_layer + 1
5      height ← Design.get_distance_between_layers(current_layer, layer_above)
6      rectangle ← first_projection(segment, height)                  // Projection
7      wires_at_layer_above ← Design.get_wires_at_layer(layer_above)
8      sub_rectangles = rectangles.split(wires_at_layer_above)        // Delimitation
9      new_sub_rectangles = empty_list()
10     for r ∈ sub_rectangles do
11         current_layer ← layer_above
12         layer_above ← current_layer + 1
13         height ← Design.get_distance_between_layers(current_layer, layer_above)
14         r.update_projection_angles(segment)
15         r.project_up(height)                                       // Projection
16         wires_at_layer_above ← Design.get_wires_at_layer(layer_above)
             new_sub_rectangles.add(r.split(wires_at_layer_above))
17     sub_rectangles = new_sub_rectangles
18     while layer_above < top_layer do
19         goto step 9
20 return sub_rectangles
```

where $z$ is the distance between metal layers. It depends on the level of the metal layer and the used technology.

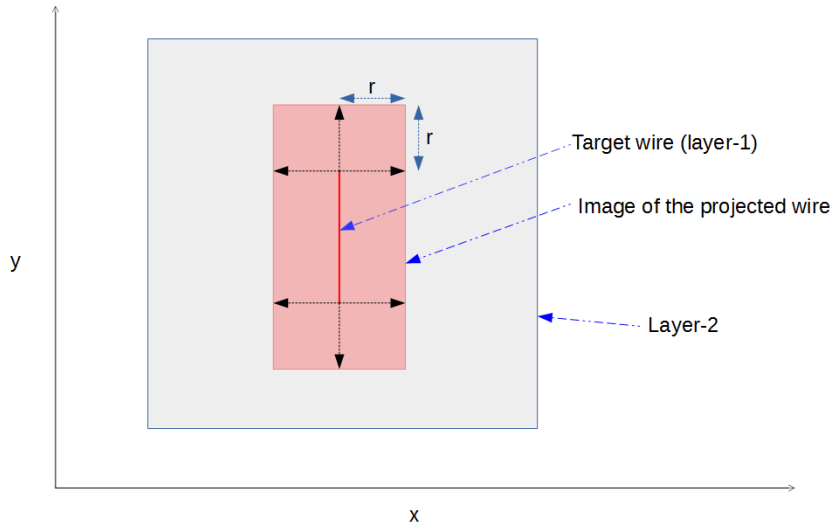



Figure 7.2: First projection of a sensitive wire to the top layer.

The whole area allows accessing the target wire by different angled holes. Figure 7.2 shows the projection phase of a wire located at layer $M1$. The image of the projection gives a rectangle at layer $M2$. We consider that, from any point from this rectangle, the sensitive signal can be accessed by the FIB.

The rectangle may be crossed by some signals located at layer $M2$. Thus, it should be divided into smaller sub-rectangles. This is the second step of our method and will be detailed in the next section.

#### 7.4.4.2 Delimitation

The purpose of the delimitation step is to check if the projected rectangle is crossed by some wires in the layer above. For each wire, we need to split and delimit the area to form other sub-rectangles, thus we obtain a new list of independent areas. Once the delimitation is done as illustrated in fig. 7.2, and the list of rectangles are determined, we can project them again to the layer above, and so on, to reach the first layout.

In this step, we can eliminate the region where the diameter of the hole exceeds the size of the area (we cannot mill through this area without completely cutting a wire).

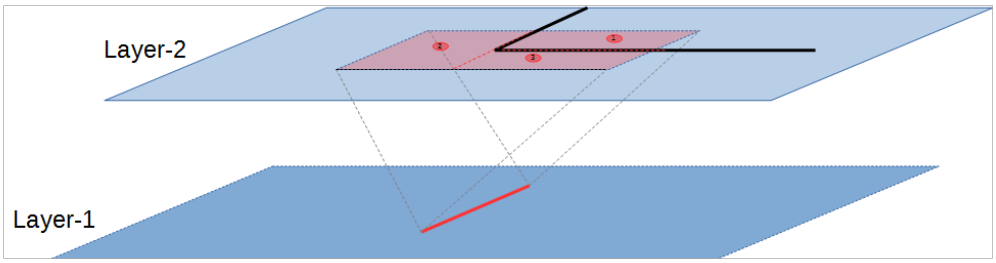



Figure 7.3: The projected area is crossed by one wire. It will be divided into small rectangles.

The projection angle has to be determined by the limits of the targeted wire, and the maximum realisable angle. We illustrate in fig. 7.4 the process of the projection of each area. Each rectangle becomes independent, and the accessibility of the signal should be determined by the projection path. In fact, many rectangles can be projected to some zone and make a bigger area, but this should not be considered as a contiguous one. The angles of projection for each sub-rectangle should take into a account its location.

The angles of projection also depend on their location. For each rectangle, this angle is determined by either its maximal value ($\theta_{max} = \theta^*$), or the extremities of the targeted wire and the rectangle location, as illustrated in fig. 7.4 in green. Therefore, each area has its own projection angle computed after its creation.

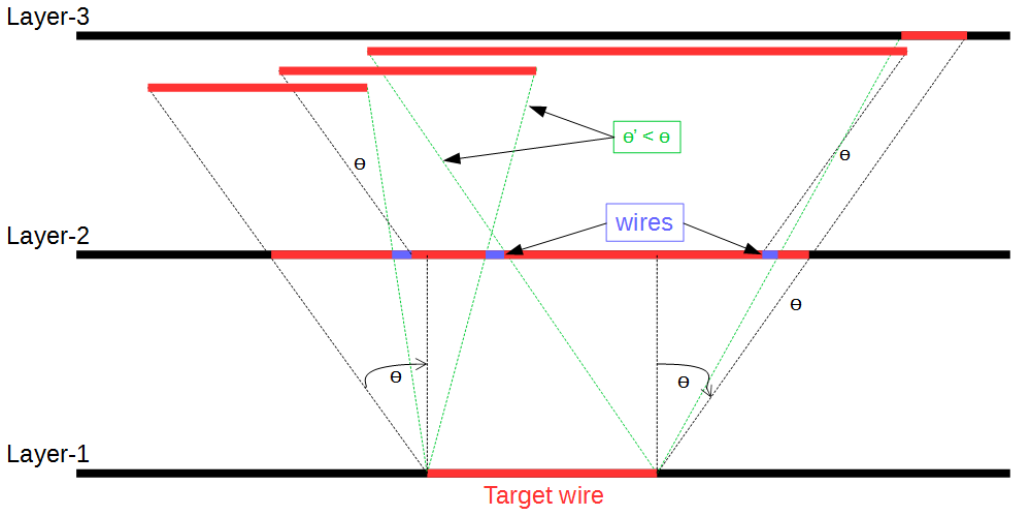



Figure 7.4: Cross-section of projected sensitive wire to the top layers: The projection angle $\theta$ is adapted following each situation.

#### 7.4.4.3 FIB model

Once the phase of projection and delimitation are done, one needs to see how much is difficult to access the sensitive wire. This basically depends on two parameters; the surface of the access path and the performance of the FIB. Obviously, the larger the surface is, the easier the access is. So as a priority, we will sort all the available access paths according to their surfaces. It allows us to find the optimal set-up to access the sensitive wire. Once this phase is completed, we can estimate the setting of the FIB as well as the complexity of milling (or milling time).

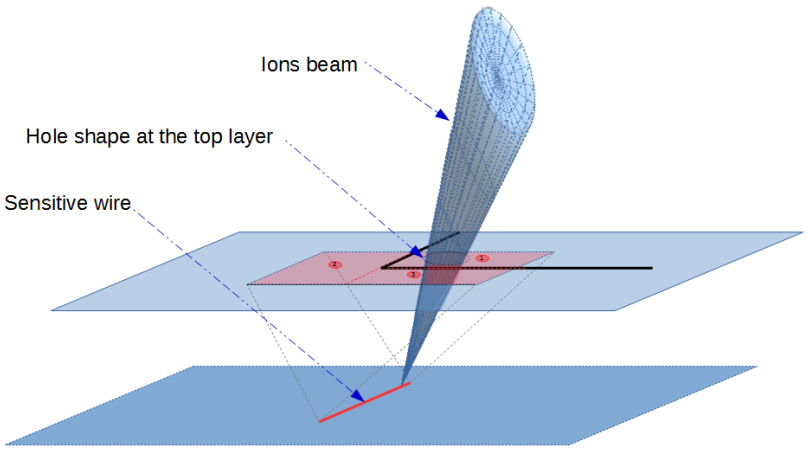



Figure 7.5: Illustration of the FIB model for milling.

Depending on the best found area, we can determine the shape and the volume of the optimal cone that allows to access the sensitive wire and thus, fix the voltage and the current

of the ions beam. With this information we can estimate the time needed to make the hole.

## 7.5 Study-case on AES

To demonstrate the reliability of our methodology on a concrete case. We apply our method to evaluate an ASIC circuit, implementing an AES protected with an active shield.

### 7.5.1 Target IP

The circuit is composed of different IPs including AES, a Physical Unclonable Function (PUF), Digital sensors and also an active shield used to protect the circuit against probing attacks.


(a) (b)

Figure 7.6: The circuit used for the evaluation: (a) Logic part of different IPs, (b) Shield mesh located at top-most metal layer [101].

An overview of this design is presented in fig. 7.6. It is composed of 8 IPs, particularly, an active shield, an AES, a PUF and two digital sensors. The active shield (described in [101]) is composed of three parts:

- ALICE (transmitter), which embeds a SIMON block cipher to generate 128 random bits.
- BOB (receiver), which also embeds a SIMON block cipher.
- Shield mesh (Figure 7.6 (b), which is composed of $n$ lines on the last metal layer. It is used as a communication channel between ALICE and BOB, and achieves the anti-tamper protection of the integrated circuit located below it, with a $128$ bits comparator.

This design uses the CMOS $65\ nm$ technology from STMicroelectronics. The core size is $560\ \mu m \times 560\ \mu m$. The shield mesh is composed of 640 parallel lines with $0.4\ \mu m$ width and $0.4\ \mu m$ spacing.

### 7.5.2 Sensitive signal location

To identify the sensitive signals, we run a leakage detection analysis with the NICV as described in section 7.4.1, using the intermediate value computed by the S-box. There are $9448$ signals (wires) at all in the AES block (without counting logic gates). After the analysis, we have only $256$ sensitive signals, which correspond to the output of the S-box, and the input of *MixColumns*, as detailed in table 7.1.

Table 7.1: Result of parsing and sensitive signal identification.

| Block | #Signals | #Sensitive signals |
|---|---|---|
| AES | 9448 | 256 |
| S-box | 6511 | 128 |
| *MixColumns* | 268 | 128 |

Therefore, it is those signals that are vulnerable against a probing attack. We note that the *ShiftRow* block is not present in the design, as it is just a wiring of the S-box output into the input of *MixColumns*.

### 7.5.3 FIB-probing evaluation

We have selected the output of the S-box. This signal is routed over layers $M3$, $M4$ and $M5$. To compare the FIB attack with an implementation without shield, we consider only the metals at levels lower than $6$. For the performance of the FIB, we have fixed the ratio to $5$ ($R_{FIB} = 5$). The criticality of a probing attack can be measured by the number of exposed areas, their surfaces and the angle to the target wire. The larger the angle is (compared to the normal angle), the greater the relative hole depth becomes. Thus, more time will be needed to complete the hole.

To heuristically estimate the difficulty of the FIB attack, we have defined a metric taking the different parameters into account, namely the surface of each exposed area and its relative depth. The bigger the area is, the easier the attack is. Moreover, the bigger the angle (or the depth) is, the more the attack is difficult. Hence, this heuristic $I$ can be calculated as follows:

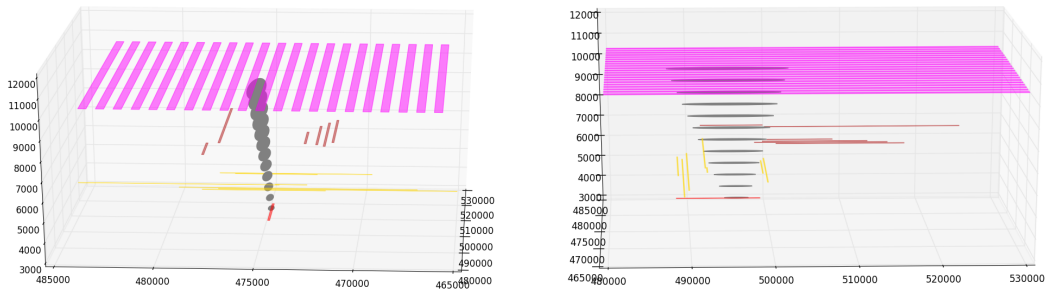
$$\begin{aligned} I_i &= \frac{R_i}{D_i} \\ I &= max_{I_i}\{I_i\} \end{aligned} \tag{7.1}$$

where $R_i$ are the exposed rectangles surfaces, and $D_i$ is the relative depth from $R_i$ to the sensitive signal. This latter is computed from the center of the rectangle. The larger $I$ is, the easier the probing attack is.

Table 7.2: Results for different angles. For each angle we show the number of exposed areas and the value of $I$ $(\mu m)$ (eq. (7.1)).

| Implementation \ $\theta_{max}$ | $\frac{\pi}{3}$ | $\frac{\pi}{4}$ | $\frac{\pi}{6}$ |
|---|---|---|---|
| w/t shield | 143 (23.784) | 39 (21.632) | 16 (13.543) |
| w shield (M7) | 525 (2.101) | 142 (1.643) | 61 (1.635) |

We reported in table 7.2, the number of exposed area for different realisable angles. These angles can be chosen by the evaluator relatively to the capacity of the FIB station. The targeted segment of the sensitive signal is the one at level $M3$. We can see that the number of exposed areas is higher at $M7$, because each exposed area at $M6$ will further be divided at $M7$ according to the shield wires, but the surfaces are smaller. The indicator $I$ is significantly lower when considering $M7$ (as expected). This shows that the attack becomes difficult at $M7$, but still feasible with the chosen ratio in this case ($R_{FIB} = 5$). The exposed areas that do not verify the FIB ratio are ignored. Furthermore, for bigger angles the indicator is bigger, because more susceptible (larger) areas can be found, with a relative low depth.




(a) (b)

Figure 7.7: Best area for milling. The sensitive signal is presented at layer $M3$. The path of the hole is presented as small (gray) ellipses. (a) front side section, (b) left side section

For a signal taking the output of the S-box, we illustrate in fig. 7.7 the best exposed area for the attacker to mil. Interestingly, at this position, there is no much signals at layer $M6$. This allows us to get larger exposed areas when running algorithm 2. As we can see, the hole could have an ellipsis shape ($0.800\mu m \times 12.8\mu m$). As there is no wire at layer $M6$, the hole can be extended further (if needed) along the shield wire direction and thus, allow making a deeper

hole. As we can see in this evaluation, the shield did not provide significant protection. We note an improvement in the difficulty of the attack in the case where no shield is added, but the attack remains feasible and it is only the depth of the hole which increases, without making its realization impossible with the chosen ratio.

### 7.5.4 Security improvements

To see possible improvements, we can imagine adding a second layer of a shield ($M8$). We consider two ways for that:

1. A second parallel shield, but with an *offset* relatively to $M7$.
2. A second *orthogonal* shield with respect to $M7$.

We then calculate the score $I$ to find the best area in both cases. We find that in case (1), there is a very negligible (or even no) improvement. We always get rectangles with a very large length, around $15.8\mu m$ and a width of $0.800\ \mu m$. The latter is limited by the characteristics of the shield (wire width and spacing). The second solution offers more protections. Surfaces with a very large width at $M7$ level are forced to be divided when projected to $M8$. All holes that can be milled from $M8$ must be restricted to a diameter less than $800\ \mu m$ at $M7$. By limiting the diameter, the depth that could be reached is restricted.

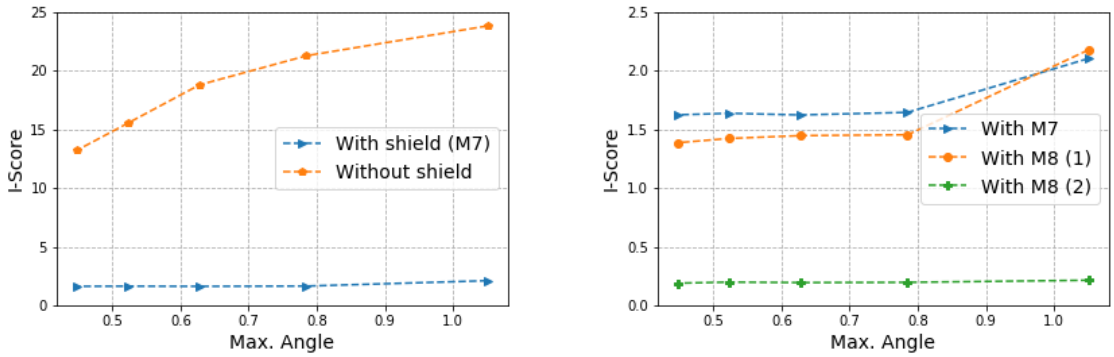
Table 7.3: Evaluation with a second shield $M8$. For each angle we show the value of $I(\mu m)$ (eq. (7.1)).

| M8 \ $\theta_{max}$ | $\frac{\pi}{3}$ | $\frac{\pi}{4}$ | $\frac{\pi}{6}$ |
|---|---|---|---|
| Parallel with offset (1) | 2.174 | 1.452 | 1.421 |
| Orthogonal (2) | 0.214 | 0.196 | 0.198 |

As expected, we can deduce from the value reported in table 7.3, that a second shield with an orthogonal orientation relatively to $M7$ is more efficient. Besides, with the same chosen ratio ($R_{FIB} = 5$), the signal shown in fig. 7.7 cannot be accessed. As the highest diameter that we can achieve at layer $M7$ is less than $0.8\ \mu m$, the ratio of the FIB should be higher than $9$ to be able to access that signal.

In fig. 7.8, we show the improvement of the security level estimated by eq.(7.1) when there is no shield, after the insertion of two parallel shields and then, after the insertion of two orthogonal shields. The results show that the security level increases more significantly with two orthogonal shields.

With this procedure, we can determine the available ways to secure a given implementation against probing attacks. For example, manual re-routing of excessively exposed signals to




(a) With and without shield.

(b) With parallel and orthogonal shields.

Figure 7.8: $I$ score with different shield configuration.

lower levels makes these attacks more difficult as demonstrated in the last test, but still, we can also move other signals (not necessarily sensitive ones) in empty areas above the sensitive signals, which force the size of the exposed areas to be reduced.

### 7.5.5 Discussion

In [133], the authors described a methodology to evaluate a shield against probing attack. They demonstrate on some state-of-the-art implementations the effectiveness of their approach. Following the same idea, we built a new complementary approach, with additional features, to model a more powerful attacker, not only with a very high ratio, but also one who can achieve more complex holes.

In our approach, the exposed areas are delimited according to the presence of wires at each metal layer, by considering a maximum angle allowed to an attacker. This allows us to track all the possible attack paths, and to determine the contribution of the shield on each zone of our implementation. Besides, no interaction is required with the routing tool, and it is fully autonomous. This provides us a way to perform a fact evaluation of custom countermeasures without (re-)running the whole routing process.

We demonstrate our approach on a real implementation of an AES protected with one shield, and we evaluate the different ways that may improve the security of the device. Our results are equivalent to the results exposed in [3], in the sense that it recommends orthogonal shields to – provide more and – enhance security against probing attack.

## 7.6 Conclusion

In this study we have presented an end-to-end methodology, allowing to evaluate a hardware implementation against a probing attack. The selection of sensitive signals is performed automatically, with minimal configuration (random or fixed input). We have shown an example of an attack on an implementation protected by an active shield, considering the parameters of a typical FIB. This later can be adapted to model a more powerful attacker, being able to make smaller holes at higher depth as shown in the state-of-the-art with different techniques. By analysing the possible angles of attack identified exhaustively, the designer can choose to modify the routing in the optimal way according to the performance of a given FIB, such as re-routing over lower metal layers, moving some signals to empty areas, or inserting a second layer of a shield. Besides, our framework is autonomous, and no interaction is required with the routing tool, thus he designer can test some countermeasures and re-routing without launching the full routing process, and estimate the security gains more in advance.

# Chapter 8

# Conclusion & Perspectives

Contents


8.1 Conclusion . . . . . 115
8.2 Perspectives . . . . . 117
8.3 List of Publications . . . . . 119


## 8.1 Conclusion

Side-channel attacks remain a permanent threat against embedded systems, thus reliable protections should be implemented and must be minutely evaluated. In this thesis, we have studied different possible ways to allow better assessment against such threats. We studied in the first part the possibility of carrying out an evaluation to validate a security level on an unprotected and protected hardware implementation. This approach makes it possible to estimate in advance the expected security level on a real circuit. Indeed, thanks to this approach based on digital simulations, more or less obvious leakages can be avoided, and this by going beyond the algorithmic specifications of a countermeasure. It should be noted that others flaws can arise not only because of a mis-integration, but also because of the runtime environment itself, as demonstrated in chapter 4.

We then explore an efficient and more exhaustive way to test a masked implementation against vulnerabilities induced by glitches. We took advantage in this approach to setup a better model of this phenomenon, and to explain the form of the generated leakage by giving its potential equation and a spectral characterisation that can be applied also to real acquisitions. This allows us to explain why standard leakage models are ineffective, and why a prior characterization is required to be able to exploit this kind of flaw. With this better understanding of the

leakage, we were able to design more compact and robust functions against first-order attacks, thing that we have validate on simulated traces and real EM traces. From a design point of view, it is not always easy to know how a signal is protected by a given mask. Therefore, vulnerabilities can be induced not only because of an unintentional complete unmasking, but also because of a combination of signals that depends on some sensitive value with a low bias. This kind of vulnerabilities requires a large number of traces to be able to characterise it and then exploit it, even in simulation mode where the SNR is very high. This may prevent its detection when the number of observations is relatively low.

When the phenomena causing exploitable leakages are known, a formal analysis is a very powerful tool that allows their detection. Besides, it offers in the mean time a way to correct and avoid them. However, such an approach is limited by the adopted models, which limits the spectrum of detectable leakages at this level. It is still necessary to make an empirical assessment at different design life step, to have more visibility on the behavior linked to the technological dispersion. In fact, there are no guarantees that the power consumption will be equivalent for the logic gates which perform the same Boolean function. In addition, current combining can be generated between the different combinatorial blocks, and thus, generates a leakage dependent on a sensitive value [135]. Certainly, this kind of behaviour is not very significant and very hard to detect with measurement probes, which justifies the enhanced security level compared to an unprotected implementation, but it is still something that should be considered in some case. In the second part, we explored active attacks such fault injection and micro-probing attacks.

Firstly, we presented a study about fault injection on a protected hardware implementation with a scheme based on error-detecting code. After synthesis with different options, the detection results vary depending on the optimization criterion. Like all countermeasure based on redundancy, the synthesizer can remove all or small part of the detection block. This, once again justifies the necessity of verifying countermeasures at each design stage, and the advantage of a pre-silicon analysis based on digital simulations. Some fault attacks cannot however be modelled at digital level, such as power glitch, which require a high-level model

Secondly, the vulnerability detection at the post-layout level is essential to check the physical attacks by micro-probing. We have therefore proposed a complete and automated methodology to assess an implementation against micro-probing attack. We mixed side-channel techniques to detect the vulnerable signals and a geometrical concept to analyze their accessibility. The detection of vulnerable signals is based on the NICV metric, but it can be done with other metrics like T-test or other distinguishers. For the signal location step, we proposed a bottom-up process that allows to explore all possible attack paths. It takes into account the FIB ratio

and the maximal angle of the hole. This approach offers a way to estimate the security level by analyzing the accessibility of sensitive signals by a given FIB station.

## 8.2 Perspectives

Modelling an attacker is the most important thing to evaluate an implementation against SCA. It allows us to predict, according to known phenomena, the expected leakage from a target device. Nevertheless, there may be others unknown phenomena that may be risky for certain countermeasure. As already mentioned, technological dispersion is very difficult to control or even impossible, so it would be very interesting to setup models taking into account such phenomena in a relatively reliable way and to project the results on a real target. It may help to see the criticality of this parameter in terms of SCA threat, and detect more vulnerabilities, but also pushes designers and researches to propose more robust countermeasures that support such modelling.

Our formal study about glitches offers a more realistic way to model such phenomena. On the other hand, it is only verified at order $d = 1$, which leaves its extension to high order ($d \geq 2$) as another focus of research worth to explore. In particular, we should explore the complexity of this approach compared to that already existing. As the verification and the design of a secure circuit are two very linked fights, we hope that for a such more reliable approach, we could design more compact circuits while respecting and keeping the expected security level, as demonstrated on the AES S-box.

Formal analysis is also limited to purely software or hardware implementations. However, there are many hybrid implementations mixing software code on one hand for the control instructions, and other hardware blocks to speed up calculations on the other hand. Designing a formal analysis tool capable of dealing with such implementations could be more difficult, in particular when tracking sensitive variables from the software layer to the hardware layer. More instrumentation of the code may be necessary to allow this kind of analyses.

On the other hand, thanks to digital simulation, this remains quite feasible when the description of the implementation is available, and the number of traces is high with respect to the security level. With a progressive approach (RTL, PS and PR), we can detect and correct vulnerabilities very efficiently. In addition to vertical leakages, software implementation can suffer from timing vulnerabilities, which can be detected only when executing the application, and can weaken the effectiveness of a countermeasure, but also offers a simpler path of attack.

In the context of active attacks, such as fault injections and FIB, where the attacker model is stronger, several areas of research could be considered. For the moment, we only support functional injection tests, which do not take into account routing. Our injections are consequently

limited to the internal – explicitly selected – signal modification. A possible improvement would be to take into account the circuit layout, and thus be able to test spatial injection, to test, other mode of injection (such as EM), and other hardware countermeasures (such as digital sensors).

The pre-silicon evaluation against FIB probing attacks is a very recent field, and the publications on this topic are very limited. It would be therefore interesting to move forward on models supporting more routing options and thus, check more exotic countermeasures such as the insertion of a shield (or even combinatorial wires) only on certain parts of the circuit, or alternatively, coerce the routing tool to place the sensitive signals (or hardware block) below the shield meshes. This will facilitate protecting third-party hardware IPs, without a deeper knowledge about the implementation.

## 8.3 List of Publications

### 8.3.1 Book chapter

**Fault Analysis Assisted by Simulation** *Kais Chibani, Adrien Facon, Sylvain Guilley, Damien Marion, Yves Mathieu, Laurent Sauvage , Youssef Souissi & Sofiane Takarabt. (2019) 10.1007/978-3-030-11333-9_12*

### 8.3.2 Journal

**Side-Channel Evaluation Methodology on Software** *Sylvain Guilley, Khaled Karray, Thomas Perianin, Ritu-Ranjan Shrivastwa , Youssef Souissi, & Sofiane Takarabt. Cryptography, 2020, vol. 4, no 4, p. 27.*

### 8.3.3 Conference

**Secure silicon: Towards virtual prototyping** *Laurent Sauvage, Sofiane Takarabt & Youssef Souissi. In : 2017 International Symposium on Electromagnetic Compatibility-EMC EUROPE. IEEE, 2017. p. 1-5.*

**Detection of Side-channel Leakage Through Glitches Using an Automated Tool** *Sofiane Takarabt, Sylvain Guilley, Yves Mathieu, Laurent Sauvage, Youssef Souissi. (2020, April). In International Conference on Defense Systems: Architectures and Technologies (DAT'2020).*

**Cache-timing attacks still threaten IoT devices** *Sofiane Takarabt, Alexander Schaub, Adrien Facon, Sylvain Guilley, Laurent Sauvage, Yves Mathieu, Youssef Souissi. International Conference on Codes, Cryptology, and Information Security. Springer, Cham, 2019.*

**Formal Evaluation and Construction of Glitch-resistant Masked Functions** *Sofiane Takarabt, Sylvain Guilley, Yves Mathieu, Laurent Sauvage, Khaled Karray & Youssef Souissi. HOST, Dec 2021, Virtual, United States.*

**Post-Layout Security Evaluation Methodology Against Probing Attacks** *Sofiane Takarabt, Thuy Xuan Ngo, Youssef Souissi, Sylvain Gueilly, Laurent Sauvage & Yves Mathieu. INISCOM 2021.*

### 8.3.4 Workshop

**Pre-silicon embedded system evaluation as new EDA tool for security verification** *Sofiane Takarabt, Kais Chibani, Adrien Facon, Sylvain Guilley, Yves Mathieu, Laurent Sauvage, Youssef Souissi. In 2018 IEEE 3rd International Verification and Security Workshop (IVSW) (pp. 74-79). IEEE.*

# Appendix A

In this appendix, we give more analyses on the masked $\mathrm{GF}_{16}$ inverter. In appendix A.1, we address the delay insertion countermeasure suggested in chapter 5. We verify the implementation against first and second order DPA. We show that this implementation resits to first order attack.

In appendix A.2, we detail the equations of the masked $\mathrm{GF}_{16}$ inverter, and we make explicit the masked computation of each bit. We study some vulnerabilities, severs, but not too obvious, that can be avoided by designers in appendix A.3.

## A.1 Evaluation of delay insertion countermeasure

We have already clarified in section 5.4.2 the leakage created by glitches in the function illustrated by fig. 5.3, which is the result of the absorbed transitions by the $XOR$ gate. If each transition is evaluated independently, the leakage will be independent from the secret, as it only amplifies the activity created by the input (masked) signals.

It is possible to insert delay elements to carefully make sure that one of the two signal arrives before the other, and accordingly, avoid the leakage at $csb_1$ (caused by $csa_1$ and $mb_1$). The transition caused by the change of $a_0$ at $cst_1$ and $an_1$ are mutually dependent. The joint distribution of the transitions is not independent from $X$ (the combined activity will allow an attacker to recover the secret key). Actually, we can consider this analysis as a second order one, as it combines different time samples. The glitch caused by $an_1$ may leak ($\{n_i\}$) and the glitch caused by $cst_1$ may leak ($\{b_i\}$). Obviously, both combined together will leak ($x_0 \oplus x_1$).

Regarding probing attack, only one probe is necessary to recover the secret, but this needs to combine two observations at two different time samples (multi-variate probing attack). It is worthy to consider also this analysis as a second order one even in the probing model, as the exploitation of this kind of leakage from a passive attacker point of view requires a second order analysis.

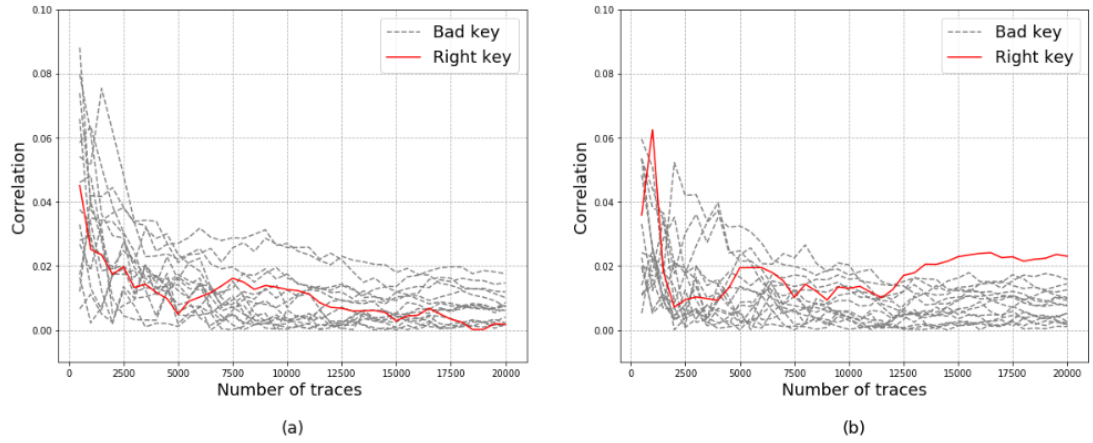



Figure A.1: First (a) and second (b) order CPA on $csb_1$ activity after delay insertion based countermeasure.

In fig. A.1 we show the result of the CPA on the signal $csb_1$ when a delay element is inserted on the signals $cst_1$ and $an_1$. The leakages caused by $an_1$, $mb_1$ and $cst_1$ are separated in time and each Point of Event (PoE) will depend only on one of the three signals.

Unfortunately, this countermeasure is very complicated to extend more: a flaw should also be avoided at the next non-linear function ($GF_2$ multiplier by $(m_0 \oplus m_1)$) that uses the signal $csb_1$. Moreover, if we want to conserve the structure of the design, the signal $csb_1$ should be registered, and the output of the next non-linear layer should also be registered. This leads to a latency of two cycles at least.

## A.2 Masked inversion in $GF_{16}$

In the following, we detail the different steps to consider in order to implement the full $\mathrm{GF}_{16}$ inverter, secure against glitches as explored in appendix A.2.

The inversion of an element $x \in GF_{16}$ can be computed as the following:

$$\begin{aligned} y_0 &= x_1 * x_2 * x_3 \oplus x_0 * x_2 \oplus x_0 * x_3 \oplus x_1 * x_3 \oplus x_2 \\ y_1 &= x_0 * x_2 * x_3 \oplus x_0 * x_3 \oplus x_1 * x_3 \oplus x_2 \oplus x_3 \\ y_2 &= x_0 * x_1 * x_3 \oplus x_0 * x_2 \oplus x_1 * x_2 \oplus x_1 * x_3 \oplus x_0 \\ y_3 &= x_0 * x_1 * x_2 \oplus x_1 * x_2 \oplus x_1 * x_3 \oplus x_1 \oplus x_0 \end{aligned} \tag{A.1}$$

To get a first order Boolean sharing, we should replace each $x_i$ by $a_i \oplus m_i$, hence:

$$\begin{aligned} y_0 &= x_1 * x_2 * x_3 \oplus x_0 * x_2 \oplus x_0 * x_3 \oplus x_1 * x_3 \oplus x_2 \\ &= x_1 * (x_2 * x_3 \oplus x_3) \oplus x_0 * x_2 \oplus x_0 * x_3 \oplus x_2 \\ &= (a_1 \oplus m_1) * (a_2 \oplus m_2) * (a_3 \oplus m_3) \\ &\oplus (a_0 \oplus m_0) * (a_2 \oplus m_2) \oplus (a_0 \oplus m_0) * (a_3 \oplus m_3) \\ &\oplus (a_1 \oplus m_1) * (a_3 \oplus m_3) \oplus (a_2 \oplus m_2) \end{aligned} \tag{A.2}$$

We can follow the TINC property to build a secure version against glitches as shown in section 5.4.3, thus:

$$\begin{aligned} x_1 * x_2 * x_3 &= (a_1 \oplus m_1) * (a_2 \oplus m_2) * (a_3 \oplus m_3) \\ &= a_1 * (a_2 * a_3 \oplus a_2 * m_3 \oplus m_2 * a_3 \oplus m_2 * m_3) \\ &\oplus\ m_1 * (a_2 * a_3 \oplus a_2 * m_3 \oplus m_2 * a_3 \oplus m_2 * m_3) \\ &= a_1 * a_2 * a_3 \\ &\oplus\ a_1 * a_2 * m_3 \\ &\oplus\ a_1 * m_2 * a_3 \\ &\oplus\ a_1 * m_2 * m_3 \\ &\oplus\ m_1 * a_2 * a_3 \\ &\oplus\ m_1 * a_2 * m_3 \\ &\oplus\ m_1 * m_2 * a_3 \\ &\oplus\ m_1 * m_2 * m_3 \end{aligned} \tag{A.3}$$

For each aligned term we can add the shared version of each term given in eq. A.2, such that it still respects TINC. By introducing new fresh random bits, we can reduce the number of total shares as shown in section 5.4.2. Therefore, the terms ($a_3 * m_2$, $a_2 * m_3$) can be merged into one signal as:

$$a_3 * (m_2 \oplus Z_0) \oplus (m_3 \oplus Z_1) \tag{A.4}$$

which will reduce the number of terms from $8$ to $6$ (it holds for both $a_1$ and $m_1$).

In table A.1, we give the full design of the glitch-resistant inversion for any element of $GF_{16}$. Each $y_i$ verify:

$$y_i = \bigoplus_{j=0}^{5} Y_{ij}$$

We remind that the intermediate results $Y_{ij}$ must be *xored* (and then registered) with an independent fresh random $Z_k$, before the compression step that allows to reduce the number of shares from $6$ to $2$. When integrated into the AES S-box, the fresh random $Z_i$ used for each $y_i$ should be different and independent.

## A.3 Uniformity and mask reuse

Another question that one could ask about Boolean masking concerns the distribution of the intermediate value of the signals, and in which case are we allowed to reuse the masks between cycles? For that, let's consider the gadget illustrated in figure A.2.

The circuit performs the following calculation ($x = a \oplus m, y = b \oplus n$):

Table A.1: Glitch-resistant masked inverter of any element $x = (a \oplus m) \in GF_{16}$.

| | |
|---|---|
| $y_0$ | $Y_{00} = a_1 * (a_2 * (m_3 \oplus Z_1) \oplus a_3 * (m_2 \oplus Z_2))$<br>$Y_{01} = a_1 * (a_2 * a_3 \oplus a_2 * Z_1 \oplus a_3 * Z_2) \oplus a_0 * a_2 \oplus a_0 * a_3 \oplus a_1 * a_3 \oplus a_2$<br>$Y_{02} = a_1 * m_2 * m_3 \oplus a_0 * m_2 \oplus a_1 * m_3 \oplus a_0 * m_3$<br>$Y_{03} = m_1 * (a_2 * (m_3 \oplus Z_1) \oplus a_3 * (m_2 \oplus Z_2))$<br>$Y_{04} = m_1 * (a_2 * a_3 \oplus a_2 * Z_1 \oplus a_3 * Z_2) \oplus m_0 * a_2 \oplus m_0 * a_3 \oplus m_1 * a_3$<br>$Y_{05} = m_1 * m_2 * m_3 \oplus m_0 * m_2 \oplus m_0 * m_3 \oplus m_1 * m_3 \oplus m_2$ |
| $y_1$ | $Y_{10} = a_3 * (a_0 * (m_2 \oplus Z_1) \oplus a_2 * (m_0 \oplus Z_2))$<br>$Y_{11} = a_3 * (a_2 * a_0 \oplus a_0 * Z_1 \oplus a_2 * Z_2 \oplus a_0 \oplus a_1) \oplus a_3$<br>$Y_{12} = a_3 * (m_0 * m_2 \oplus m_0 \oplus m_1) \oplus m_2$<br>$Y_{13} = m_3 * (a_0 * (m_2 \oplus Z_1) \oplus a_2 * (m_0 \oplus Z_2))$<br>$Y_{14} = m_3 * (a_2 * a_0 \oplus a_0 * Z_1 \oplus a_2 * Z_2 \oplus a_0 \oplus a_1) \oplus a_2$<br>$Y_{15} = m_3 * (m_0 * m_2 \oplus m_0 \oplus m_1) \oplus m_3$ |
| $y_2$ | $Y_{20} = a_3 * (a_0 * (m_1 \oplus Z_1) \oplus a_1 * (m_0 \oplus Z_2))$<br>$Y_{21} = a_3 * (a_0 * Z_1 \oplus a_1 * Z_2 \oplus a_0 * a_1 \oplus a_1) \oplus a_2 * (a_0 \oplus a_1) \oplus a_0$<br>$Y_{22} = a_3 * (m_0 * m_1 \oplus m_1) \oplus a_2 * (m_0 \oplus m_1)$<br>$Y_{23} = m_3 * (a_0 * (m_1 \oplus Z_1) \oplus a_1 * (m_0 \oplus Z_2))$<br>$Y_{24} = m_3 * (a_0 * Z_1 \oplus a_1 * Z_2 \oplus a_0 * a_1 \oplus a_1) \oplus m_2 * (a_0 \oplus a_1)$<br>$Y_{25} = m_3 * (m_0 * m_1 \oplus m_1) \oplus m_2 * (m_0 \oplus m_1) \oplus m_0$ |
| $y_3$ | $Y_{30} = a_1 * (a_0 * (m_2 \oplus Z_1) \oplus a_2 * (m_0 \oplus Z_2))$<br>$Y_{31} = a_1 * (a_2 * a_0 \oplus a_0 * Z_1 \oplus a_2 * Z_2 \oplus a_2 \oplus a_3) \oplus a_1$<br>$Y_{32} = a_1 * (m_0 * m_2 \oplus m_2 \oplus m_3)$<br>$Y_{33} = m_1 * (a_0 * (m_2 \oplus Z_1) \oplus a_2 * (m_0 \oplus Z_2))$<br>$Y_{34} = m_1 * (a_2 * a_0 \oplus a_0 * Z_1 \oplus a_2 * Z_2 \oplus a_2 \oplus a_3) \oplus m_1 \oplus a_0$<br>$Y_{35} = m_1 * (m_0 * m_2 \oplus m_2 \oplus m_3) \oplus m_0$ |

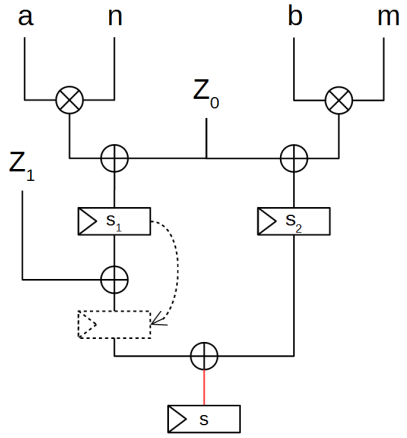



Figure A.2: Vulnerable circuit against glitches. The inserted registered do not prevent the leakage. To secure this circuit the register $s_1$ should be moved (as shown with the dashed lines).

$$\begin{aligned} s_1 &= a * n \oplus Z_0 \\ s_2 &= b * m \oplus Z_0 \\ s &= (s_1 \oplus Z_1) \oplus s_2 \\ &= a * n \oplus b * m \oplus Z_1 \end{aligned} \tag{A.5}$$

In terms of value, the signal $S$ is protected by $Z_1$. Despite $s_1$ and $s_2$ being registered before their reduction on $s$, the circuit can leak information dependent on ($x$, $y$). Indeed, if we consider the transition $s \rightarrow s'$ (for the same $z_1$), we have:

$$\begin{aligned} s \oplus s' &= a * n \oplus b * m \oplus a' * n' \oplus b' * m' \\ &= x * n \oplus y * m \oplus a' * n' \oplus b' * m' \end{aligned} \tag{A.6}$$

which depends on ($x$, $y$). The first reason is that, the term ($a * n \oplus b * m$) depends on ($x$, $y$), and its distribution is not uniform. The second is that, the term ($a' * n' \oplus b' * m'$) acts as a mask, but its distribution is not uniform (although it is assumed to be unknown to an attacker). This is equivalent to protecting sensitive variable with a non-uniform mask. To fix this vulnerability, the registering stage should be performed after resharing by $Z_1$. In this case, $Z_0$ is not required any more.

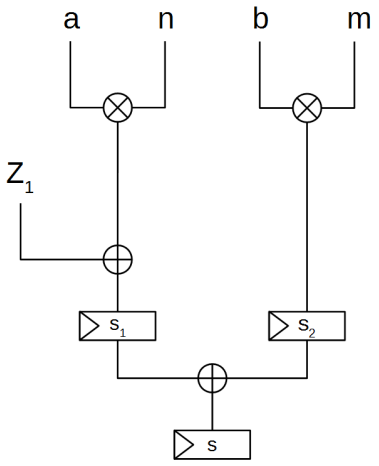



Figure A.3: Secure version of circuit fig. A.2. In this case, the fresh random $Z_1$ should be updated at each cycle.

This version is presented in fig. A.3. Another solution is to use different fresh random to reshare $s_1$ and $s_2$. In a practical case, this issue should be prevented at the last multiplier of the S-box. When considering the secure version presented in fig. A.3, the circuit leaks ($x$, $y$) if $Z_1$ is reused between two cycles. The reason of the leakage is the same as the one given in eq. A.6, as $Z_1$ will be simplified and the term ($a' * n' \oplus b' * m'$) is not uniform.

Those two cases are very likely to happen in a real circuit when the intermediate value is not obvious to the designer, or when reusing some randomness to enhance the performance of the design. We should also notice that this kind of leakage is not very significant compared with a non-masked implementation. As the sensitive information ($x$, $y$) is revealed only in some cases (when it takes the zero-value ($0$, $0$)), more observations are needed to see a significant peak when performing a T-test or an NICV analyses.

# Bibliography


[1] Y. Nakano, Y. Souissi, R. Nguyen, L. Sauvage, J.-L. Danger, S. Guilley, S. Kiyomoto, and Y. Miyake, "A pre-processing composition for secret key recovery on android smartphone," in *IFIP International Workshop on Information Security Theory and Practice*, pp. 76–91, Springer, 2014.

[2] E. Trichina, "Combinational logic design for aes subbyte transformation on masked data.," *IACR Cryptol. EPrint Arch.*, vol. 2003, p. 236, 2003.

[3] H. Wang, Q. Shi, D. Forte, and M. M. Tehranipoor, "Probing assessment framework and evaluation of antiprobing solutions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1239–1252, 2019.

[4] "Nist. data encyrption standard," 1999.

[5] J. Daemen and V. Rijmen, "Reijndael: The advanced encryption standard.," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 26, no. 3, pp. 137–139, 2001.

[6] C. E. Shannon, "Communication theory of secrecy systems," *The Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[7] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[8] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[9] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*, pp. 104–113, Springer, 1996.

[10] C. Arnaud and P.-A. Fouque, "Timing attack against protected rsa-crt implementation used in polarssl," in *Cryptographers' Track at the RSA Conference*, pp. 18–33, Springer, 2013.

[11] C. Chen, T. Wang, and J. Tian, “Improving timing attack on rsa-crt via error detection and correction strategy,” *Information Sciences*, vol. 232, pp. 464–474, 2013.

[12] D. J. Bernstein, “Cache-timing attacks on aes,” 2005.

[13] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, “Power analysis attacks of modular exponentiation in smartcards,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 144–157, Springer, 1999.

[14] M. Dugardin, S. Guilley, J.-L. Danger, Z. Najm, and O. Rioul, “Correlated extra-reductions defeat blinded regular exponentiation,” in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 3–22, Springer, 2016.

[15] P.-A. Fouque and F. Valette, “The doubling attack–why upwards is better than downwards,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 269–280, Springer, 2003.

[16] C. D. Walter, “Sliding windows succumbs to big mac attack,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 286–299, Springer, 2001.

[17] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Annual international cryptology conference*, pp. 388–397, Springer, 1999.

[18] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*, vol. 31. Springer Science & Business Media, 2008.

[19] E. Brier, C. Clavier, and F. Olivier, “Correlation power analysis with a leakage model,” in *International workshop on cryptographic hardware and embedded systems*, pp. 16–29, Springer, 2004.

[20] Y. Souissi, N. Debande, S. Mekki, S. Guilley, A. Maalaoui, and J.-L. Danger, “On the optimality of correlation power attack on embedded cryptographic systems,” in *IFIP International Workshop on Information Security Theory and Practice*, pp. 169–178, Springer, 2012.

[21] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, “Mutual information analysis,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 426–442, Springer, 2008.

[22] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon, “Mutual information analysis: a comprehensive study,” *Journal of Cryptology*, vol. 24, no. 2, pp. 269–291, 2011.

[23] C. Whitnall, E. Oswald, and L. Mather, "An exploration of the kolmogorov-smirnov test as a competitor to mutual information analysis," in *International Conference on Smart Card Research and Advanced Applications*, pp. 234–251, Springer, 2011.

[24] C. Whitnall and E. Oswald, "A fair evaluation framework for comparing side-channel distinguishers," *Journal of Cryptographic Engineering*, vol. 1, no. 2, pp. 145–160, 2011.

[25] S. Mangard, "Hardware countermeasures against dpa–a statistical analysis of their effectiveness," in *Cryptographers' Track at the RSA Conference*, pp. 222–235, Springer, 2004.

[26] M. Rivain, "On the exact success rate of side channel analysis in the gaussian model," in *International Workshop on Selected Areas in Cryptography*, pp. 165–183, Springer, 2008.

[27] Y. Fei, Q. Luo, and A. A. Ding, "A statistical model for dpa with novel algorithmic confusion analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 233–250, Springer, 2012.

[28] V. Lomné, E. Prouff, M. Rivain, T. Roche, and A. Thillard, "How to estimate the success rate of higher-order side-channel attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 35–54, Springer, 2014.

[29] E. Prouf, "Efficient evaluation of the success rate in side channel attacks," in *H2020 Project REASSURE*, 2018.

[30] A. Thillard, E. Prouff, and T. Roche, "Success through confidence: Evaluating the effectiveness of a side-channel attack," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 21–36, Springer, 2013.

[31] S. Guilley, A. Heuser, and O. Rioul, "A key to success: success exponents for side-channel distinguishers (extended version of [15])," tech. rep., Cryptology ePrint Archive, Report 2016/987, 2016.

[32] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 443–461, Springer, 2009.

[33] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 13–28, Springer, 2002.

[34] C. Rechberger and E. Oswald, “Practical template attacks,” in *International Workshop on Information Security Applications*, pp. 440–456, Springer, 2004.

[35] W. Schindler, K. Lemke, and C. Paar, “A stochastic model for differential side channel cryptanalysis,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 30–46, Springer, 2005.

[36] A. Moradi, O. Mischke, and T. Eisenbarth, “Correlation-enhanced power analysis collision attack,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 125–139, Springer, 2010.

[37] T. Roche and V. Lomné, “Collision-correlation attack against some 1 st-order boolean masking schemes in the context of secure devices,” in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 114–136, Springer, 2013.

[38] E. Prouff, M. Rivain, and R. Bevan, “Statistical analysis of second order differential power analysis,” *IEEE Transactions on computers*, vol. 58, no. 6, pp. 799–811, 2009.

[39] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, “Nicv: normalized inter-class variance for detection of side-channel leakage,” in *Electromagnetic Compatibility, Tokyo (EMC’14/Tokyo), 2014 International Symposium on*, pp. 310–313, IEEE, 2014.

[40] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, “Side-channel leakage and trace compression using normalized inter-class variance,” in *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*, pp. 1–9, 2014.

[41] E. Oswald and S. Mangard, “Template attacks on masking—resistance is futile,” in *Cryptographers’ Track at the RSA Conference*, pp. 243–256, Springer, 2007.

[42] M. Tunstall, C. Whitnall, and E. Oswald, “Masking tables—an underestimated security risk,” in *International Workshop on Fast Software Encryption*, pp. 425–444, Springer, 2013.

[43] J. Pan, J. Den Hartog, and J. Lu, “You cannot hide behind the mask: Power analysis on a provably secure s-box implementation,” in *International Workshop on Information Security Applications*, pp. 178–192, Springer, 2009.

[44] H. Saputra, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and R. Brooks, “Masking the energy behaviour of encryption algorithms,” *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 5, pp. 274–284, 2003.

[45] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev, "Improving the security of dual-rail circuits," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 282–297, Springer, 2004.

[46] T. Popp and S. Mangard, "Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 172–186, Springer, 2005.

[47] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Annual International Cryptology Conference*, pp. 398–412, Springer, 1999.

[48] J.-S. Coron and L. Goubin, "On boolean and arithmetic masking against differential power analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 231–237, Springer, 2000.

[49] T. S. Messerges, "Securing the aes finalists against power analysis attacks," in *International Workshop on Fast Software Encryption*, pp. 150–164, Springer, 2000.

[50] J. Waddle and D. Wagner, "Towards efficient second-order power analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 1–15, Springer, 2004.

[51] E. Trichina, D. De Seta, and L. Germani, "Simplified adaptive multiplicative masking for aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 187–197, Springer, 2002.

[52] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the aes s-box," in *International workshop on fast software encryption*, pp. 413–423, Springer, 2005.

[53] L. Genelle, E. Prouff, and M. Quisquater, "Secure multiplicative masking of power functions," in *International Conference on Applied Cryptography and Network Security*, pp. 200–217, Springer, 2010.

[54] L. De Meyer, O. Reparaz, and B. Bilgin, "Multiplicative masking for aes in hardware," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 431–468, 2018.

[55] S. Mangard and K. Schramm, "Pinpointing the side-channel leakage of masked aes hardware implementations," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 76–90, Springer, 2006.

[56] D. Canright, "A very compact s-box for aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 441–455, Springer, 2005.

[57] E. Oswald, S. Mangard, and N. Pramstaller, "Secure and efficient masking of aes-a mission impossible?," *IACR Cryptol. ePrint Arch.*, vol. 2004, p. 134, 2004.

[58] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," in *Advances in Cryptology — ASIACRYPT 2001* (C. Boyd, ed.), (Berlin, Heidelberg), pp. 239–254, Springer Berlin Heidelberg, 2001.

[59] D. Canright and L. Batina, "A very compact "perfectly masked" s-box for aes," in *International Conference on Applied Cryptography and Network Security*, pp. 446–459, Springer, 2008.

[60] J. Blömer, J. Guajardo, and V. Krummel, "Provably secure masking of aes," in *International workshop on selected areas in cryptography*, pp. 69–83, Springer, 2004.

[61] S. Mangard, T. Popp, and B. M. Gammel, "Side-channel leakage of masked cmos gates," in *Cryptographers' Track at the RSA Conference*, pp. 351–365, Springer, 2005.

[62] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked aes hardware implementations," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 157–171, Springer, 2005.

[63] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *International conference on information and communications security*, pp. 529–545, Springer, 2006.

[64] S. Nikova, V. Rijmen, and M. Schläffer, "Secure hardware implementation of nonlinear functions in the presence of glitches," *Journal of Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.

[65] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of aes," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 69–88, Springer, 2011.

[66] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "A more efficient aes threshold implementation," in *International Conference on Cryptology in Africa*, pp. 267–284, Springer, 2014.

[67] A. Moradi and O. Mischke, "Glitch-free implementation of masking in modern fpgas," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 89–95, IEEE, 2012.

[68] M. Alam, S. Ghosh, M. Mohan, D. Mukhopadhyay, D. R. Chowdhury, and I. Gupta, "Effect of glitches against masked aes s-box implementation and countermeasure," *IET Information Security*, vol. 3, no. 1, pp. 34–44, 2009.

[69] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "Higher-order threshold implementations," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 326–343, Springer, 2014.

[70] V. Arribas, S. Nikova, and V. Rijmen, "Vermi: Verification tool for masked implementations," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 381–384, IEEE, 2018.

[71] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order.," *IACR Cryptology ePrint Archive*, vol. 2016, p. 486, 2016.

[72] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating masking schemes," in *Annual Cryptology Conference*, pp. 764–783, Springer, 2015.

[73] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *Annual International Cryptology Conference*, pp. 463–481, Springer, 2003.

[74] T. S. Messerges, "Using second-order power analysis to attack dpa resistant software," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 238–251, Springer, 2000.

[75] M. Joye, P. Paillier, and B. Schoenmakers, "On second-order differential power analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 293–308, Springer, 2005.

[76] E. Oswald, S. Mangard, C. Herbst, and S. Tillich, "Practical second-order dpa attacks for masked smart card implementations of block ciphers," in *Cryptographers' Track at the RSA Conference*, pp. 192–207, Springer, 2006.

[77] E. Peeters, F.-X. Standaert, N. Donckers, and J.-J. Quisquater, "Improved higher-order side-channel attacks with fpga experiments," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 309–323, Springer, 2005.

[78] A. Moss, E. Oswald, D. Page, and M. Tunstall, "Compiler assisted masking," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 58–75, Springer, 2012.

[79] A. G. Bayrak, F. Regazzoni, D. Novo, and P. Ienne, "Sleuth: Automated verification of software power analysis countermeasures," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 293–310, Springer, 2013.

[80] H. Eldib, C. Wang, and P. Schaumont, "Formal verification of software countermeasures against side-channel attacks," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 2, pp. 1–24, 2014.

[81] G. Bertoni, M. Martinoli, and M. C. Molteni, "A methodology for the characterisation of leakages in combinatorial logic," *Journal of Hardware and Systems Security*, vol. 1, no. 3, pp. 269–281, 2017.

[82] R. Bloem, H. Groß, R. Iusupov, B. Könighofer, S. Mangard, and J. Winter, "Formal verification of masked hardware implementations in the presence of glitches," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 321–353, Springer, 2018.

[83] R. Bloem, H. Groß, R. Iusupov, M. Krenn, and S. Mangard, "Sharing independence & relabeling: Efficient formal verification of higher-order masking.," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 1031, 2018.

[84] G. Barthe, S. Belaïd, G. Cassiers, P.-A. Fouque, B. Grégoire, and F.-X. Standaert, "maskverif: Automated verification of higher-order masking in presence of physical defaults," in *European Symposium on Research in Computer Security*, pp. 300–318, Springer, 2019.

[85] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini, "Strong non-interference and type-directed higher-order masking," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 116–129, 2016.

[86] M. Kirschbaum and T. Popp, *Evaluation of power estimation methods based on logic simulations*. na, 2007.

[87] S. Bhasin, J.-L. Danger, T. Graba, Y. Mathieu, D. Fujimoto, and M. Nagata, "Physical security evaluation at an early design-phase: A side-channel aware simulation methodology," in *Proceedings of International Workshop on Engineering Simulations for Cyber-Physical Systems*, pp. 13–20, 2013.

[88] S. Soydan, "Analyzing the dpa leakage of the masked s-box via digital simulation and reducing the leakage by inserting delay cells," in *2010 Fourth International Conference on Emerging Security Information, Systems and Technologies*, pp. 221–227, IEEE, 2010.

[89] N. Veshchikov, "Silk: high level of abstraction leakage simulator for side channel analysis," in *Proceedings of the 4th Program Protection and Reverse Engineering Workshop*, pp. 1–11, 2014.

[90] O. Reparaz, "Detecting flawed masking schemes with leakage detection tests," in *International Conference on Fast Software Encryption*, pp. 204–222, Springer, 2016.

[91] V. Lomné, E. Prouff, and T. Roche, "Behind the scene of side channel attacks," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 506–525, Springer, 2013.

[92] P. Bottinelli and J. W. Bos, "Computational aspects of correlation power analysis," *Journal of Cryptographic Engineering*, vol. 7, no. 3, pp. 167–181, 2017.

[93] D. Samyde, S. Skorobogatov, R. Anderson, and J.-J. Quisquater, "On a new way to read data from memory," in *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pp. 65–69, IEEE, 2002.

[94] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *International conference on the theory and applications of cryptographic techniques*, pp. 37–51, Springer, 1997.

[95] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Annual international cryptology conference*, pp. 513–525, Springer, 1997.

[96] P. Dusart, G. Letourneux, and O. Vivolo, "Differential fault analysis on aes," in *International Conference on Applied Cryptography and Network Security*, pp. 293–306, Springer, 2003.

[97] G. Piret and J.-J. Quisquater, "A differential fault attack technique against spn structures, with application to the aes and khazad," in *International workshop on cryptographic hardware and embedded systems*, pp. 77–88, Springer, 2003.

[98] D. Mukhopadhyay, "An improved fault based attack of the advanced encryption standard," in *International Conference on Cryptology in Africa*, pp. 421–434, Springer, 2009.

[99] R. Lashermes, G. Reymond, J.-M. Dutertre, J. Fournier, B. Robisson, and A. Tria, "A DFA on AES Based on the Entropy of Error Distributions," in *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography* (IEEE, ed.), pp. 34–43, Sept 2012.

[100] P. Maistri, R. Leveugle, L. Bossuet, A. Aubert, V. Fischer, B. Robisson, N. Moro, P. Maurine, J.-M. Dutertre, and M. Lisart, "Electromagnetic analysis and fault injection onto secure circuits," in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, IEEE, 2014.

[101] J.-M. Cioranesco, J.-L. Danger, T. Graba, S. Guilley, Y. Mathieu, D. Naccache, and X. T. Ngo, "Cryptographically secure shields," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 25–31, IEEE, 2014.

[102] M. Y. Ali, W. Hung, and F. Yongqi, "A review of focused ion beam sputtering," *International journal of precision engineering and manufacturing*, vol. 11, no. 1, pp. 157–170, 2010.

[103] Y. Fu and K. A. B. Ngoi, "Investigation of aspect ratio of hole drilling from micro to nanoscale via focused ion beam fine milling," 2005.

[104] V. Sidorkin, E. van Veldhoven, E. van der Drift, P. Alkemade, H. Salemink, and D. Maas, "Sub-10-nm nanolithography with a scanning helium beam," *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena*, vol. 27, no. 4, pp. L18–L20, 2009.

[105] H. Wu, L. Stern, D. Xia, D. Ferranti, B. Thompson, K. Klein, C. Gonzalez, and P. Rack, "Focused helium ion beam deposited low resistivity cobalt metal lines with 10 nm resolution: implications for advanced circuit editing," *Journal of Materials Science: Materials in Electronics*, vol. 25, no. 2, pp. 587–595, 2014.

[106] F. S. Jamaludin, M. F. M. Sabri, and S. M. Said, "Controlling parameters of focused ion beam (fib) on high aspect ratio micro holes milling," *Microsystem technologies*, vol. 19, no. 12, pp. 1873–1888, 2013.

[107] H.-W. Li, D.-J. Kang, M. Blamire, and W. T. Huck, "Focused ion beam fabrication of silicon print masters," *Nanotechnology*, vol. 14, no. 2, p. 220, 2003.

[108] J. Zhou and G. Yang, "Focused ion-beam based nanohole modeling, simulation, fabrication, and application," *Journal of manufacturing science and engineering*, vol. 132, no. 1, 2010.

[109] H. Luo, H. Wang, Y. Cui, and R. Wang, "Focused ion beam built-up on scanning electron microscopy with increased milling precision," *Science China Physics, Mechanics and Astronomy*, vol. 55, no. 4, pp. 625–630, 2012.

[110] W. C. Hopman, F. Ay, W. Hu, V. J. Gadgil, L. Kuipers, M. Pollnau, and R. M. de Ridder, "Focused ion beam scan routine, dwell time and dose optimizations for submicrometre period planar photonic crystal components and stamps in silicon," *Nanotechnology*, vol. 18, no. 19, p. 195305, 2007.

[111] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors.," *Smartcard*, vol. 99, pp. 9–20, 1999.

[112] J.-M. Schmidt and C. H. Kim, "A probing attack on aes," in *International Workshop on Information Security Applications*, pp. 256–265, Springer, 2008.

[113] H. Handschuh, P. Paillier, and J. Stern, "Probing attacks on tamper-resistant devices," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 303–315, Springer, 1999.

[114] N. Chang, K. Kim, and H. G. Lee, "Cycle-accurate energy consumption measurement and analysis: Case study of arm7tdmi," in *Proceedings of the 2000 international symposium on Low power electronics and design*, pp. 185–190, 2000.

[115] K. Yamakoshi and A. Yamagishi, "Estimation of cpa attack for aes using simulation method," *Technical Report of IEICE, ISEC*, vol. 109, no. 42, pp. 13–20, 2009.

[116] M. Yoshikawa and T. Asai, "High-level simulation for side channel attacks," in *Proc. of The International MultiConference of Engineers and Computer Scientists*, vol. 2, pp. 1565–1568, Citeseer, 2011.

[117] N. Debande, M. Berthier, Y. Bocktaels, and T.-H. Le, "Profiled model based power simulator for side channel evaluation.," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 703, 2012.

[118] A. Moradi, O. Mischke, and T. Eisenbarth, "Correlation-enhanced power analysis collision attack," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 125–139, Springer, 2010.

[119] G.-Z. Xiao and J. L. Massey, "A spectral characterization of correlation-immune combining functions," *IEEE Transactions on information theory*, vol. 34, no. 3, pp. 569–571, 1988.

[120] H. Groß, R. Iusupov, and R. Bloem, "Generic low-latency masking in hardware," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–21, 2018.

[121] D. Canright and L. Batina, "A Very Compact "Perfectly Masked" S-Box for AES," in *ACNS*, vol. 5037 of *Lecture Notes in Computer Science*, pp. 446–459, 2008.

[122] D. Canright, "David Canright's tiny AES S-boxes." Verilog structural code of the netlist: `https://github.com/coruus/canright-aes-sboxes`.

[123] M. Joye and M. Tunstall, *Fault Analysis in Cryptography*. Springer LNCS, March 2011. DOI: 10.1007/978-3-642-29656-7 ; ISBN 978-3-642-29655-0.

[124] S. Guilley and J.-L. Danger, "Global Faults on Cryptographic Circuits." Chapter 17 of [123].

[125] N. Beringuier-Boher, M. Lacruche, D. El-Baze, J. Dutertre, J. Rigaud, and P. Maurine, "Body Biasing Injection Attacks in Practice," in *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC, Prague, Czech Republic, January 20, 2016*, pp. 49–54, 2016.

[126] N. Selmane, S. Guilley, and J.-L. Danger, "Setup Time Violation Attacks on AES," in *EDCC, The seventh European Dependable Computing Conference*, (Kaunas, Lithuania), pp. 91–96, May 7-9 2008. ISBN: 978-0-7695-3138-0, DOI: 10.1109/EDCC-7.2008.11.

[127] C. Giraud, "DFA on AES," in *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, pp. 27–41, 2004.

[128] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *International workshop on cryptographic hardware and embedded systems*, pp. 2–12, Springer, 2002.

[129] A. Y. Nikiforov and P. Skorobogatov, "Physical principles of laser simulation for the transient radiation response of semiconductor structures, active circuit elements, and circuits: A linear model," *Russian Microelectronics*, vol. 33, no. 2, pp. 68–79, 2004.

[130] J. Breier, D. Jap, and C.-N. Chen, "Laser profiling for the back-side fault attacks: with a practical laser skip instruction attack on aes," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, pp. 99–103, 2015.

[131] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *IEEE transactions on Computers*, vol. 52, no. 4, pp. 492–505, 2003.

[132] G. Bertoni, L. Breveglieri, I. Koren, and V. Piuri, "Fault detection in the advanced encryption standard," in *Proc. Conf. Massively Parallel Computing Systems (MPCS'02)*, pp. 92–97, 2002.

[133] Q. Shi, N. Asadizanjani, D. Forte, and M. M. Tehranipoor, "A layout-driven framework to assess vulnerability of ics to microprobing attacks," in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 155–160, IEEE, 2016.

[134] S. Skorobogatov, "Physical attacks on tamper resistance: progress and lessons," in *Proc. of 2nd ARO Special Workshop on Hardware Assurance, Washington, DC*, 2011.

[135] T. De Cnudde, M. Ender, and A. Moradi, "Hardware masking, revisited," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 2, pp. 123–148, 2018.

ECOLE
DOCTORALE

**Titre :** Évaluation pré-silicium des circuits sécurisés face aux attaques par canal auxiliaire

**Mots clés :** attaques par canaux auxiliaires; contremesures; évaluation pré-silicium.

**Résumé :** Les systèmes embarqués sont constamment menacés par diverses attaques, notamment les attaques side-channel. Pour garantir un certain niveau de sécurité, les implémentations cryptographiques doivent valider des tests d'évaluation recommandés par les standards de certifications, et ainsi répondre aux besoins du marché. Pour cette raison, il est nécessaire d'implémenter des contremesures fiables pour contrer ce type d'attaques. Néanmoins, une fois ces contremesures implémentées, les tests de vérification et de validation peuvent s'avérer très coûteux en temps et en argent. Ainsi, minimiser le nombre d'allers-retours, entre l'étape de conception et l'étape d'évaluation est primordial. Nous allons explorer une classe très large d'attaques existantes (passives et actives), et proposer des méthodes d'évaluations au niveau pré-silicium, permettant d'un côté, de détecter les différents types de fuites qu'un attaquant donné pourrait exploiter, et de l'autre, exposer des techniques de protection permettant de contrer ces attaques, tout en respectant l'aspect performance et taille en silicium. Nous nous basons dans nos analyses sur des méthodes formelles et empiriques, pour tracer l'impact de chaque vulnérabilité sur les différents niveaux d'abstraction du circuit, et ainsi proposer des contremesures optimales.

**Title :** Pre-silicon evaluation of secured circuit against side-channel attacks

**Keywords :** side-channel attacks; countermeasures; pre-silicon evaluation.

**Abstract :** Embedded systems are constantly threatened by various attacks, including side-channel attacks. To guarantee a certain level of security, cryptographic implementations must validate evaluation tests recommended by the certification standards, and thus meet the market needs. For this reason, it is necessary to implement reliable countermeasures to counter this type of attacks. However, once these countermeasures are implemented, verification and validation tests can be very costly in terms of time and money. Thus, optimizing the lifecycle of the circuit, between the design stage and the evaluation stage is paramount. We will explore a very broad class of existing attacks (passive and active), and propose methods of pre-silicon level assessments, allowing on the one hand, to detect the different types of leakages that a given attacker can exploit, and on the other hand, expose different techniques to counter these attacks, while respecting the performance and area aspect. In our analyses, we apply formal and empirical methods to track the impact of each vulnerability on the different abstraction levels of the circuit, and thus propose optimal countermeasures.

**Institut Polytechnique de Paris**
91120 Palaiseau, France