KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Cryptography Secured Against Side-Channel Attacks

Thomas De Cnudde

Supervisors:
Prof. dr. ir. Vincent Rijmen
Dr. Svetla Nikova

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

December 2018

Cryptography Secured Against Side-Channel Attacks

Thomas DE CNUDDE

Examination committee:
Prof. dr. ir. Pierre Verbaeten, chair
Prof. dr. ir. Vincent Rijmen, supervisor
Dr. Svetla Nikova, supervisor
Prof. dr. ir. Liesbet Van der Perre
Prof. dr. ir. Ingrid Verbauwhede
Prof. dr. ir. Joan Daemen
(Radboud University Nijmegen)
Priv.-Doz. dr. Amir Moradi
(Ruhr-Universität Bochum)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

December 2018

© 2018 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Thomas De Cnudde, Kasteelpark Arenberg 10, box 2452, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Acknowledgments

"Een Vlaming heeft altijd geluk..."

Flemish Proverb

The last 4 years have been positively transformational. There is a great number of people I wish to thank for this. Attempting a full enumeration would be futile, a small subset is given special acknowledgment below.

Dear Vincent, I can not thank you enough for the freedom you allowed me to pursue my research in my own millennial way. In addition to this thesis, it resulted in me gaining some of your values. You have shown me that a human being is infinitely more rich than a human doing.

Dear Svetla, thank you for the countless opportunities you gave me to grow as a person and a researcher. Your office door was always open for anything ranging from discussion on research directions to spontaneous (joyful or desperate) reports on lab experiments.

Dear Jury Members, thank you for attentively reading this thesis. Any positive aspect of the editorial quality of this thesis is entirely thanks to your comments.

Dear parents, thank you for your inexhaustible support and patience with me. Sofie, thank you for your unquestioning support and understanding, and Pieter, thank you for constantly expecting better of me.

Begül Bilgin, Oscar Reparaz, Benedikt Gierlichs, you have prepared me for paper submissions, conference talks, this thesis, and you have polished my critical thinking, communication and collaboration skills. Thank you for guiding my journey as a researcher.

Special thanks to the people that made the biweekly TI meetings, COSIC weekends, and everyday life in general unforgettable. Extra special thanks to

Péla 🌲, Machines, far-out office discussions, my 'maatjes' in Ghent and the Boston TSS team. COSIC has been a wonderful and unique place to carry out research, I hope to infuse some of its spirit in any of my future environments.

Pınar'ım, yürüdüğümüz yollar sana dolanmış kollarım gibi birbirine dolansın.

Finally, I want to thank the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) for their generous support, and STUKcafé, NMBS trains (not the new ones, they don't have tables) and Caffè Nero Boston for being excellent writing locations.

Thomas De Cnudde
December 2018

Abstract

Embedded devices like wireless sensor networks and smart cards have become ubiquitous in everyday life. Their widespread deployment have placed them in a prominent position in the emerging Internet of Things, and for sectors as diverse as health care, entertainment, financial, industry and automotive. The foundational role fulfilled by embedded systems has made them an attractive target for malicious exploits. Especially in devices for intelligent homes, self driving cars and even electronic body implants, the risks are obvious and hence the necessity for cryptography has become well-accepted. In the Black-Box model, where an attacker only has access to some inputs and/or outputs, and potentially a description of the cipher, standardized cryptographic block ciphers provide strong security. This is not the case in the Grey-Box model, where sensitive key information can be derived from physical parameters leaked by the implementation of the cipher on a device.

The Grey-Box model can be subdivided in two types of physical attacks: passive attacks and active attacks. The former attempts to retrieve sensitive information by passively observing data or operation dependent variations in the timing, power consumption, electromagnetic radiation of the device or other so-called side channels. The latter attempts to reveal secrets by actively injecting faults in the circuitry. These attacks can further be subdivided in the cheaper non-invasive attacks and the more expensive invasive ones. A powerful passive, non-invasive attack is differential power analysis (DPA) and can successfully retrieve secret keys from unprotected implementations with a high success rate at a low cost. Its dual attack in the active category is differential fault analysis (DFA) and requires a more knowledgeable attacker to achieve success. Combined attacks are even more advanced and mix both active and passive methods. The easiest attack point in a device requires the highest priority to mitigate, and various countermeasures have been proposed for both passive and active types of attacks.

The threshold implementations (TI) masking scheme is a countermeasure against

side-channel analysis (SCA). It provides provable security against DPA in the presence of hardware glitches given the assumption that the total leakage of the device is a linear combination of leakages from the different shares and shared functions. This minimal assumption on the hardware results in TI achieving a lower circuit complexity and higher throughput than countermeasures with equal security. TI and other TI-based countermeasures rely on four properties for its SCA security, namely correctness, non-completeness, uniformity of the shared inputs and uniformity of the shared functions. Thwarting fault attacks often relies on either spatial or temporal redundancy of the cryptographic algorithm resulting in either a higher circuit complexity or a longer execution time. Alternatively, redundancy can be added to the intermediate variables through error correcting or detecting codes, which result in an increase of the area as well.

In the constrained environments in which embedded cryptosystems are deployed, the overheads induced by Grey-Box countermeasures should be minimized. This dissertation provides contributions to the field of embedded security with small overheads in three ways. For our first contribution, we relate different d^{th}-order secure hardware masking schemes, namely Boolean, TI-like masking, Inner-Product masking and Polynomial masking which use a minimal number of shares $d + 1$. We extract a common structure shared by the masking schemes and we examine circuit complexity-randomness-security trade-offs. We proceed by taking the most compact scheme (Boolean, TI-like masking) to implement a first- and second-order DPA secure advanced encryption standard (AES) and perform an SCA evaluation on a Field Programmable Gate Array (FPGA) platform. While the theory is sound, we come across an unexpected problem: the evaluations are not always free of leakage. This leads us to the investigation of the common assumptions of masking schemes and forms our second contribution. We show that the linear leakage assumption is violated in the presence of coupling. To this end, we use the very strong Test Vector Leakage Assessment (TVLA) evaluation methodology. In our third contribution, we evaluate TI in the context of active fault attacks. We start by assessing its resistance against clock and supply voltage glitching. We finally describe an implementation of a combined countermeasure (Private Circuits II) that leverages TI to achieve a lower circuit complexity and a lighter randomness requirement. The result is an implementation that can help benchmark future combined countermeasure implementations.

Beknopte samenvatting

Ingebedde systemen zoals draadloze sensornetwerken en smartcards zijn alomtegenwoordig in het dagelijks leven. Door hun wijde verspreiding hebben ze een prominente positie ingenomen in het Internet of Things en dat voor uiteenlopende sectoren zoals gezondheidszorg, entertainment, financiën, industrie en automotive. Door hun fundamentele rol vormen ze een aantrekkelijk doelwit voor aanvallen. Vooral in apparaten voor zogenaamde *smart* huizen, zelfrijdende auto's en zelfs elektronische lichaamsimplantaten zijn de gevolgen van slecht bedoelde aanvallen duidelijk. Het is daarom noodzakelijk om cryptografie aan ingebedde systemen toe te voegen. In het zogenaamde *Black-Box* model, waar een aanvaller enkel toegang heeft tot een selectie van ingangen en/of uitgangen, en mogelijk ook een beschrijving van het cryptografisch cijfer heeft, bieden gestandaardiseerde cryptografische blokcijfers voldoende beveiliging. Dit is niet het geval in het *Grey-Box* model, waar informatie van de sleutel kan worden afgeleid uit fysieke parameters die de implementatie van het cijfer op een apparaat lekt.

Het *Grey-Box* model kan worden onderverdeeld in twee soorten fysieke aanvallen: passieve aanvallen en actieve aanvallen. Het eerste type aanval probeert gevoelige informatie te verkrijgen door de implementatie passief te observeren. De tijd van een operatie, zijn vermogenverbruik, zijn elektromagnetische straling of andere zogenaamde nevenkanalen kunnen allemaal afhangen van de geheime sleutel. Het tweede type aanval probeert geheimen te onthullen door actief fouten te injecteren in de implementatie. Deze aanvallen kunnen verder worden onderverdeeld in de goedkopere niet-invasieve aanvallen en de duurdere invasieve aanvallen. Een krachtige passieve, niet-invasieve aanval is differentiële vermogensanalyse (DPA) en kan met succes geheime sleutels van onbeschermde implementaties onthullen aan een lage kost en met een hoog slaagpercentage. De duale aanval in de actieve categorie is differentiële foutanalyse (DFA) en vereist een meer ervaren aanvaller. Gecombineerde aanvallen zijn nog geavanceerder en combineren zowel actieve als passieve methoden. Het eenvoudigste aanvalspunt in een apparaat vereist de hoogste prioriteit om te beveiligen en er zijn

verschillende gekende tegenmaatregelen voor zowel passieve als actieve aanvallen.

De Threshold Implementatie (TI) is een tegenmaatregel voor nevenkanaal-analyse (SCA) gebaseerd op *masking*. Het biedt bewijsbare veiligheid tegen DPA in de aanwezigheid van hardware glitches, op voorwaarde dat de totale gelekte informatie van het apparaat een lineaire combinatie vormt van de gelekte informatie van de verschillende gedeelde variabelen en gedeelde functies. Deze minimale assumptie op de hardware heeft tot gevolg dat TI een lagere complexiteit heeft en een hogere doorvoer bereikt dan tegenmaatregelen met vergelijkbare beveiliging. TI en andere TI-gebaseerde tegenmaatregelen vereisen de voldoening van vier eigenschappen, zijnde correctheid, niet-compleetheid, uniformiteit van de gedeelde variabelen en uniformiteit van de gedeelde functies. De bescherming tegen foutaanvallen berust vaak op ofwel ruimtelijke of temporele redundantie, resulterend in ofwel een toename in complexiteit of een toename in uitvoeringstijd. Als alternatief kan redundantie aan de tussenvariabelen worden toegevoegd door middel van foutcorrectie of detectiecodes, wat ook resulteert in een toename van de ingenomen oppervlakte.

De extra kosten van tegenmaatregelen in het *Grey-Box* model zijn bij voorkeur minimaal. Dit proefschrift levert op drie manieren bijdrage aan de beveiliging van ingebedde systemen met minimale extra kosten. Als eerste bijdrage hebben we verschillende d^{de}-order masking schema's voor hardware vergeleken, namelijk TI-gebaseerde Booleaanse masking, Inwendig-Product masking en Polynomiale masking. Allen maken gebruikt van het minimale aantal gedeelde variabelen $d + 1$. We extraheren een gemeenschappelijke structuur die wordt gedeeld door de masking schema's en we onderzoeken de verschillende trade-offs tussen de complexiteit en de verbruikte random getallen. We gebruiken vervolgens het meest compacte schema (TI-gebaseerde Booleaanse masking) om een eerste en tweede order DPA-beveiligde AES te implementeren. We voeren een SCA-evaluatie uit met behulp van een FPGA platform. Ondanks de sluitende theorie, komen we een onverwacht probleem tegen: de evaluaties zijn niet altijd vrij van lekken. Dit leidt ons tot het onderzoeken van de algemene, onderliggende assumptie van de masking schema's, wat onze tweede bijdrage vormt. We laten zien dat de lineaire lek-assumptie wordt geschonden in de aanwezigheid van koppelingen. Hiervoor gebruiken we de zeer sterke *Test Vector Leakage Assessment* (TVLA) evaluatiemethodologie. In onze derde bijdrage evalueren we TI in de context van actieve foutaanvallen. We beginnen met het onderzoeken van zijn weerstand tegen klok- en voedingsspanning glitches. We beschrijven tenslotte een implementatie van een gecombineerde tegenmaatregel (Private Circuits II) die gebruikmaakt van TI om een lagere complexiteit en een kleiner aantal verbruikte random getallen te verkrijgen. Het resultaat is een implementatie die kan helpen bij het vergelijken van toekomstige implementaties van gecombineerde tegenmaatregelen.

List of Abbreviations

AES Advanced Encryption Standard.

ASIC Application-Specific Integrated Circuit.

CMS Consolidated Masking Scheme.

DFA Differential Fault Analysis.

DOM Domain-Oriented Masking.

DPA Differential Power Analysis.

EM Electromagnetic.

FA Fault Attack.

FPGA Field-Programmable Gate Array.

FS Fault Sensitivity.

FSA Fault Sensitivity Analysis.

GE Gate Equivalent.

HD Hamming Distance.

HO-DPA Higher-Order Differential Power Analysis.

HW Hamming Weight.

IC Integrated Circuit.

IP Inner-Product.

LUT Look-Up Table (LUT).

MPC Multi-Party Computation.

OFB Output Feedback.

PC-II Private Circuits II.

PRNG Pseudorandom Number Generator.

SCA Side-Channel Analysis.

SNI Strong Non-Interference.

SNR Signal-to-Noise Ratio.

SPA Simple Power Analysis.

TI Threshold Implementation.

TVLA Test Vector Leakage Assessment.

Contents

Abstract iii

Beknopte samenvatting v

List of Abbreviations viii

Contents ix

List of Figures xv

List of Tables xix

1 Introduction 1

1.1 Situation and Motivation for our Research 4

1.1.1 Adversary Models 4

1.1.2 Categorizing Physical Attacks 6

1.1.3 Categorizing Countermeasures Against Physical Attacks 8

1.2 Research Questions 11

1.3 Roadmap of the Thesis 13

2 Physical Attacks and Countermeasures 17

2.1 Notation and Definitions 18

2.2 Passive Physical Attacks and Countermeasures 18
2.2.1 Side-Channel Analysis Attacks 19
2.2.2 Side-Channel Analysis Countermeasures 21
2.3 Active Physical Attacks and Countermeasures 23
2.3.1 Fault Attacks 24
2.3.2 Fault Attack Countermeasures 26
2.4 Security Validation 26
2.4.1 Side-Channel Measurement Setup 27
2.4.2 t-Test Based Leakage Detection 29
2.5 Conclusion 30

3 State of the Art Hardware Masking Schemes 31
3.1 Security Models 32
3.2 Boolean, TI-Like Masking 34
3.2.1 Initial Sharing 36
3.2.2 Masked Multiplication 36
3.2.3 A Second-Order Secure Example 37
3.3 Inner-Product Masking 38
3.3.1 Initial Sharing 38
3.3.2 Masked Multiplication 38
3.3.3 A Second-Order Secure Example 39
3.4 Polynomial Masking 39
3.4.1 Initial Sharing 40
3.4.2 Masked Multiplication 41
3.4.3 A Second-Order Secure Example 42
3.5 Extracting A Generalized Structure 43
3.6 Security Proofs 45

3.7 Variations for Trade-Offs, Offset Remasking and Non-Complete Compression . . . 47
3.8 Conclusion . . . 51

4 Securing AES with Boolean, TI-Like Masking 53
4.1 An Unprotected Implementation of AES . . . 54
4.1.1 A Very Compact Hardware Implementation . . . 54
4.1.2 Canright's Very Compact AES S-box . . . 55
4.2 Masking AES at Different Orders . . . 56
4.2.1 Linear Operations . . . 57
4.2.2 Redefining the S-box Decomposition . . . 57
4.2.3 Implementation 1: First-Order TI of the AES S-box with $d+1=2$ Shares . . . 58
4.2.4 Implementation 2: Second-Order TI of the AES S-box with $d+1=3$ Shares . . . 61
4.2.5 Implementation 3: Second-Order TI of the AES S-box with $6 > td+1$ Shares . . . 63
4.3 Leakage Detection . . . 63
4.3.1 Implementation 1: (2,4)-Sharing . . . 64
4.3.2 Implementation 2: (3,9)-Sharing . . . 64
4.3.3 Implementation 3: (6,6)-Sharing . . . 64
4.4 Implementation Cost . . . 65
4.5 Conclusion . . . 67

5 Evidence of Leakage from Coupling in Masked Implementations 73
5.1 Sources of Out-of-Model Leakage . . . 74
5.1.1 Failure of Independent Leakage . . . 74
5.1.2 Power Consumption in Masking Schemes . . . 75
5.1.3 Crosstalk . . . 76

5.1.4 IR Drop 77
5.2 Coupling in Threshold Implementations 77
5.2.1 Crosstalk 78
5.2.2 IR Drop 79
5.3 KATAN-32 and Its Threshold Implementation 80
5.4 Coupling in a TI of KATAN-32 with 3 Shares 81
5.4.1 Secure Threshold Implementation of KATAN-32 82
5.4.2 Leaking Threshold Implementation of KATAN-32 84
5.5 Discussion 85
5.6 Conclusion 87

6 Glitch-Resistant Masking Schemes Prevent FSA 89
6.1 Fault Sensitivity Analysis 90
6.2 Assumptions on the Masked Implementation and FSA Attack . 91
6.3 Glitch Resistance and Fault Sensitivity 93
6.3.1 Extending the Relation Between FSA and Power Analysis 94
6.3.2 Threshold Implementations Resist FSA 94
6.3.3 The Roche and Prouff Masking Scheme Resists FSA . . 95
6.4 Conclusion 97

7 Protecting PRESENT Against Combined SCA & Arbitrary Fault Injections 99
7.1 Background 101
7.1.1 PRESENT Block Cipher 101
7.1.2 Private Circuits or ISW 101
7.1.3 Private Circuits II 103
7.2 The Masking Process 104

7.2.1 ISW vs. TI 106
7.2.2 Masking PRESENT with Threshold Implementations . 107
7.2.3 Testing PRESENT-TI with Leakage Detection Tests . . 108
7.3 Applying PC-II 110
7.3.1 Encoding 110
7.3.2 Gates Encoding 111
7.3.3 Error Cascading 112
7.3.4 Leakage Detection 113
7.3.5 Circuit Complexity 113
7.4 Resistance Against Fault Attacks 115
7.4.1 Fault Attack Simulation 116
7.4.2 Increased Resistance Against Differential Fault Analysis 118
7.5 Conclusion 119

8 Conclusions & Open Problems **121**
8.1 Summary 121
8.2 Directions for Future Work 123

Bibliography **127**

List of Publications **141**

List of Figures

1.1 In the Black-Box Adversary model, the attacker can choose plaintexts or ciphertexts freely to try to mathematically break a (known) algorithm 5

1.2 An (impossible) look-up table implementation that holds 2^{128} plaintext-ciphertext entries encrypted under a given key is secure in the White-Box Adversary model 5

1.3 In the Grey-Box Adversary model attackers can observe side channels through which key information leaks, they can additionally induce erratic but exploitable behavior 7

2.1 ISW with an imposed execution order from the registers 22

3.1 Multiplication with Boolean, TI-like Masking 43

3.2 Multiplication with IP Masking 43

3.3 Multiplication with Polynomial Masking 44

4.1 State and Key Array 56

4.2 Operations in the Canright AES Sbox. The lines depict pipeline registers to keep the non-completeness in masked scenarios . . 56

4.3 Additive remasking for the first-order implementation (left), ring remasking for the second-order implementations (right) 62

4.4 Structure of the second-order TI of the AES S-box 69

4.5 Masked AES with (2,4)-sharing, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 5k traces, first-order t-test with uniform masks using 50M traces, second-order t-test with uniform masks using 50M traces 70

4.6 Masked AES with (3,9)-sharing, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 5k traces, first-order t-test with uniform masks using 50M traces, second-order t-test with uniform masks using 50M traces, third-order t-test with uniform masks using 50M traces 71

4.7 Masked AES with (6,6)-sharing, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 5M traces, first-order t-test with uniform masks using 50M traces, second-order t-test with uniform masks using 50M traces, third-order t-test with uniform masks using 50M traces 72

5.1 Crosstalk between two wires w_1 and w_2 originates from the inter-wire capacitance $C_{1,2}$ 77

5.2 Static and dynamic IR drop occurs from the non-zero resistance of conductive supply voltage and ground wires 77

5.3 Power supply noise or IR drop in the PDN couples shares . . . 80

5.4 KATAN-32 consists of two sets of shift registers and four groups of nonlinear operations (Source: [21]) 81

5.5 Placing the individual shares far apart leads to a secure design 82

5.6 Placing all shares in close proximity leads to a design that leaks 83

5.7 Leakage detection tests of a secure KATAN-32 TI, 20k traces masks off (top), 100M traces masks on 1st-order (middle), 100M traces masks on 2nd-order (bottom) 84

5.8 Leakage detection test of an insecure KATAN-32 TI, 20k traces masks off (top), 100M traces masks on 1st-order (middle), 100M traces masks on 2nd-order (bottom) 85

5.9 Evolution of the points of maximum absolute values of the leakage with increasing number of traces for the secure and insecure KATAN-32 TI and plaintext value 00000000_{hex} 86

5.10 Evolution of the points of maximum absolute values of the leakage with increasing number of traces for the secure and insecure KATAN-32 TI and plaintext value $087D2EC1_{hex}$ 86

6.1 When the leakage requirement holds, the total power consumption trace can be decomposed into power traces of the different shared sub-circuits 93

6.2 General structure of the threshold implementation example . . 96

6.3 General structure of the Rivain and Prouff masked multiplier . 97

7.1 In this chapter we search for an alternative, more effective way to achieve a Private Circuits-II implementation of PRESENT . 100

7.2 The error cascading stage propagates any e-bit fault to all wires before values are registered (illustrated here for $e = 1$) 104

7.3 Masked PRESENT-TI, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 20k traces, first-order t-test with uniform masks using 100M traces, second-order t-test with uniform masks using 100M traces 109

7.4 Output logic for PRESENT-TI 110

7.5 Control structure for PRESENT-TI 111

7.6 Logic structure of a multiplexer 111

7.7 Masked PC-II protected PRESENT, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 20k traces, first-order t-test with uniform masks using 100M traces, second-order t-test with uniform masks using 100M traces 114

7.8 Traces of signals from the PRESENT-TI implementation with a set fault on the `ready` signal 116

7.9 Traces of signals from the PC-II protected PRESENT implementation with a set fault on one of the wires of the encoded `ready` signal 117

7.10 Traces of signals from the PRESENT-TI implementation with a set fault on the first share of the S-box input 117

7.11 Traces of signals from the PC-II protected PRESENT implementation with a set fault on one of the wires of the encoded first share of the S-box input 117

List of Tables

4.1 Circuit complexity of different functions of the masked AES . . 65

4.2 Implementation cost of different TIs of AES 67

7.1 Truth tables for the gadgets, where d is the order of SCA security and e is the number of tolerable faults 105

7.2 4-Bit to 4-Bit Substitution of the PRESENT S-Box [25] and a quadratic decomposition $F(G(x))$ [110] 108

7.3 Circuit Complexity of Different Functions of the PRESENT Designs 115

7.4 Circuit Complexity of the Different PC-II Gadgets Used 115

7.5 Resource comparison of the different first-order SCA resistant PRESENT versions on the Spartan-6 FPGA 115

Chapter 1

Introduction

Spread around the globe is a billion-dollar network of sensors with the critical purpose of detecting and pinpointing nuclear bomb tests anywhere in the world. This International Monitoring System (IMS) is used by the members of the Comprehensive Nuclear-Test-Ban Treaty (CTBT) of 1996 to keep nuclear mayhem at bay. Since nuclear inspectors are not allowed to enter all countries, this network fills an important gap in controlling illicit nuclear activity. Over a course of 24 hours, this network records 26 gigabytes of data from IMS stations in 89 countries. Once recorded, this data is sent to Vienna over satellite networks and secure ground links for further processing. Needless to say, security is a cornerstone for a network that fulfills such an important task.

We need to think of security on two levels here: the security of the communication channels on one hand, and the security of the "base stations" on the other. Cryptography plays a quintessential role in both aspects and makes it harder for a malicious country to initiate nuclear winter by providing following important properties. For a full list, we refer the reader to the Handbook of Applied Cryptography [94].

Confidentiality. Confidentiality (or privacy) is the notion perhaps most strongly associated with cryptography and security. Its purpose is to only allow information to be revealed to intended communicating parties, while restricting unauthorized parties from accessing that information.

Data Integrity. Data Integrity guards information from being manipulated or altered by unauthorized parties. It assures the accuracy and consistency of

communicated, processed or stored data.

Authentication. Authentication can apply to entities or messages. Entity authentication (or Identification) confirms the identity of a person or an object (e.g. a credit card) by validating certain attributes they posses. In case of message authentication, it validates the origin of the information.

Established cryptographic algorithms make sure these attacks are, when used properly, virtually impossible. Three branches of cryptography are responsible for various aspects of satisfying those properties: Symmetric-Key Cryptography, Public-Key Cryptography and Hash Functions.

Symmetric-Key Cryptography. Confidentiality is most efficiently achieved through Symmetric-Key cryptographic algorithms like the Advanced Encryption Standard (AES) [40]. The bulk of information transmitted between senders/receivers will be encrypted/decrypted using the same (hence symmetric) key; a vital question then becomes how the parties securely agree on a key. This class of cryptography encompasses Block Ciphers, Stream Ciphers and the more recent Authenticated Encryption algorithms.

Public-Key Cryptography. Public-Key Cryptography (or Asymmetric-Key Crypto) uses different keys for encryption and decryption, hence the asymmetry. The Public Key is available to anyone and is used for encryption, whereas the Private Key is known exclusively by the receiving party to decrypt a message. Public-Key Cryptography enables a.o. authentication (e.g. digital signatures) and key exchange and thus simplifies deploying Symmetric-Key Cryptography.

Hash Functions. Hash Functions take an input message of variable length and return a fixed-size message. They do not require a key. An essential property of Hash Functions is that they are irreversible, i.e. once "encrypted" they can not be decrypted. Applying a Hash Function on a message creates a so-called digital fingerprint that can be sent alongside the message. On the receiving side, the message can be hashed and compared to the digital fingerprint to check for any alteration. Hash Functions can thus be used to test data integrity.

Once cryptography is properly implemented to secure the communication links between IMS base stations, one might think a malicious country has no chance at reading and altering data, or impersonating other countries anymore. Indeed, the recommended AES-128 algorithm would require to search a key in a haystack

of 2^{128} possibilities[1], which is infeasible even with our most modern computing resources.

There is however a way to alleviate this search if our malicious country has access to a base station: it can try to break into the base station to retrieve its keys. So in addition to providing strong security to the communication between base stations, it is crucial to make them so-called tamper-resistant. The vital purpose this network fulfills justifies a high cost that can be allocated to achieve this physical property. Except for accuracy, availability and high security, there are little other constraints that need to be considered during their design. This (rather extreme) example illustrates the importance of tamper resistance when cryptographic systems operate in hostile environments.

A less extreme example, at least on the surface, is found by shifting our perspective to a different notion of hostility and a different setting: the often mentioned (Industrial) Internet of Things. The omnipresence and steady rise of embedded devices in our society is impossible to ignore: smart devices are everywhere, e.g. smart assistants (including Apple HomePod, Amazon Alexa, Google Home), smart home appliances (e.g. the Philips Hue light bulbs), smartcards like London's Oyster card or credit cars, pacemakers, smart pill boxes, fitness trackers, ... They are used for a wide range of applications of which medical, military, financial, identification, smart metering, tracking of products and even people are only a small subset. While part of the functionality of these devices is related to security (e.g. in cases of access control or finance), it is not always clear for a designer to what extent crypto is needed. To make matters worse, the creativity of an attacker (and even its existence) is easily overlooked. In contrast to our previous example, there are significantly heavier constraints on embedded devices regarding efficient power consumption, compact size and high speed, which leads designers to often trade off on crypto first, even where it is essential. It comes as no surprise that for a large part of the embedded IoT devices a quick Google search reveals a security breach in one way or another.

A specific case study reveals just how detrimental badly deployed crypto can be. Ronen et al. [122] developed a worm that can rapidly infect a network of adjacent, unconnected Philips Hue smart lamps by leveraging built-in functionality only. Their attack allows to turn all smart lights in a city on or off, to brick them permanently or even to recruit them for a massive Distributed Denial-of-Service Attack (DDOS). Their attack exploits a major bug in the implementation of a protocol and unintentional information leakage in the power consumption to extract the secret key. The knowledge of the key then enabled over-the-air malicious firmware updates to load the worm on devices. All this was performed using low-cost off-the-shelf equipment and provides an example that even large

[1] if this number looks small, try entering it in a calculator

companies have difficulties getting crypto right in implementations. When we consider that pacemakers are part of the IoT, this indeed paints a rather grim picture.

Both examples highlight the importance of hardening cryptosystems over which an adversary has control. To avoid that the law of exponential increase in IoT devices and their exponentially growing market share carries over into a law of exponential security breaches, serious research attention to these problems' solutions is needed. As constraints vary widely for different types of devices and settings, this topic is not a trivial matter. This work is situated at an intersection of these difficulties, and fortunately, there is a solid base of research to start our study from.

1.1 Situation and Motivation for our Research

The examples given in our introduction highlight just how crucial it is to prevent (physical) attacks on cryptosystems. We have shown that the research of countermeasures in the form of tamper-resistance is valuable and that we need to design these given a wide variety of often contradicting constraints. As we play the role of a defender trying to oppose an attacker, it is crucial to know what type of adversaries we are dealing with. In this section, we first give an overview of different adversary models and describe broadly which model we employ. We additionally situate our work in the state-of-the-art of this field of research. We present the global related work which we gradually narrow down in scope to prime the reader before we state our research questions.

1.1.1 Adversary Models

Three well known and broad adversary models are the Black-Box model, the White-Box model and the Grey-Box model. This classification is based on the amount of information that is available to the adversary.

Black-Box Model. The first adversary model assumes that the attacker has access to the inputs and outputs of the cryptosystem only (Figure 1.1). This model is used to study implementation-independent characteristics of a cryptographic algorithm, like how good they resist cryptanalytic or mathematical attacks. Good algorithms have emerged from the interplay between designers and attackers (i.e. cryptanalysts) and one such example is the Advanced Encryption Standard [40]. Ideally, the only option an attacker has against such algorithms

is an exhaustive search of all keys known as a brute force attack. If the key space is large enough, this search will be impossible. While the symmetric-key encryption algorithms we consider are deemed unbreakable to modern standards, cryptanalysis is a still evolving research area, but out of the scope of this work.

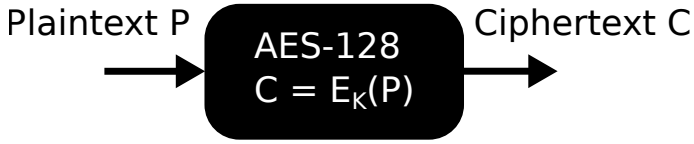

Figure 1.1: In the Black-Box Adversary model, the attacker can choose plaintexts or ciphertexts freely to try to mathematically break a (known) algorithm

White-Box Model. This second adversary model assumes that the attacker has access to all information of the implementation. Protecting against such an adversary requires an obfuscated implementation. Naively, one could achieve this by hardcoding the AES key in the software code. Upon release of the code however, an attacker can very easily retrieve the key. An implementation secure in this model would look as follows. A key is chosen and a table is made that stores all ciphertexts next to their corresponding plaintexts. Given the block size of the algorithm[2] it is of course impossible to store such a look-up table in memory (Figure 1.2). The difficulty that follows from this very strong adversary model leads to designs that are often insecure [28]. Increasing research attention is directed towards this problem.

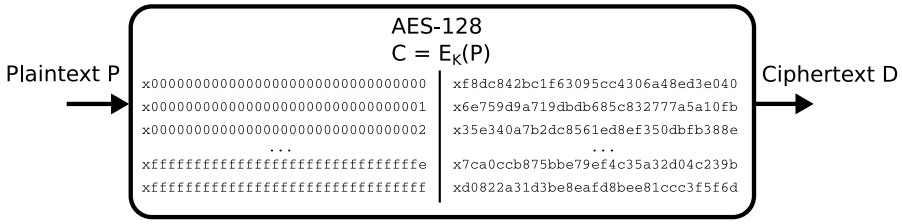

Figure 1.2: An (impossible) look-up table implementation that holds 2^{128} plaintext-ciphertext entries encrypted under a given key is secure in the White-Box Adversary model

[2] there would be 2^{128} table entries for AES

Grey-Box Model. The third adversary model, and the one we consider in this work, is the Grey-Box model. It lies in between the two extremes spanned by the Black-Box and White-Box models. In this model, an attacker has access to the platform the cryptographic algorithm is implemented on. This model is inherently tied to the physical world a device resides in (Figure 1.3).

The first set of attacks in this model were revealed in the late nineties by Kocher et al. [83]. By pointing out that the timing of operations is dependent on the secret values, many implementations were shown to be exploitable. Alternatively, the power consumption or the electromagnetic radiation of a device during encryption was shown to depend on intermediate secret values [84]. By measuring and analyzing these physical parameters, the secret key material can be extracted with surprising ease, at a reasonably low cost and with basic electronic equipment. Creative new attacks are still being uncovered and include exploiting information from sound, heat or light [59, 60, 75]. These unintended channels that leak information on key material are called side channels. It is important to reduce the information leaked through them when deploying cryptosystems in hostile environments (including consumers' hands). In early 2018, the Spectre [82] and Meltdown [88] attacks targeting Intel processors gained widespread media coverage and further stresses the importance and realism of the Grey-Box model.

As the attacker has access to the implementation of the cryptographic algorithm, there is no limitation in how he uses the device. The attacker can even misuse the cryptosystem in hope to reveal secret information [26]. By hovering electromagnetic probes (i.e. antennas) over the surface of a decapsulated chip's memory, stored secret keys could be read out directly. Alternatively, putting the device under stress can possibly trigger exploitable behavior, e.g. by heating it or by inducing errors in intermediate results using lasers.

For cryptographic algorithms to be useful, they will at some point need to be implemented. As a result these physical attacks are a real concern and their mitigation is critical for the security of embedded devices.

1.1.2 Categorizing Physical Attacks

We need to know what an attacker is capable of in order to design a device that is effectively secure against physical attacks. To this end, the Grey-Box Adversary model is often refined and subdivided in the following taxonomy of attackers [4].

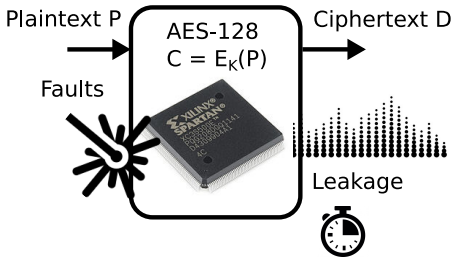

Figure 1.3: In the Grey-Box Adversary model attackers can observe side channels through which key information leaks, they can additionally induce erratic but exploitable behavior

Class I (clever outsiders). Attackers in Class I have some knowledge about the subject and generally know what to look for, but have insufficient knowledge of the details of the system. Their equipment is at most moderately sophisticated and they more than often rely on existing weaknesses rather than the creation of new ones.

Class II (knowledgeable insiders). Attackers in Class II have far more specialized technical education and experience than an attacker in Class I. Their understanding of parts of the system can be high and they can potentially even have access to the system. They have highly sophisticated equipment for analysis at their disposal.

Class III (funded organizations). Attackers in Class III are extremely well funded and can assemble teams of experts with complementary skills. They can perform in-depth system analysis and launch sophisticated custom attacks. They have access to the most advanced equipment.

The division of adversaries in classes and the physical nature of the attacks hints at a wide range of existing attacks that can be mounted, and leads to further subdivision. An accepted way to partition the attacks is based on the interactivity of the attacker (either passive or active) and the level of intrusion in the system by the attacker (invasive versus non-invasive).

Passive and Active Attacks. The first subdivision is made according to how the attacker interacts with the device in the Grey-Box model. Passive attacks are considered when the device is operational as expected and the attacker is only observant of the device. Attacks in this category are called Side-Channel

Analysis (SCA) attacks. Active attacks on the other hand consider an attacker that manipulates and misuses the device with the goal to induce some unexpected or faulty behavior. An attack in this category is referred to as a Fault Attack (FA). Both active and passive attacks can be combined to form even more powerful attacks. Such attacks are referred to as Combined Attacks.

Non-Invasive and Invasive Attacks. The second subdivision is based on the level of intrusion of the attacker in the system and is orthogonal to the previous classifications. Non-invasive attacks require no alteration of the system. An attacker will only tamper with the interface of the device and/or its environment. Invasive attacks are more expensive and time consuming to prepare but have the advantage that they are subject to less limitations: an attacker gains access to the inner level of the circuit, which can be very powerful. Semi-invasive attacks lie in between the previous two extremes. They involve a certain level of intrusion that typically stops at the passivation layer of the circuit. Decapsulation of the chip falls in this category.

The advantage of the previous classification of attacks is that they provide an intuition on the cost and effort of the attack. The higher the level of intrusion, the higher the resulting cost of the equipment and the higher the expertise and time investment required to successfully mount an attack. Invasive attacks can e.g. involve Scanning Electron Microscopes (SEM) or Focused Ion Beams (FIB), which are extremely expensive to acquire and require expertise to operate. It is conceivable that such attacks will be less of a priority to protect against than attacks that are easier and cheaper to mount. In other words, the weakest security link in the systems will need to be strengthened first.

1.1.3 Categorizing Countermeasures Against Physical Attacks

The multitude of attacks underlines the importance of securing cryptosystems in the Grey-Box model. The degree of resistance against physical attacks can thus be considered an extra design parameter, along with the classic dimensions of the embedded design space: throughput, circuit complexity and energy consumption.

Different countermeasures can be classified along the resistance they bring, the overhead they introduce, or the level on which they are deployed. We use the latter for our overview.

High-Level Countermeasures. Higher-level, or system-level countermeasures take advantage of mathematical constructs and bounds on information leakage. Re-keying limits the number of iterations an algorithm can make under the same key, with as drawback that establishing a new key is expensive. Leakage Resilient Cryptography alleviates this cost by generating the key algorithmically [53]. The advantage of high-level countermeasures is that they can be proven formally. Their disadvantage are the many assumptions on the behavior of the underlying layers that need to be satisfied.

Low-Level Countermeasures. Low-level solutions have the advantage that they consider the root of the problem: the physical leakage of the device. Once a device is secured at this level, it is unlikely that leakage will trickle down from higher levels. In practice however, they lead to large area overheads, or require tremendous expertise and caution to get right.

Mid-Level Countermeasures. In between the high level and low level are countermeasures that alleviate the disadvantages of these two extremes. Mid-level countermeasures make an abstraction of the underlying levels by making assumptions on the leakage behavior of the lowest levels (e.g. transistors). More generally applicable countermeasures can then be derived. This way, less expertise and time is required to implement the countermeasures correctly. Another benefit is that they can be applied on existing platforms that do not have the low-level countermeasures in place, e.g. microprocessors or CPUs.

In low- and mid-level countermeasures, a designer has several choices depending on the underlying platform that needs to be secured. Firstly, the designer can choose between hiding countermeasures and masking countermeasures to protect against a passive attacker.

In hiding countermeasures, a designer can aim to make the power consumption of logic operations equal for every possible input value (a low-level countermeasure). Wave Dynamic Differential Logic (WDDL) IC cells [135] are one example and have the advantage that they can be built from standard CMOS logic. Still, applying such countermeasures properly requires a significant investment in time and expertise. A designer can alternatively choose mid-level countermeasures to secure the design: randomizing the execution order of operations, destabilizing the clock or introducing dummy operations are good examples, but have been shown to not be sufficient [90].

In masking countermeasures, secret sharing with randomized data is used to break the direct relation between sensitive data and the side channel. A secret variable can be split in two parts (or shares) such that they individually appear

random and such that both are needed to reconstruct the secret. When applied correctly, masking with two shares mitigates Differential Power Analysis (DPA) attacks [84]. A more advanced attack that can break a masked implementation with two shares is Higher-Order Differential Power Analysis (HO-DPA) [35, 95]. It comes in two variations: univariate HO-DPA (a single point in the power trace is exploited) and multivariate HO-DPA (multiple points of the trace are exploited). A scheme that shares the secret over more than two shares when masking linear operations is called a higher-order masking scheme and can mitigate HO-DPA. The number of shares is related to the masking order. Given a set level of noise, the more shares a secret variable is split in, the harder it is for an attacker to perform a SCA attack. Masking has been of increasing interest to the research community and the industry. A developer has the choice to implement the cryptosystems in either software or hardware. Software, at least traditionally, is inherently sequential. As a result, it becomes increasingly more expensive in terms of both timing and code size to implement higher-order masking schemes in software [69]. We therefore focus on masked hardware implementations, and how we can reduce their implementation cost. A major advantage is that provable security can be achieved in a certain leakage model and under certain assumptions. These assumptions are however not always satisfied, and their investigation is a topic of our study as well. A well-known example of a violated assumption is that logic gates execute only once and in a certain sequence. The ISW masking scheme [77] (named after its authors Ishai, Sahai and Wagner) is a masking scheme that relies on this assumption for its SCA security. When this scheme is implemented in CMOS however, this assumption does not necessarily hold and the implementation can become insecure. The more recent Threshold Implementation (TI) masking scheme [21, 105–107] was designed with this issue in mind, and relaxes the assumptions that need to hold for the scheme to be secure.

Countermeasures against fault attacks are often researched in isolation from masking countermeasures. Well-known ways to prevent FAs is through the addition of redundancy in space or time (e.g. computing the encryption twice and checking whether or not the output is the same), through the addition of shields and sensors to detect anomalies, or through mathematical constructs like error detection/correction codes. If a fault is detected, the system starts a procedure that can range from the permanent destruction of the device to the device releasing garbage or zeroed output values. The latter response is an effective way to counter Differential Fault Analysis (DFA) attacks [20], which require the faulty output for a successful key retrieval. Applying both masking and FA countermeasures orthogonally can lead to an unreasonably high overhead on the implementation cost. We therefore investigate how both can be combined to reduce the cost of countering combined attacks.

Note that no perfect solution that counters all physical attacks exists. In practice, a cryptosystem can be considered secure when the resources needed for breaking it outweigh the losses to the company or end-users entailed by the compromised system. The reproducibility of an attack is thus important as well. Security certificates provide a metric for how secure a product is against physical attacks with various strength, and are obtained through specialized certification labs. Every system has to be evaluated and certified for the expected type of attacker one needs to defend against. Suitable combinations of countermeasures will have to be chosen to achieve the desired protection against the relevant physical attacks. Ideally, the total overhead is as low as possible.

1.2 Research Questions

At the onset of this PhD the research landscape in our (sub)field looked as follows. Bilgin et al. extended the provable security of threshold implementations from resisting first-order DPA to resisting univariate HO-DPA [21]. Following this development, Reparaz et al. presented a consolidation of the ISW masking scheme, the Trichina masked AND gate and the threshold implementations masking scheme [117]. The authors then used this consolidation to extend threshold implementations to resist multivariate HO-DPA. Practical security evaluation of masked implementations just transitioned from the slower and more computationally-intensive key retrieval processes to fast and flexible leakage detection tests. Moreover, masking schemes were considered in isolation from countermeasures against active physical attacks. Although it was suggested that TI could achieve an increased resistance against fault attacks [106], no such investigation was yet performed. The research questions of our thesis are directly influenced by and build upon that current state in three dimensions.

With the consolidation of Boolean masking schemes, a first question that arises is whether related patterns can be found in masking schemes that are based around different secret sharing schemes. Secondly, as the presented consolidation is theoretical, we ask whether we can put its results in practice to push the limits of implementation sizes.

Question 1. *Can we expand on the consolidation of Boolean, TI-like masking schemes by finding relations with different classes of masking schemes, particularly the Inner-Product Masking and the Shamir's-Secret-Sharing-based Polynomial Masking scheme? What advantages and disadvantages can we obtain in actual implementations of the consolidated theory?*

We aim to answer these research questions in two ways: horizontally by extending it on the theoretical level, and vertically, by implementing one scheme and testing its security on Field-Programmable Gate Array (FPGA) platforms. From this, we can contemplate on a multitude of implementation cost trade-offs an embedded designer can make.

Several design goals can be targeted when implementing a cipher, e.g. low area complexity, low power consumption or low latency. In all our implementations we focus on minimizing the circuit complexity. Optimizing for this design goal, that is also referred to as the area, is shared with many masked implementations. We encountered several difficulties while investigating how far we can reduce the implementation cost. Firstly, we noticed that several things can go wrong during the process that translates code to a technology mapped netlists (a list of all physical components and connections that together form the implementation). Secondly, we noticed that implementations can still show leakage even when their netlists are conform the properties of the masking scheme. We attribute this unexpected leakage to a violation in the foundational assumption of masking schemes. The underlying assumption that masking schemes make is that the total leakage of the circuit is a linear combination of the leakages of the shared sub-circuits. It is known that some physical phenomena, e.g. crosstalk and power supply noise, can violate this assumption. We suspect that this effect can be observed in some platforms. Based on these observations we formulate our second research question.

Question 2. *Firstly, can we find a way to induce leakage in an FPGA implementation of a netlist that adheres to all properties of a masking scheme? And secondly, what gives rise to this leakage and how does it impact the side-channel security of FPGAs?*

We show that unexpected leakage from coupling through e.g. the power supply, the ground connection, crosstalk or other physical mechanisms is visible on FPGA platforms.

An attacker will find the easiest and cheapest point of attack to break a system. To effectively increase the resistance of a device against physical attacks, the weakest attack points should thus be secured first. Once we have achieved a certain level of side-channel resistance, we should provide resistance against cheap fault attacks. The joint application of countermeasures should ideally come at minimal cost overhead. This justifies looking at their mitigation jointly instead of in isolation. Our third research question addresses the unexplored resistance TI potentially has to offer against fault attacks. As masking alone is known to not be enough to prevent DFA [29], we first look at a type of fault attack that does not require faulty outputs. We show how TI natively resists

Fault Sensitivity Analysis. Afterwards, we investigate how TI can help reduce the implementation cost of a known combined countermeasure.

Question 3. *How does the threshold implementation masking scheme hold up against known Fault Attacks? Can we make current countermeasures against combined attacks more efficient by relying on TI?*

To this end, we merge a known combined countermeasure with threshold implementations and apply it on the standardized PRESENT algorithm. The resulting implementation can serve as a benchmark to compare implementations of future countermeasures against. In the line of our investigation on how to apply schemes correctly, we provide a detailed overview of how we achieve combined security. The search for efficient countermeasures against combined physical attacks is expected to be a research direction that will receive considerable attention in the foreseeable future.

1.3 Roadmap of the Thesis

This thesis consists of eight chapters. A brief description of the content of each chapter and my personal contribution within each is given below. They are clustered with respect to the research question they answer.

Introduction to the thesis

Chapter 1. The first chapter, i.e. this chapter, briefly introduces the increasing importance of security in embedded systems and frames the research contained in this work. From the initial observation that security in embedded systems is necessary, we funnel down to the specific research questions posed and treated here. We situate the work in the right context: **"side-channel resistant implementations of symmetric-key cryptographic primitives in a Grey-Box model using masking schemes on hardware platforms"**. A clear picture surrounding this scope is provided by a taxonomy of the breadth of physical attacks and countermeasures. This chapter is concluded with a roadmap of the thesis that gives an overview of the content and contributions of each chapter.

Chapter 2. The second chapter provides more detail to the surrounding research field of our topic. The preliminaries on physical attacks and

countermeasures are illustrated. The concepts include amongst others side channels, power analysis attacks, threshold implementations, fault attacks and our security validation setup. The evolution of masking is described from the discovery of side channels to the state it reached at the onset of this research. More advanced concepts (e.g. power supply noise) are omitted here and are instead presented in the chapters that rely on them.

Research Question 1. Advancing the Consolidation of Masking Schemes

Chapter 3. The third chapter presents the theory we contribute to the field. It combines a review of state-of-the-art adversary models and countermeasures, and extends the latter. This chapter addresses our first research question: can we extend the consolidation of masking schemes beyond the Boolean masking schemes? We establish relations between Boolean, TI-like masking, Inner-Product masking and Polynomial masking, compare their implementation cost and contemplate on possible trade-offs between security and implementation cost. This study is part of ongoing work with Dhooghe and Gierlichs.

Chapter 4. In the fourth chapter we implement several masked versions of AES. We start by pushing the limits of compact area by providing the first implementation that used the theoretical minimum number of shares to achieve higher-order security. Afterwards, we present a higher-order threshold implementation of AES using the classical number of shares. An evaluation in the form of leakage detection is provided for all implementations. We additionally compare our implementations with masked AES designs that were published during the course of this PhD. The trade-offs in implementation cost for Boolean, TI-like masking are highlighted across first- and second-order secure implementations.

These results are presented by De Cnudde, Bilgin, Reparaz, Nikov, Nikova and Rijmen at the Smart Card Research and Advanced Application Conference (CARDIS 2015) [42] and the Conference on Cryptographic Hardware and Embedded Systems (CHES 2016) [47].

Research Question 2. Fundamental Leakage Assumption

Chapter 5. In the fifth chapter we point out evidence of leakage from masked implementations. Our focus is directed towards the process of inducing leakage

by manipulating placement and routing rather than towards investigating the specific sources of the leakage. This study provides a first insight in how a secure netlist of an implementation can still leak. We stress that we only detect rather than exploit leakage.

These results are presented by De Cnudde, Bilgin, Gierlichs, Nikov, Nikova and Rijmen at the International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2017) [41].

Research Question 3. Threshold Implementations vs. Fault Attacks

Chapter 6. In the sixth chapter we direct our attention to our third research question. We investigate the resistance of glitch-resistant masking schemes under Fault Sensitivity Analysis (FSA), an actively triggered side-channel analysis attack. In this preliminary investigation, we argue that under certain assumptions, properly implemented masking schemes achieve an increased resistance against this active attack.

These results are part of joint work with Arribas and Šijačić that is published at the Fault Diagnosis and Tolerance in Cryptography (FDTC 2018) workshop [6].

Chapter 7. After the preliminary investigation of Chapter 6, we present an application of a combined side-channel analysis and fault attack countermeasure to the PRESENT block cipher. We leverage the efficiency of threshold implementations to achieve a more cost effective implementation of Private Circuits II compared to its native ISW (or Private Circuits) basis. We go through the process of applying the countermeasure on both the data and control path, we test the result on its side-channel resistance and subdue the countermeasure to known DFA attacks.

This chapter is based on joint work with Nikova and is published at the workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2016) [45] . An extended version is published in the IEEE Transactions on VLSI journal [46].

Conclusion

Chapter 8. In the eighth and final chapter we conclude our thesis. We review our research questions in light of our presented results. We finally propose

open questions for future research in the field of masking and masking-based combined countermeasures for embedded systems security.

Chapter 2

Physical Attacks and Countermeasures

In this chapter we detail the technical background required to understand our research results. We start by introducing our general notational conventions. More specific notations, definitions and concepts are kept for their respective chapters. As we stated in the introduction, modern ciphers offer strong, computationally unbreakable security on a mathematical level, but many of their implementations are susceptible to physical attacks. We provide insights into these attacks and their countermeasures for the two families of physical attacks: passive attacks or Side-Channel Analysis (SCA) attacks in Section 2.2 and active attacks or Fault Attacks (FAs) in Section 2.3. For both these attack classes, we briefly introduce a number of well-known attacks and known countermeasures and discuss their advantages and disadvantages.

The main focus of this thesis is securing symmetric-key algorithms against these physical of attacks. In order to confirm the security of our implementations, we have to subdue them to an attacker. For this purpose we use a general technique that was adopted by the community at the onset of this research, namely leakage detection tests or Test Vector Leakage Assessment (TVLA). This method is a set of very strong statistical tests that allows to quickly and easily assess the (possibly unexploitable) leakage of an implementation. The details of this technique are presented in Section 2.4. The implementation and the attacker interact in a certain environment that can be very noisy (in practice) or artificially clean (in a lab). The validation of our designs are performed in the latter environment and we describe our measurement setup and methodology in Section 2.4.

2.1 Notation and Definitions

We refer to a finite field with 2^n elements as $\mathbb{GF}(2^n)$ and use $|\mathbb{GF}(2^n)| = 2^n$ to denote its size or cardinality. The exponent n will be determined by the size of the element, e.g. a 1-bit element resides in $\mathbb{GF}(2)$, a nibble in $\mathbb{GF}(2^4)$ and a byte in $\mathbb{GF}(2^8)$. The number of bits that are equal to one in an element x of a field is called its Hamming Weight (HW) and is denoted by $\mathrm{HW}(x)$. The difference of the HWs between two elements x and y is referred to as their Hamming Distance (HD) and is written as $\mathrm{HD}(x, y)$.

We use small and bold letters to describe elements of $\mathbb{GF}(2^n)$ and their sharing respectively. Upper-case characters are used for stochastic variables, and bold upper-case characters indicate stochastic sharings. The letter r will generally denote a unit of randomness. One unit of randomness is defined as a set of independent and uniformly distributed bits with the field size of the value as its cardinality. We assume that any possibly sensitive variable $a \in \mathbb{GF}(2^n)$ is split into s shares $(a_1, \ldots, a_s) = \mathbf{a}$, where $a_i \in \mathbb{GF}(2^n)$, in the initialization phase of the cryptographic algorithm. A possible way of performing this initialization, which we adopt, is as follows: the shares $a_1, \ldots, a_{s-1}$ are selected randomly from a uniform distribution and a_s is calculated such that $a = \sum_{i \in \{1,\ldots,s\}} a_i$. The set of all such correct sharings is represented by $Sh(a)$. We refer to the jth bit of a as a^j unless $a \in \mathbb{GF}(2)$. We use the same notation to share a function f into s shares $\mathbf{f} = (f_1, \ldots, f_s)$. The number of input and output shares of $\mathbf{f}$ are denoted by s_{in} and s_{out} respectively, and we will then call $\mathbf{f}$ an (s_{in}, s_{out})-sharing.

We refer to field multiplication, field addition and concatenation of two values as $\otimes$, $\oplus$ and $\Vert$ respectively. Where more convenient, the multiplication of two values $x \otimes y$ is sometimes written as xy and their addition is sometimes written as $x + y$. The inner product of x and y is denoted by $\langle x, y \rangle$. We use special register brackets $[\,]_{reg}$ to indicate that the value within the brackets is registered before starting further computation. Generally, we reserve the letters d and e for the order of SCA security and the number of tolerable injected faults for FA security respectively. Finally, the letter t indicates the algebraic degree of a function.

2.2 Passive Physical Attacks and Countermeasures

Passive physical attacks or Side-channel Analysis (SCA) attacks are carried out by a passive attacker. By merely observing physical attributes of a computing device, an attacker attempts to retrieve a secret key. A source that leaks such

information on the key is known as a side channel, and arises from measurable physical phenomena of the underlying digital circuit. These side channels are often unintentional and difficult to prevent, and to make matters worse, exploiting them is cheap and relatively easy. As such, side channels form a major threat to the security of deployed cryptosystems when an attacker has access to a device. Using off-the-shelf equipment, an attacker can gain information on the key through the timing of operations, their power consumption or EM radiation, acoustics, various optical phenomena, or temperature variations. As these side channels can be observed from the interface of the device or its environment, these information sources are called non-invasive side channels.

A determined attacker might go further than non-invasive observations. Invasive attacks consist of an attacker intruding in the system to access otherwise unexploitable side channels. Probing is an expensive but powerful invasive attack: using thin needles placed over wires over which secret key bits traverse, the attacker can straightforwardly read out values of the key. An attack like this is often not considered a true side-channel attack as it exploits the information directly. A less extreme (semi-invasive) method is to enhance the signal quality of an Electromagnetic (EM) antenna by first decapsulating the chip before measuring to allow the antenna to be in closer proximity to the leakage sources. Needless to say, the more invasive an attack is the more involved and expensive the attack becomes. With increasing invasive character, the effectiveness of the attack tends to increase as well.

Our focus in what follows is predominantly on non-invasive passive attackers, but can in certain cases be extended to (semi-)invasive ones.

2.2.1 Side-Channel Analysis Attacks

What follows is a brief description of some very powerful attacks.

Simple Power Analysis. Simple Power Analysis (SPA) works on a single or a few power traces (EM traces can work as well) measured from an encrypting or decrypting implementation of a cryptographic algorithm. Hidden information in these traces are exploited to attempt to obtain information on the secret key. The classic example of SPA can be found in public key cryptography, where the choice of a longer multiplication or a faster squaring is chosen by a 1 or 0 bit in the key. Based on the differences in the execution time of the multiplication and the squaring, which can be observed in the traces, the key can be extracted. In platforms that are more parallel in their operations (i.e. traditional hardware) SPA becomes much more complicated than in sequential platforms (traditional

software). Details of the implementation need to be known and the noise needs to be minimized in order for an attack to be successful.

Differential Power Analysis. In contrast to SPA, Differential Power Analysis (DPA) uses multiple power or EM traces. The more traces are collected, the more robust the attack becomes to noise, and the higher the success rate of a key retrieval. It is easiest explained through an example.

We start with a collection of measured traces of the instantaneous power consumption of an encryption algorithm working on different plaintexts under a fixed key. Next, we will decide on a so-called leakage model. The instantaneous power consumption of a device can be modeled as a noisy leakage function $L(.)$ that has the intermediate values and operations happening at a certain time as input. A common simplification is to assume that the total leakage function can be decomposed in different, linear leakage components related to the different intermediate values, and an additional independent Gaussian noise source [35]. Let's consider a secret variable $x \in \mathbb{GF}(2)$ and assume the leakage of a value 0 differs from the leakage of a value 1, i.e. that $L(0) \neq L(1)$. This model is known as the Hamming Weight leakage model, as the leakage is related to the Hamming Weight (HW) of x. Another leakage model that is particularly useful for hardware is the Hamming Distance model: when a newly computed intermediate variable x' overwrites the old variable x in memory, the resulting leakage is of the form HD(x',x). With the HD as leakage model, we can proceed with the attack by guessing a part of the key (divide-and-conquer strategy). This guess and the known plaintext are used to calculate the HD that occurs, which is then correlated with the power trace. If there is a high correlation, the subkey was likely guessed correctly.

DPA attacks come in different flavors, e.g. one can exploit the difference of means [84] or the Pearson correlation [31] to settle on a likely key guess. A more complex, and more powerful version of this attack is Higher-Order DPA (HO-DPA) [35, 95]. The most intuitive way to explain HO-DPA is with the probing model. If we assume a very powerful attacker, having a number d of probes (thin needles in an invasive scenario) at his disposal and all details of the circuit, we can think of first-order DPA as the attacker having only one probe $d = 1$. In an unprotected implementation, this probe can be used to (theoretically) read out any sensitive value on a wire. With this one probe, an attacker can compromise the security of an unprotected implementation. One way to protect an implementation against an attacker with $d = 1$ probe is to mask all sensitive values: x is split in $x_1 = x + r$ and $x_2 = r$ where r is a uniformly distributed random value. If the implementation now operates on both x_1 and x_2 instead of on x, an attacker with a single probe will be unable

to compromise the circuit. If now $d > 1$, we have a higher-order DPA and the attacker has more information he can (nonlinearly) combine to exploit the side channels. In our example, the attacker can read out both x_1 and x_2 to obtain the sensitive value x. We further distinguish between univariate attacks and multivariate attacks. In the former the attacker only exploits leakage information from a single, fixed point in time, whereas in the latter, the attacker combines leakage information from multiple points in time.

2.2.2 Side-Channel Analysis Countermeasures

One way to categorize countermeasures against DPA is on how they decrease the sensitive information exposed in the side channel. Put differently, they can be categorized on how they decrease the Signal-to-Noise Ratio (SNR) captured by the measurements [90]. Countermeasures based on secure logic styles, a subset of the hiding family of countermeasures, decrease the SNR by equalizing the data-related power consumption of a circuit implementation [134, 135]. An alternative way to decrease the SNR is to increase the noise component of the signal rather than reducing the informative signal component in the side channel. Another subset of the hiding countermeasures increases the noise component by randomly shuffling the operations in time [137]. The masking family of countermeasures increases the noise component in a different way, by processing algorithmically-randomized data while maintaining the overall correctness of the circuit [35, 68].

Hiding Countermeasures. It has been shown in many works that equalizing the power consumption is challenging to achieve as strict assumptions need to hold, e.g. no early signal propagation, no imbalanced routing [90]. Satisfying these assumptions become increasingly harder to attain in advanced technology nodes as parasitic effects increase [104]. Ad-hoc countermeasures that introduce noise by shuffling suffer security issues as well, as they are easily defeated by preprocessing the measured traces. Masking on the other hand offers provable security, and can therefore be made more robust against these issues [112].

Masking Countermeasure (and Glitches). Many hardware tailored masking schemes have been developed, but not all of them result in actual secure designs. The resulting insecurities in the masking scheme are mainly due to wrong assumptions. An example can be found in some pioneering schemes [33, 35, 77, 108, 136] that assume transistor gates to execute in a sequential manner and only once. This behavior was shown to be overly optimistic due to glitches and early propagation of signals [90, 91, 100]. A glitch (or hazard) is an unwanted

transition on a signal before it stabilizes to its intended value. We use the term early propagation to denote various delays that can affect the arrival times of new values on wires. Another violated assumption is so-called memory recombination and is easiest explained with an example. When a value x is masked with a random mask r into two shares we get $x_1 = x + r$ and $x_2 = r$. When these shares are subsequently stored in the same register their mask r cancels out which creates a vulnerability.

We illustrate what goes wrong in more detail when using the ISW AND-gate naively. ISW has a strict execution order, which should be enforced through registers (Figure 2.1). In equations, we will denote such an execution order with special brackets $[x]_{reg}$ that indicate storing a result x in registers before proceeding.

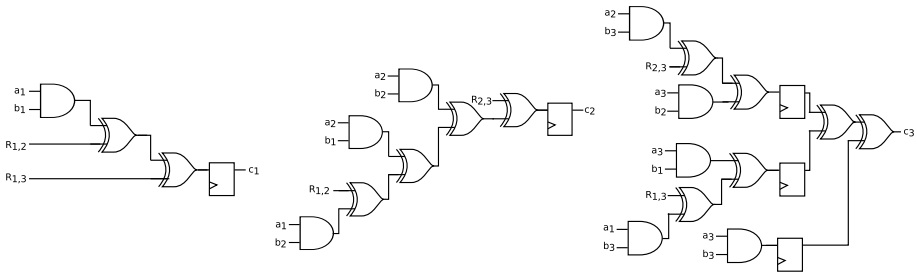

Figure 2.1: ISW with an imposed execution order from the registers

Due to various routing delays some signals arrive faster than others and an unmasked value can potentially reside on a wire at a specific moment in time. We can have a look at what happens in this case when some signals propagate faster than others. The secure way to evaluate c_3 is using registers:

$$c_3 = [a_3 b_3]_{reg} \oplus [a_3 b_1 \oplus r_{1,3} \oplus a_1 b_3]_{reg} \oplus [a_3 b_2 \oplus r_{2,3} \oplus a_2 b_3]_{reg}$$

If the registers are omitted, we can shuffle the terms and put the late signals to the right, which gives the following expression.

$$\begin{aligned} c_3 &= a_3 b_1 \oplus a_3 b_2 \oplus a_3 b_3 \oplus r_{1,3} \oplus a_1 b_3 \oplus r_{2,3} \oplus a_2 b_3 \\ &= a_3 b \oplus r_{1,3} \oplus a_1 b_3 \oplus r_{2,3} \oplus a_2 b_3 \end{aligned}$$

The unmasked value b is available for probing at at least one given point in time, rendering the "protected" circuit insecure. While this explanation is overly simplified, actual attacks that target glitches have been successful [91].

In order to make masking as independent of the target platform as possible, Nikova et al. introduced the Threshold Implementations (TI) masking scheme. It is designed to inherently resist the security deterioration emerging from glitches and early propagation [105–107]. Optimizations building on TI have been presented in the form of higher-order TI by Bilgin et al. [21], through the Consolidated Masking Scheme (CMS) by Reparaz et al. [117] and through Domain-Oriented Masking (DOM) and the Unified Masking Approach (UMA) by Gross et al. [70–72]. All the above offer various trade-offs in security, latency, circuit complexity and required randomness (which also influences the actual circuit complexity). These recent developments have been validated using theoretical measures, e.g., by toggle counts or formal tools [7, 24, 116] which both have limitations.

The underlying security model in all masking schemes is the same. The following underlying assumption is expected to hold: "the total power consumption of the chip is a linear combination of the power consumptions from the component functions and shares". If this assumption is violated in any way, e.g., through coupling, then the security of the scheme is not guaranteed. To distinguish leakage due to coupling from leakage due to glitches, early propagation or memory recombinations, we will refer to coupling leakage as out-of-model-leakage. Throughout, we will in general refer to TI, CMS or DOM as Boolean, TI-like masking schemes or threshold implementations for simplicity.

2.3 Active Physical Attacks and Countermeasures

Active physical attacks or Fault Attacks (FA) are mounted by an active attacker. In contrast to passively observing side channels, the attacker will interact with the implementation in a way that induces stress until some unexpected behavior arises, e.g. faulty outputs [26]. Using knowledge about the effects of the injected fault on the device (the fault model) an attacker can extract a secret key with relative ease. As with side channels, protection mechanisms should prevent or harden the device against this type of attack. This is made difficult by the wide variety of faults that can be injected and by the uncertainty of protecting against all, even potentially unknown attack vectors.

FAs can be mounted at a varying degree of complexity and cost related to the precision of the injected fault in both space and time. To facilitate their classification and threat we can again make the distinction based on their level of intrusion.

Non-invasive attacks exploit tampering with interfaces or with the environment of the device. A particularly effective injection is a clock or power supply

glitch, which is the injection of a short pulse on the clock signal or supply voltage. Other examples are the injection of electromagnetic pulses or varying the temperature. Invasive attacks on the other hand are more expensive and involve decapsulating or even altering the chip functionality. The revealed surface can than be hit with a laser to mount an optical attack. In an extreme case, a Focused Ion Beam (FIB) can be used to cut and rewire metal wires in a chip, to e.g. tie the key to the output at all times.

In what follows we describe a set of well known fault attacks.

2.3.1 Fault Attacks

Before we describe some attacks, we give more details on fault models. A fault model is a crucial part of fault attacks and can be seen as a mathematical representation of the effect of a fault injection. Following characteristics are generally used to describe a fault.

- **Fault Granularity.** The number of affected bits can vary from a very strict single bit manipulation to a coarser one of a nibble, byte, or even larger numbers of (unrelated) bits.
- **Fault Type.** The type of injected fault is related to the effect the injection has on the affected bits. This can range from bit flips (targeted bits are complemented), bits stuck-at-zero or stuck-at-one, setting bits to 1 or resetting bits to 0, and fully random faults following a given distribution.
- **Spatial and Temporal Fault Control.** Control over a fault can refer to either control in space or in time. Control in space relates to the locality of the injection: is the injected fault distributed over the chip or is it localized in a very specific part, e.g. a register? Control in time is spanned by the extremes of having no control when the fault is injected to a very specific moment within a clock cycle.
- **Fault Duration.** The fault duration is the life-span of the induced error and can be transient (only affected in one or few clock cycles), permanent (until the device is reset), or destructive (irreversibly damaged).

We can illustrate these fault models using following well-known fault attacks. Their requirements for success vary from a more relaxed to a more strict fault injection. Some attacks require a fault model with an unrealistic level of precision, and as such, it is hard to argue their practical relevance.

Safe-Error Analysis (SEA). A safe-error analysis attack (or Ineffective Fault Analysis (IFA)) is a very strong attack that can break many countermeasures against fault attacks. An attacker would inject a very specific fault e.g. a stuck-at-zero in a very specific place of the implementation e.g. a bit of the key register. The purpose of the fault injection is to affect the device in such a way that the output is only faulty if a sensitive variable has a certain value. If an attacker injects a stuck-at-fault on the first bit of the key register and the output is correct, the value of that bit is gained. The stuck-at value and the correct behavior of the device reveal the actual value of that bit of the key register. The attacker can proceed to scan the whole key register in this way. More generally, the attacker can gain information to retrieve the key by only using the fault model and the information whether or not the computation was correct. The fault model for this attack to succeed is very strict and requires a skillful attacker.

Differential Fault Analysis (DFA). This fault attack exploits differences between faulty and correct ciphertexts, and the type of injected fault in order to gain the device's secret key. The core idea behind this attack is the mathematical differential cryptanalysis, where a series of differential equations are solved to discriminate key guesses. The fault model used in this attack varies from attack to attack, and in general, the stricter the fault model, the less correct/faulty ciphertext pairs are needed to retrieve the key.

Fault Sensitivity Analysis (FSA). The core motivation of Fault Sensitivity Analysis is to allow an attacker to bypass FA countermeasures that withhold faulty outputs to prevent DFA. Its key idea is to create a profile of the propagation delays of a chip under a set of inputs and a known key. Once this profile is obtained, the propagation delays of a chip with unknown key can be correlated to extract the key. A key concept is Fault Sensitivity (FS), and denotes the point of failure when a fault injection is gradually intensified, e.g. the width of a clock glitch or power glitch, or the intensity of a laser [124]. This attack requires control over the time of the fault injection, but is more relaxed with respect to the fault location, type, duration and granularity. A clock glitch generator has shown to provide enough resolution for successfully mounting FSA attacks on both IC and FPGA implementations [56, 57]. Several attacks are derived from this principle, e.g. Differential Fault Intensity Analysis [63].

2.3.2 Fault Attack Countermeasures

Fault attacks are very powerful ways to break cryptosystems. Security critical devices should therefore be protected with countermeasures. Countermeasures against FAs are generally categorized as follows.

Redundancy. Redundancy can be added in two ways, either through spatial or temporal redundancy, or through mathematical redundancy. In the former, a given computation is either repeated (time redundancy) or the same operation is computed in parallel (area redundancy) in order to check whether a fault was injected [80]. In the latter, error correction or detection codes are appended to the intermediate values [79].

Infective Computation. The goal of infective computation as a countermeasure is to obscure the relation between the injected faults and the resulting faulty ciphertext. When a single, well-located fault is injected, it is propagated (rather than detected) throughout the ciphertext in a complex way to make DFA more difficult [89].

Detectors and Shields. Active sensors or detectors, e.g. to check for light from a laser or to check for temperature variations, may lessen the efficacy of the fault injections. Another approach is to rely on shielding countermeasures, e.g. shielding parts of an Integrated Circuit (IC) to prevent optical attacks [133]. In case tampering is detected, an alarm can be triggered to withhold the faulty ciphertexts from being output to prevent DFA attacks. On-chip filters can flatten out glitches on the supply voltage. Or alternatively, the clock signals can be generated internally and voltage regulators can be integrated to prevent these signals from being tampered with by a non-invasive attacker. Adding noise, e.g. randomizing executions or providing an unstable clock can be used to lower the injected fault's precision in time.

2.4 Security Validation

In this work we propose several implementations of masking schemes and one implementation of a combined countermeasure. When presenting implementations, it is important to validate their security against attacks before we derive any meaning from them. We are concerned with masking in hardware so we have two options to materialize our implementations: Application-Specific

Integrated Circuits (ASICs) or FPGAs. Due to the high financial cost and time effort required to produce an ASIC, our choice of the platform for our validation goes to FPGAs. This has the additional advantage that we can explore and validate many different countermeasures and variations. This choice leads to an important question: "Do our results translate to ICs if we only validate them on FPGAs?". The answer is negative, as their underlying structure is very different. ASICs, and more specifically standard cell based designs are gate oriented, whereas FPGAs are Look-Up Table (LUT) (LUT) based. This difference does not exclude masking schemes from achieving security on ASICs, and when properly implemented they should provide the expected SCA resistance.

In what follows we describe the platform on which we implement our schemes and describe our evaluation methodology.

2.4.1 Side-Channel Measurement Setup

Our design and validation process always share following steps and properties.

Preliminary tests. We generally perform a preliminary evaluation with the simulation tool from [116], which allows us to refine our design and quickly catch algorithmic flaws in our code. We then proceed with the side-channel evaluation based on actual measurements.

Synthesis. Our designs are coded in VHDL and we use standard design flow tools (Xilinx ISE) to synthesize our designs. Depending on the experiment we run, we will use different synthesis constraints. We provide a brief description of the ones we use and refer to the Xilinx Constraint Guide [141] for their full description.

- **Keep Hierarchy.** "Keep Hierarchy" is a synthesis and implementation constraint and is commonly referred to in papers about threshold implementations [21, 22, 103, 110]. HDL designs are generally a collection of hierarchical modules and submodules. In masked implementations this constraint is used to avoid optimizations over share boundaries. It preserves the hierarchy throughout the implementation process and avoids the flattening of the design, which could unintentionally merge different shares and compromise the SCA security. Three values can be set for this option: true, soft and false. True preserves the design hierarchy throughout both synthesis and implementation. Soft keeps the hierarchy during synthesis but not during the implementation phase. False allows

all the submodules of the design to be merged within the top level module. In a masking context, the option is set to true globally as a synthesis option.

- **Keep.** "Keep" is a constraint that influences the mapping phase of the implementation. It avoids nets from being merged into a single logic block. Taking the AND/XOR function $X \oplus YZ$ as example, the HDL code would explicitly declare an AND and an XOR operation. In the subsequent mapping phase, both gates would be merged into a single LUT. This constraint is applied to signals in the HDL code.
- **Prohibit.** "Prohibit" is a constraint that forbids the use of selected CLBs or Slices during Place-and-Route. It can be used to guide implementation resources to certain areas on the FPGA.

Platform. We use one of two different platforms depending on the experiment we launch. Both boards are designed specifically for side-channel evaluation.

For very low noise settings we use a SASEBO-G (Side-channel Attack Standard Evaluation Board) board [1]. The SASEBO-G features two Xilinx Virtex-2 Pro FPGAs: an XC2VP7 to hold our cryptographic implementations and an XC2VP30 for handling the communication between the board, the measurement PC and other equipment.

When the SASEBO-G is too small to hold our designs we perform the evaluation on the more recent SAKURA-G board [74]. The SAKURA-G (Side-Channel Attack User Reference Architecture) provides an SCA evaluation environment based around two Xilinx Spartan-6 FPGAs. An XC6SLX75 to hold our cryptographic implementations and an XC6SLX9 for controlling the communication between the board, the measurement PC and other equipment.

Low noise. The lower the noise in our measurement setup, the faster vulnerable implementations will show leakage. The platforms themselves are very low noise (DPA on an unprotected AES succeeds with few tens of traces). We further reduce the noise in our setup by providing a stable clock of 3MHz to the FPGAs. This relatively low clock frequency allows us to measure both the dynamic as well as the static component of the power consumption of our implementations. We measure the instantaneous power consumption as the voltage drop over a 1Ω resistor between the ground lines of the crypto FPGA core and the board. We acquire the power traces with a Tektronix DPO 7254C oscilloscope.

Randomness. The randomness for the initial sharings of the plaintext and key are provided by an AES-128 implementation in Output Feedback (OFB) mode that resides on the control FPGA. In case the design requires additional randomness, i.e. for mask refreshing, we supply them by a PRNG that runs on the crypto FPGA. The PRNG is built from a fully unrolled round-reduced PRINCE [27] in OFB mode that receives a fresh key from the AES on the control FPGA at the start of each new encryption. The PRINCE PRNG outputs 64 bits of randomness per call, so we use multiple instances in parallel when more fresh random bits are required per clock cycle. We interleave the execution of the single-cycle PRINCE PRNG with every clock cycle of the masked implementation in order to decrease the impact of noise introduced by the PRNG.

2.4.2 t-Test Based Leakage Detection

One way to test for side-channel leakages that can potentially lead to successful key retrieval attacks in cryptographic systems is based on the t-test statistic [16, 65, 127]. As opposed to DPA, it has the advantage that it is independent of an underlying leakage model. Despite this independence, it still is sensitive enough to uncover a wide range of potential problems, both in the dynamic as well as the static power consumption. After acquiring a sufficient number of power consumption/EM traces, the traces are divided in two sets, A and B, based on an intermediate value in the computation. Throughout this work we employ the non-specific t-test, which fixes the input for one of the sets while randomizing the input for the other set. Unless stated otherwise, we choose zero for the fixed input. For testing leakage in the first-order statistical moment, the t-test statistic is calculated sample-wise on the two sets A and B as:

$$t = \frac{\overline{T_A} - \overline{T_B}}{\sqrt{\frac{s_A^2}{N_A} + \frac{s_B^2}{N_B}}}$$

where $\overline{T_i}$, s_i^2, N_i are the sample mean, sample variance and sample size of the set $T_{i \in A,B}$ respectively.

If no t-test value exceeds a certain confidence threshold $\pm C$, no relation between the processed intermediate value and the mean instantaneous power consumption can be found. When the t-test statistic exceeds $\pm C$, the power consumption and the processed intermediate values are related in a statistically significant way with a confidence level related to C, making the device potentially vulnerable to first-order SCA attacks. Throughout this paper, we set the confidence level to $\pm C = \pm 4.5$, which corresponds to a 99.999% certainty of the concluded outcome.

2.5 Conclusion

In this chapter we introduced various physical attacks that threaten implementations of symmetric-key algorithms. We explained the basic, most well-known attacks and illustrated how known countermeasures can thwart them. The major sources that enable side-channel leakage were addressed as storing intermediate values in memory elements or combinational operations to calculate these intermediate values. In contrast, FA vulnerability is less related to actual implementations but more to the underlying algorithm. We categorized faults using their fault model which is related to the cost of their injection.

We devoted special attention to the problem of glitches and early propagation in hardware masking schemes, and showed how this issue was resolved in the threshold implementations masking scheme.

The measurement setup on which we validate our side-channel secure implementations was detailed together with the t-test leakage detection method. We focus on detecting leakages rather than actually exploiting them for a faster evaluation cycle.

This section provides a foundation for the material that comes, and more specialized knowledge and definitions are elaborated in their respective chapters.

Chapter 3

State of the Art Hardware Masking Schemes

In this chapter we present and relate three well-known masking schemes: Boolean (and more precisely TI-like), Inner-Product and Polynomial masking. While we limit our scope to hardware security, the presented descriptions can be converted to software with some care. This chapter treats our first research question theoretically: we expand on the consolidation of Boolean, TI-like masking schemes by finding relations with different classes of masking schemes. We propose additional trade-offs designers can make in their implementations.

The consolidation we present is a continuation of the work by Reparaz et al. that points out relations between the ISW masked multiplication, the Trichina AND-gate and TI [117]. Parallel to this work, other consolidations have been initiated as well. Gross et al. translated a mask refreshing approach from software to hardware masking schemes [70] and Poussier et al. [111] have presented a relation between the Inner-Product masking scheme and Direct-Sum Masking.

We open the chapter by introducing various security models for masking schemes. These security models define up to how powerful an attacker can be resisted by a masking scheme developed soundly in that model. Different properties of different models like composability of masked algorithms and the inclusion of a range of (unintentional) hardware behaviors are established and form the subject of Section 3.1. We proceed with an overview of the state of the art in masking schemes in the following three sections: Boolean, TI-like masking in Section 3.2, Inner-Product masking in Section 3.3, and Polynomial masking in Section 3.4. From their descriptions we extract a general structure shared

by all three schemes which we discuss in Section 3.5. We prove the security of this general scheme in Section 3.6 and propose several variations with different trade-offs in Section 3.7. We conclude the chapter and present open research questions in Section 3.8.

This chapter provides a solid theoretical frame for this thesis and additionally holds our theoretical contribution to the field. Several of the constructions we highlight here are built upon later on, in more practical chapters. The work presented in this chapter is a segment of a vast topic with many open questions and is part of ongoing work with Gierlichs and Dhooghe.

3.1 Security Models

Before we propose any algorithms, we summarize various models in which the security of masking schemes against d^{th}-order side-channel analysis can be argued. A security model essentially formalizes the relation between the power an attacker has on one hand, and a resulting level of side-channel resistance on the other. An attacker in general will always try to break a masking algorithm by attempting to obtain enough information to unmask any masked value. If the attempt is unsuccessful with d observations, the algorithm is said to be secure at order d. Note that in reality an attacker will not give up here, but just acquire more observations and mount an attack at a higher order. Still, these models have practical value as the complexity to mount a successful $(d+1)^{th}$-order attack against a dth-order secure implementation is exponential with d when sufficient noise is present. The models presented here have been successfully used to reason on masked algorithms and have helped to uncover flaws in several proposals. We start the overview with the well-known (not composable) probing model and conclude with the (composable) robust probing model, which is a recent proposal that is suited for hardware platforms.

The d-probing model introduced by Ishai et al. [77] allows an attacker to read up to d intermediate values, inputs and outputs, and is formalized by Definition 3.1.1.

Definition 3.1.1. (d-Probing Security.) [120] An algorithm is d-probing secure if and only if every set of d (intermediate) values is independent of any sensitive variable.

In other words, an algorithm secure in the d-probing model will resist an attacker exploiting information from up to d intermediate values, inputs and outputs.

The d-probing model can also be expressed using a simulator that tries to simulate an attacker's view using black-box access to the algorithm only. This leads to the d-Non Interference model introduced by Barthe et al. [14].

Definition 3.1.2. (d-Non Interference.) [14] An algorithm is d-Non Interferent (d-NI) if and only if for any set of d_1 probes on its intermediate values and set of d_2 probes on its output shares, the totality of the probes can be simulated by at most $d_1 + d_2 \leq d$ shares of each input.

When more complex algorithms are masked, an easy and convenient way to provide security is by building them from smaller masked algorithms with so-called composable security. Otherwise, having an output of an algorithm as input to another can lead to additional information being leaked unintentionally. A stronger security model that provides such composability is the d-Strong Non-Interference model proposed by Barthe et al. [14].

Definition 3.1.3. (d-Strong Non-Interference.) [14] An algorithm is d-Strong Non-Interferent (d-SNI) if and only if for any set of d_1 probes on intermediate variables and any set of d_2 probes on output shares such that $d_1 + d_2 \leq d$, the totality of the probes can be simulated by only d_1 shares of each input.

We have seen in Chapter 2 that things can go wrong when actually implementing masking algorithms, e.g. the recombination of shares through unwanted glitches. Furthermore, in the context of software implementations, one well-known example is unwanted leakage from transitions in registers or memory elements. In hardware implementations, in addition to glitches, it has recently been shown that coupling between shares can lead to detectable leakage [41].

As such, not only composability but robustness against such physical deficiencies as well are two advantageous properties for the actual secure implementation of masking schemes. Fortunately, the recently proposed robust probing model offers both. It incorporates the physical deficiencies using a tweak on the probing model that allows a probe to be so-called ϵ-extended. The robust probing model handles glitches, memory transitions and coupling that can occur in real-world implementations in the following way.

1. For glitches, any combinational sub-circuit with ϵ number of inputs is susceptible to combinational recombinations. These can be modeled with ϵ-extended probes so that probing the output of that sub-circuit allows the adversary to observe all its ϵ inputs.

2. For transitions or memory recombinations in a register, a 2-extended probe that probes the input and output of that register allows the adversary to obtain any pair of values stored in two of its consecutive invocations.

3. For coupling between any set of values $V = (v_1, ..., v_d)$, the recombinations can be modeled with specifically c-extended probes such that probing the value v_i allows the adversary to observe c nearby values. As a convention $c = 1$ means no recombinations from coupling.

The robust probing model is formalized as follows.

Definition 3.1.4. ((g, t, c)-Robust d-Probing/d-NI/d-SNI Security.) [58] An algorithm is said to be (g, t, c)-robust d-probing/$d-$NI/$d-$SNI secure if and only if the algorithm is d-probing/$d-$NI/$d-$SNI secure with an adversary whose probes are extended with glitches if $g = 1$ (if $g = 0$ glitches are assumed mitigated at the implementation level), with transitions if $t = 1$ (if $t = 0$ recombinations in the memory are assumed avoided at the implementation level) and with coupling if $c > 1$ (if $c = 1$ possibly detrimental effects from coupling are assumed avoided at the implementation level).

The advantage of the robust probing model is its inclusion of physical defects. The g, t and c parameters allow trading off more conservative and more risky designs tailored to real-world deployment.

In the remainder of this chapter, the algorithms we describe will be secure in the $(1, 0, 1)$-robust Strong Non-Interference (SNI) model. In other words we take into account the presence of glitches, we do not consider coupling, and we do not take into account memory recombinations. Care needs to be taken to avoid coupling and memory recombinations when implementing these algorithms.

Before describing the masking schemes, we draw attention to the following two general remarks.

- The underlying assumption of all masking schemes is that the total leakage of the device can be written as a linear combination of the leakages of the individual shares. In the algorithms we present, this condition must be satisfied for their security to hold.
- Not being robust and composable does not imply that an implementation is insecure and as such, composability is not always needed and is often too strong and expensive. The algorithms we describe in this work are therefore open for optimization.

3.2 Boolean, TI-Like Masking

The first d-probing secure multiplication on a set of $d + 1$ input shares was proposed by Ishai, Sahai and Wagner (ISW) [77]. Their proposed algorithm

achieves d-SNI security when the gates evaluate only once and in a strict sequential order [14]. We have seen in Chapter 2 (Preliminaries) what happens when that strict order is violated, e.g. due to wrong assumptions on the underlying platform regarding glitches and/or early propagation. Enforcing the sequential execution of the intermediates can be costly to achieve in hardware in both circuit complexity (by adding registers) or in design effort (by adding delays in the layout stage). The need for a strict sequential execution order was abolished by the non-completeness property introduced in the Threshold Implementations (TI) masking scheme [105–107]. A threshold implementation requires fulfillment of the following properties.

Correctness. The first property of TI is one shared with all masking schemes. Correctness is fulfilled when the output of an unprotected operation equals the unmasked output of a masked operation, and is written formally as:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \implies y = \bigoplus_i y_i = \bigoplus_i f_i(\mathbf{x}) = f(x)$$

Uniform Masking. The second property of TI, that is also shared with other masking schemes, is that all correct sharings $Sh(x)$ of x must occur with equal probability. Formally:

$$\forall \mathbf{x} \in Sh(x) \implies Pr(\mathbf{X} = \mathbf{x} | X = x) = p, \text{ and}$$

$$\sum_{\mathbf{x} \in Sh(x)} Pr(\mathbf{X} = \mathbf{x}) = Pr(X = x)$$

$d^{th}-$**Order Non-Completeness.** Any combination of up to d component functions f_i of $\mathbf{f}$ must be independent of at least one input share x_i.

The non-completeness property is unique to TI and provides a basis for various optimizations in implementation cost. Traditionally, TI uses $2d + 1$ shares for a dth-order SCA secure multiplication. One alley of optimizations was the reduction of this number of input shares from $2d + 1$ for a multiplication to the theoretical minimum of $d + 1$. The first optimization targeting $d + 1$ shares is known as the Consolidated Masking Scheme (CMS) [117], but was shown to lack composability [15] due to the mask refreshing. We note once more that the lack of composability does not imply insecure designs, it merely makes security proofs more complex when an implementation is assembled from individually secure but uncomposable algorithms. Afterwards the Domain Oriented Masking and Unified Masking Approach were introduced by Gross et al. [70, 72] which

also leverage the non-completeness property of TI. Their algorithm leads to an ISW multiplication with quadratic randomness consumption. Faust et al. have recently shown that by registering the final outputs of the DOM algorithm, the multiplication can be made composable in the presence of glitches [58]. We now give algorithms for the initial sharing of a secret value and a dth-order (1,0,1)-robust SNI masked multiplication.

3.2.1 Initial Sharing

The algorithm to share a secret variable in a field $\mathbb{GF}(2^m)$ using Boolean masking is described in Algorithm 1 and is used by all Boolean masking schemes.

Algorithm 1 Sharing value x against d^{th}-order SCA using Boolean masking

Input: secret value x
Output: $d+1$ shares $(x_1, ..., x_{d+1})$ such that $x = \sum_{i=1}^{d+1} x_i$

```
x_1 ← x
for i = 2 to d + 1 do
    r_i ← rand(GF(2^m))
    x_1 ← x_1 + r_i
    x_i ← r_i
end for
```

3.2.2 Masked Multiplication

In order to multiply two secret variables securely, the variables are first independently shared and then provided as input to a masked multiplication algorithm. The masked multiplication described in Algorithm 2 is due to Faust et al. [58] and is d-SNI secure.

A very important requirement is that the inputs of the multiplication need to be shared independently. The following extreme example illustrates the problem: assume a first-order sharing of a masked multiplication (or AND-gate) with shared inputs (x_1, x_2) and (y_1, y_2) which necessarily calculates the terms x_1y_2 and x_2y_1. If these sharings are dependent, for the sake of the example say $\mathbf{x} = \mathbf{y}$, the term $x_1y_2 = x_1x_2$ obviously leaks x as the non-completeness is broken.

Algorithm 2 Multiplying two independently shared values x and y against d^{th}-order SCA using Boolean, TI-like masking

Input: $d+1$ shares $(x_1, ..., x_{d+1})$ of x and $(y_1, ..., y_{d+1})$ of y
Output: $d+1$ shares $(z_1, ..., z_{d+1})$ such that $z = \sum_{i=1}^{d+1} z_i = xy$
  for $i = 1$ **to** d **do**
    for $j = i+1$ **to** $d+1$ **do**
      $r_{i,j} \leftarrow \texttt{rand}(\mathbb{GF}(2^m))$
      $u_{j,i} \leftarrow [x_j y_i + r_{i,j}]_{reg}$
      $u_{i,j} \leftarrow [x_i y_j + r_{i,j}]_{reg}$
    end for
  end for
  for $i = 1$ **to** $d+1$ **do**
    $z_i \leftarrow [x_i y_i + \sum_{j=1, j\neq i}^{d+1} u_{i,j}]_{reg}$
  end for

3.2.3 A Second-Order Secure Example

For a second-order secure ($d = 2$) multiplication we share each input value into $d+1 = 3$ shares. A masked multiplication $z = xy$ is performed using the following independent sharings of x and y.

$$x_{1,Bool} = x + r_{x,1} + r_{x,2} \qquad y_{1,Bool} = y + r_{y,1} + r_{y,2}$$

$$x_{2,Bool} = r_{x,1} \qquad y_{2,Bool} = r_{y,1}$$

$$x_{3,Bool} = r_{x,2} \qquad y_{3,Bool} = r_{y,2}$$

The shared output value of z are obtained as follows, where the $[\ .\ ]_{reg}$ brackets indicate synchronization of its argument in registers before proceeding with remaining operations. Note that all registered values satisfy the non-completeness property, which is quintessential for the SCA security.

$$z_{1,Bool} = \Big[[x_1 y_1]_{reg} + [x_1 y_2 + r_{1,2}]_{reg} + [x_1 y_3 + r_{1,3}]_{reg}\Big]_{reg}$$

$$z_{2,Bool} = \Big[[x_2 y_2]_{reg} + [x_2 y_1 + r_{1,2}]_{reg} + [x_2 y_3 + r_{2,3}]_{reg}\Big]_{reg}$$

$$z_{3,Bool} = \Big[[x_3 y_3]_{reg} + [x_3 y_1 + r_{1,3}]_{reg} + [x_3 y_2 + r_{2,3}]_{reg}\Big]_{reg}$$

3.3 Inner-Product Masking

The Inner-Product (IP) masking scheme was introduced by Faust and Dziembowski [52] and later studied in practical scenarios by Balasch et al. [9,11]. It is based on the inner product operation $x = \langle \boldsymbol{l}, \boldsymbol{r} \rangle$ that returns a scalar from a multiplication of two vectors. The secret is encoded such that it is the result of the inner product between a constant vector (the recombination vector) and a vector of shares. Recently this masking scheme was further optimized by Balasch et al. [10] by drawing similarities with the ISW multiplication. This led to new algorithms that offer the d-SNI security property, and forms the scheme we describe below.

3.3.1 Initial Sharing

IP masking requires $d+1$ (publicly known) values to form the constant vector or recombination vector. These $d+1$ public coefficients $l_1, ..., l_{d+1}$ are chosen such that all of them are non-zero, and the first coefficient is one $l_1 = 1$. Note that Boolean masking can be seen as IP masking with all public coefficients set to unity. The masked representation $\boldsymbol{x}$ of a value x is obtained by following Algorithm 3.

Algorithm 3 Sharing value x against d^{th}-order attacks using IP masking

Input: secret value x, non-zero public coefficients $l_1 = 1$, l_2, ..., l_{d+1}
Output: $d+1$ shares (x_1, ..., x_{d+1}) such that $x = \langle \boldsymbol{l}, \boldsymbol{x} \rangle$

$x_1 \leftarrow x$
for $i = 2$ **to** $d+1$ **do**
  $r_i \leftarrow \texttt{rand}(\mathbb{GF}(2^m))$
  $x_1 \leftarrow x_1 + l_i r_i$
  $x_i \leftarrow r_i$
end for

3.3.2 Masked Multiplication

Multiplying two secret variables securely can be achieved by following the masked multiplication algorithm described in Algorithm 4 [10]. The two inputs should be shared independently.

Algorithm 4 Multiplying two independently shared values x and y against d^{th}-order SCA using IP masking

Input: $d+1$ shares $(x_1, ..., x_{d+1})$ of x and $(y_1, ..., y_{d+1})$ of y, and the public coefficients $(l_1, ..., l_{d+1})$
Output: $d+1$ shares $(z_1, ..., z_{d+1})$ such that $z = xy = \langle \boldsymbol{l}, \boldsymbol{z} \rangle$
  for $i = 1$ **to** d **do**
    for $j = i+1$ **to** $d+1$ **do**
      $r_{i,j} \leftarrow \texttt{rand}(\mathbb{GF}(2^m))$
      $u_{j,i} \leftarrow [l_i x_j y_i + l_j^{-1} r_{i,j}]_{reg}$
      $u_{i,j} \leftarrow [l_j x_i y_j + l_i^{-1} r_{i,j}]_{reg}$
    end for
  end for
  for $i = 1$ **to** $d+1$ **do**
    $z_i \leftarrow [l_i x_i y_i + \sum_{j=1, j\neq i}^{d+1} u_{i,j}]_{reg}$
  end for

3.3.3 A Second-Order Secure Example

A second-order secure masked multiplication $z = xy$ acts on the following independent sharings of x and y.

$$x_{1,IP} = x + l_2 r_{x,1} + l_3 r_{x,2} \qquad y_{1,IP} = y + l_2 r_{y,1} + l_3 r_{y,2}$$

$$x_{2,IP} = r_{x,1} \qquad y_{2,IP} = r_{y,1}$$

$$x_{3,IP} = r_{x,2} \qquad y_{3,IP} = r_{y,2}$$

The shared output value of z is obtained as follows.

$$z_{1,IP} = \Big[[l_1 x_1 y_1]_{reg} + [l_2 x_1 y_2 + l_1^{-1} r_{1,2}]_{reg} + [l_3 x_1 y_3 + l_1^{-1} r_{1,3}]_{reg} \Big]_{reg}$$

$$z_{2,IP} = \Big[[l_2 x_2 y_2]_{reg} + [l_1 x_2 y_1 + l_2^{-1} r_{1,2}]_{reg} + [l_3 x_2 y_3 + l_2^{-1} r_{2,3}]_{reg} \Big]_{reg}$$

$$z_{3,IP} = \Big[[l_3 x_3 y_3]_{reg} + [l_1 x_3 y_1 + l_3^{-1} r_{1,3}]_{reg} + [l_2 x_3 y_2 + l_3^{-1} r_{2,3}]_{reg} \Big]_{reg}$$

3.4 Polynomial Masking

The research on Polynomial masking has gone through following stages. Goubin et al. first proposed Shamir's secret sharing [132] for masking applications

in [67]. Shamir's secret sharing scheme [132] is used to share a sensitive value $z \in \mathbb{GF}(2^m)$ among $n < 2^m$ players such that $d+1$ players are required for reconstruction. For this purpose, a dealer generates a degree-d polynomial $P_z(X) \in \mathbb{GF}(2^m)[X]$ with constant term z and secret, random coefficients r_i:

$$P_z(X) = z + \sum_{i=1}^{d} r_i X^i$$

when working in the field $\mathbb{GF}(2^m)$. This polynomial is evaluated in n distinct, non-zero public coefficients $\alpha_1, ..., \alpha_n \in \mathbb{GF}(2^m)$ and each resulting value $z_i = P_z(\alpha_i)$ is distributed to player i. To reconstruct the secret z, the first row $(\lambda_1, ..., \lambda_n)$ of the inverse of the $(n \times n)$ Vandermonde matrix $(\alpha_i^j)_{1 \leq i,j \leq n}$ is used as:

$$z = \sum_{i=1}^{n} \lambda_i z_i$$

The sharing of the Polynomial masking scheme was shown more resistant compared to a Boolean sharing at a given order due to the increased algebraic complexity of the scheme in the plausible Hamming Weight leakage model. Later, an improved algorithm for multiplication was proposed by Coron in [38], after a flaw was found in the multiplication algorithm of [67]. Targeting hardware, a glitch resistant countermeasure relying on Polynomial masking was introduction by Prouff and Roche [121]. An implementation of the AES S-box with this scheme was shown by Moradi et al. [99]. More recently, Grosso et al. introduced a tweak to reduce the cost of the polynomial masking scheme [73]. The Roche & Prouff masking scheme uses the BGW protocol [18] by Ben-Or, Widgerson and Goldwasser for secure multiplication in the masked domain.

The Polynomial masking scheme we describe below is an original contribution of this thesis. Instead of relying on the BGW protocol we provide an algorithm for masked multiplication that is more similar to the ISW method. As a result, the same order of SCA security can be achieved with $d+1$ shares instead of $2d+1$ shares.

3.4.1 Initial Sharing

The Polynomial masking scheme requires additional public coefficients to share a value. The public coefficients $\alpha_{1 \leq i \leq d+1}$ are chosen to be non-zero and distinct values. The conditions on the public coefficients lead to an upper bound of $2^m - 1$ on the number of shares for a given field $\mathbb{GF}(2^m)$, hence Polynomial masking in $\mathbb{GF}(2)$ (or on single bits) is not possible. The masked representation of a value is obtained by following Algorithm 5.

Algorithm 5 Sharing value x against d^{th}-order attacks using Polynomial masking

Input: secret value x, non-zero and distinct public coefficients α_1, ..., α_{d+1}
Output: $d+1$ shares $(x_1, ..., x_{d+1})$ such that $x = \sum_{i=1}^{d+1} \lambda_i x_i$ where $\lambda_i = \prod_{k,k\neq i} \frac{-\alpha_k}{\alpha_i - \alpha_k}$

```
for i = 1 to d + 1 do
    x_i ← x
end for
for i = 1 to d do
    r_i ← rand(GF(2^m))
    for j = 1 to d + 1 do
        x_j ← x_j + α_j^i r_i
    end for
end for
```

3.4.2 Masked Multiplication

Securely multiplying two independently shared secret variables is achieved by following the masked multiplication algorithm described in Algorithm 6. In contrast to the BGW protocol, which is used in the Prouff and Roche masking scheme [121], this algorithm reduces the required number of shares from $2d+1$ to $d+1$ by drawing inspiration from the ISW multiplication.

Algorithm 6 Multiplying two independently shared values x and y against d^{th}-order SCA using Polynomial masking

Input: $d+1$ shares $(x_1, ..., x_{d+1})$ of x and $d+1$ shares $(y_1, ..., y_{d+1})$ of y, and the public coefficients $(\lambda_1, ..., \lambda_{d+1})$ computed from $(\alpha_1, ..., \alpha_{d+1})$
Output: $d+1$ shares $(z_1, ..., z_{d+1})$ such that $z = xy = \sum_{i=1}^{d+1} \lambda_i Z_i$

```
for i = 1 to d do
    for j = i + 1 to d + 1 do
        r_{i,j} ← rand(GF(2^m))
        u_{j,i} ← [λ_i X_j Y_i + λ_j^{-1} r_{i,j}]_reg
        u_{i,j} ← [λ_j X_i Y_j + λ_i^{-1} r_{i,j}]_reg
    end for
end for
for i = 1 to d + 1 do
    z_i ← [λ_i x_i y_i + Σ_{j=1,j≠i}^{d+1} u_{i,j}]_reg
end for
```

3.4.3 A Second-Order Secure Example

A second-order secure masked multiplication $z = xy$ operates on the following independent sharings of x and y.

$$x_{1,Poly} = x + \alpha_1 r_{x,1} + \alpha_1^2 r_{x,2} \qquad y_{1,Poly} = y + \alpha_1 r_{y,1} + \alpha_1^2 r_{y,2}$$

$$x_{2,Poly} = x + \alpha_2 r_{x,1} + \alpha_2^2 r_{x,2} \qquad y_{2,Poly} = y + \alpha_2 r_{y,1} + \alpha_2^2 r_{y,2}$$

$$x_{3,Poly} = x + \alpha_3 r_{x,1} + \alpha_3^2 r_{x,2} \qquad y_{3,Poly} = y + \alpha_3 r_{y,1} + \alpha_3^2 r_{y,2}$$

$$z_{1,Poly} = \Big[[\lambda_1 x_1 y_1]_{reg} + [\lambda_2 x_1 y_2 + \lambda_1^{-1} r_{1,2}]_{reg} + [\lambda_3 x_1 y_3 + \lambda_1^{-1} r_{1,3}]_{reg}\Big]_{reg}$$

$$z_{2,Poly} = \Big[[\lambda_2 x_2 y_2]_{reg} + [\lambda_1 x_2 y_1 + \lambda_2^{-1} r_{1,2}]_{reg} + [\lambda_3 x_2 y_3 + \lambda_2^{-1} r_{2,3}]_{reg}\Big]_{reg}$$

$$z_{3,Poly} = \Big[[\lambda_3 x_3 y_3]_{reg} + [\lambda_1 x_3 y_1 + \lambda_3^{-1} r_{1,3}]_{reg} + [\lambda_2 x_3 y_2 + \lambda_3^{-1} r_{2,3}]_{reg}\Big]_{reg}$$

To reconstruct the secret z from its shares $z_{i,Poly}$, the first row $(\lambda_1, ..., \lambda_{d+1})$ of the inverse of the $(d+1 \times d+1)$ Vandermonde matrix $(\alpha_i^j)_{1 \le i,j \le d+1}$ is used in a linear combination:

$$z = \sum_{i=1}^{d+1} \lambda_i z_i = \sum_{i=1}^{d+1} z_i \prod_{k=1, k \neq i}^{d+1} -\alpha_k (\alpha_i - \alpha_k)^{-1}$$

The λ_i coefficients that form the recombination vector are public and can be computed in advance. In our $d = 2$ example the reconstruction is expressed by:

$$\lambda_1 = \frac{\alpha_2}{\alpha_1 + \alpha_2} \frac{\alpha_3}{\alpha_1 + \alpha_3}$$

$$\lambda_2 = \frac{\alpha_1}{\alpha_2 + \alpha_1} \frac{\alpha_3}{\alpha_2 + \alpha_3}$$

$$\lambda_3 = \frac{\alpha_1}{\alpha_3 + \alpha_1} \frac{\alpha_2}{\alpha_3 + \alpha_2}$$

$$Z = \lambda_1 z_{1,Poly} + \lambda_2 z_{2,Poly} + \lambda_3 z_{3,Poly}$$

3.5 Extracting A Generalized Structure

Upon closer inspection of the masked multiplications a similar structure emerges. Figures 3.1, 3.2 and 3.3 show first-order secure multiplications with Boolean, TI-like, Inner-Product and Polynomial masking respectively. All are centered around a middle layer of d^2 registers. After that register, the three schemes show the exact same structure: a compression of d^2 shares back to d shares and a final register stage. Before the middle register, we distinguish between two stages: the first stage computes all cross products (possibly with extra public coefficients), and the second stage adds fresh randomness to specific cross products. All values in these stages are sourced from registers rather than combinational logic. In the IP and Polynomial masking schemes, these fresh random values are multiplied with an expression of the public coefficients such that they cancel out when unshared, i.e. such that the correctness is not violated.

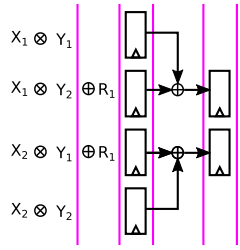

Figure 3.1: Multiplication with Boolean, TI-like Masking

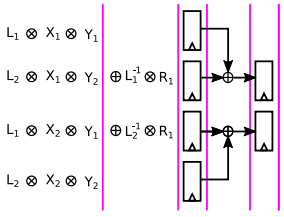

Figure 3.2: Multiplication with IP Masking

We formalize this structure in a slight adaptation of the CMS structure of Reparaz et al. [117].

Correctness layer $\mathcal{C}$. This layer is composed of all the linear and nonlinear terms ($l_j x_i y_j$ for the multiplication example) of the shared function, and hence responsible for the *correctness* of the sharing. A requirement is

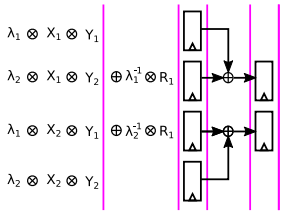

Figure 3.3: Multiplication with Polynomial Masking

that this layer must see uniformly shared and independent inputs. This layer requires *non-completeness*, the essence of TI. If the number of input shares is limited to $d+1$, the non-completeness implies the use of only one share of each unmasked input in each component function.

Mask Refreshing layer $\mathcal{MR}$. The multivariate security of a dth-order masking scheme depends on the proper insertion of additional randomness to break dependency between intermediates appearing in different clock cycles.

Internal Synchronization layer $\mathcal{IS}$. In a circuit with non-ideal gates, this layer ensures that non-completeness is satisfied in between nonlinear operations. It is depicted by the middle layer of d^2 registers in Figures 3.1, 3.2 and 3.3.

Compression layer $\mathcal{C}$. This layer is used to reduce the number of shares synchronized in $\mathcal{IS}$. It is especially required when the number of shares after $\mathcal{IS}$ is different from the number of input shares of $\mathcal{C}$.

Output Synchronization layer $\mathcal{OS}$. Similar to $\mathcal{IS}$, this layer is essential for the composability of the algorithm.

Regarding the relations between the three schemes, it is easy to verify that Boolean, TI-like masking can be obtained by setting all public values to one $l_1 = ... = l_{d+1} = 1$. This translates to both the initial sharing and the masked multiplication. Similarly, the masked multiplication with Inner-Product masking also encompasses the multiplication in Polynomial masking: the $l_1, ..., l_{d+1}$ just have to be chosen distinct and non-zero (assuming l_1 can be chosen freely and different from 1). We will leverage these relations in the next section where we will prove the composability in the dth-order robust SNI model of all three masked multiplication algorithms in one go. It is not clear how the initial sharing for IP and Polynomial masking are related and this remains to be investigated in future work. Additionally, future work can draw inspiration from other masking schemes, e.g. affine masking or (orthogonal) Direct-Sum masking, which has been initiated by Poussier et al. [111]. Another open question (that we are addressing in ongoing work) is whether the masking schemes differ in their level

of security at a given order. Lastly, the dominating cost with increasing order originates from the internal synchronization layers $\mathcal{IS}$, which entail d^2 registers for a multiplication. Finding ways to trade off this cost will prove interesting and relevant for practical applications. This trade-off has been investigated for software implementations by Belaïd et al. [17].

3.6 Security Proofs

We now prove the composable security of the masked multiplication algorithms in the dth-order (1,0,1)-robust Strong Non-Interference model. We recall that in order to achieve this, we require that an adversary can simulate d_1 internal probes and d_2 output probes with only d_1 shares of each input when $d_1 + d_2 \leq d$. The number of input shares d_1 needed for the simulation is thus independent from the number of probes d_2 on the output shares. We construct a perfect simulator that uses at most d_1 shares of each of the multiplier's shared inputs x_1, ..., x_{d+1} and y_1, ..., y_{d+1} to simulate the $d_1 + d_2 = d$ probes of the adversary.

We denote by $p_1, ..., p_d$ the probed values and construct the following groups to classify the observations of the adversary on the intermediate computations.

- **Group 1.** Probes on the values left of the Internal Synchronization layer $p_{i,j} = (x_i, y_j, r_{i,j}, \lambda_j, \lambda_i^{-1})$ with $1 \leq i, j \leq d + 1$
- **Group 2.** Probes on the values right of the Internal Synchronization layer $p_i = (u_{i,1}, ..., u_{i,d+1})$ with $1 \leq i \leq d + 1$

The $u_{i,j}$ values indicate the registered intermediate values in the masked multiplication algorithms.

Two sets of indices I and J are defined such that $|I| \leq d_1$ and $|J| \leq d_1$. Now given only the knowledge of $x_{i \in I}$ and $y_{i \in J}$, it should be possible to simulate the values of the d probes perfectly.

The sets I and J are constructed as follows.

- Initially I and J are empty.
- For every probe from $p_{i,j}$ add i to I and j to J.
- For every probe from p_i add i to I and to J (because $u_{i,i}$ depends on x_i and y_i).

As the adversary is only allowed to make at most d_1 internal observations, it holds that $|I| \leq d_1$ and $|J| \leq d_1$. After the simulator assigns a random value to every $r_{i,j}$, we show that any probe can be simulated with $x_{i \in I}$ and $y_{i \in J}$. To this end, we consider probes in group 1, group 2 and the output probes separately.

For a probed value $p_{i,j}$ of group 1 we distinguish two cases.

- If $i = j$, the simulator can perfectly simulate $p_{i,i}$ using x_i and y_i.
- If $j \in I$ and $i \in J$, then by definition the adversary has probed $p_{j,i}$. The simulator can perfectly simulate the probe with x_j, y_i and the random $r_{i,j}$.

In case the probe p_i is from group 2, we have by definition that $i \in I, J$ and the simulator can perfectly compute the ith-component ($u_{i,i} = \lambda_i x_i y_i$) of the probe using x_i and y_i. For each of the remaining j-components of p_i the following cases are distinguished.

- If $j \in J$ and $j \notin I$, then the adversary has previously probed $p_{i,j}$, which can be simulated with x_i, y_j and $r_{i,j}$ to be used as the jth component of the probe.
- If $j \in J$ and $j \in I$, then the adversary has previously probed either p_j or $p_{j,i}$. In both cases $r_{i,j}$ can be used with x_i and y_j to simulate the jth component of the probe.
- If $j \notin J$, the simulator assigns a random and independent value to the jth component of p_i.

The proof is concluded by showing how to simulate a probe on one of the output values. We need to consider the following two cases.

- If the attacker has observed intermediate wires, the partial sums previously probed are already simulated. For the remaining terms, we note that by definition of the scheme there always exists a random bit $r_{k,l}$ that is not used in the computation of the internal probes. Therefore the simulator can assign the output probe to a random value.
- If the attacker has exclusively observed output shares, he can at most have probed d of them. By definition the output shares are composed using d random values. There exists a random bit $r_{k,l}$ that contributes to one output probe that does not appear in the computation of any other observed output. The simulator can thus assign a random value to that output probe, which completes the proof. □

3.7 Variations for Trade-Offs, Offset Remasking and Non-Complete Compression

We will now propose two variations on the masked multiplication schemes in an attempt to trade off required randomness, circuit complexity and latency.

First-order (1,0,1)-robust SNI Multiplication with one register layer. We have previously seen that a 1st-order (1,0,1)-robust SNI multiplication can be computed with two register layers and has the following form.

$$z_1 = \Big[[l_1 x_1 y_1]_{reg} + [l_2 x_1 y_2 + l_1^{-1} r_{1,2}]_{reg}\Big]_{reg}$$

$$z_2 = \Big[[l_2 x_2 y_2]_{reg} + [l_1 x_2 y_1 + l_2^{-1} r_{1,2}]_{reg}\Big]_{reg}$$

We can increase the randomness to allow a 1st-order (1,0,1)-robust SNI multiplication to work with one register layer.

$$z_1 = [l_1 x_1 y_1 + l_1^{-1} r_1]_{reg} + [l_2 x_1 y_2 + l_1^{-1} r_2]_{reg}$$

$$z_2 = [l_1 x_2 y_1 + l_2^{-1} r_3]_{reg} + [l_2 x_2 y_2 + l_2^{-1} r_1 + l_2^{-1} r_2 + l_2^{-1} r_3]_{reg}$$

Its 1st-order (1,0,1)-robust SNI security is easily tracked down through a small enumeration of cases.

- **Case 1.** $d_1 = 1$, $d_2 = 0$ ($d_1 + d_2 \leq 1$). Every intermediate $l_j x_i y_j$ can be simulated with $d_1 = 1$ share of each input.
- **Case 2.** $d_1 = 0$, $d_2 = 1$ ($d_1 + d_2 \leq 1$). Every probe can be simulated with $d_1 = 0$ share of each input, due to the randomness that is assigned before the internal synchronization.

Second-order (1,0,1)-robust SNI Multiplication with one register layer. A 1st-order (1,0,1)-robust SNI multiplication can be computed with two register

layers as follows.

$$z_1 = \Big[[l_1x_1y_1]_{reg} + [l_2x_1y_2 + l_1^{-1}r_{1,2}]_{reg} + [l_3x_1y_3 + l_1^{-1}r_{1,3}]_{reg}\Big]_{reg}$$

$$z_2 = \Big[[l_2x_2y_2]_{reg} + [l_1x_2y_1 + l_2^{-1}r_{1,2}]_{reg} + [l_3x_2y_3 + l_2^{-1}r_{2,3}]_{reg}\Big]_{reg}$$

$$z_3 = \Big[[l_3x_3y_3]_{reg} + [l_1x_3y_1 + l_3^{-1}r_{1,3}]_{reg} + [l_2x_3y_2 + l_3^{-1}r_{2,3}]_{reg}\Big]_{reg}$$

By again increasing the randomness, we can reduce its circuit complexity by using only a single layer of registers instead of two. The notation of the randomness is slightly adapted for clarity. Its construction for $d = 2$ relies on a so-called ring refreshing where the two rings of fresh masks are offset with a single index.

$$\begin{aligned}
z_1 = {} & [l_1x_1y_1 + l_1^{-1}r_1 + l_1^{-1}r_9]_{reg} + \\
& [l_2x_1y_2 + l_1^{-1}r_2 + l_1^{-1}r_1]_{reg} + \\
& [l_3x_1y_3 + l_1^{-1}r_3 + l_1^{-1}r_2]_{reg} \\
z_2 = {} & [l_1x_2y_1 + l_2^{-1}r_4 + l_2^{-1}r_3]_{reg} + \\
& [l_2x_2y_2 + l_2^{-1}r_5 + l_2^{-1}r_4]_{reg} + \\
& [l_3x_2y_3 + l_2^{-1}r_6 + l_2^{-1}r_5]_{reg} \\
z_3 = {} & [l_1x_3y_1 + l_3^{-1}r_7 + l_3^{-1}r_6]_{reg} + \\
& [l_2x_3y_2 + l_3^{-1}r_8 + l_3^{-1}r_7]_{reg} + \\
& [l_3x_3y_3 + l_3^{-1}r_9 + l_3^{-1}r_8]_{reg}
\end{aligned}$$

We prove the resulting 2nd-order (1,0,1)-robust SNI security through a case study.

- **Case 1.** $d_1 = 2$, $d_2 = 0$ ($d_1 + d_2 \leq 2$). Every intermediate can be simulated with $d_1 = 2$ shares of each input x and y.
- **Case 2.** $d_1 = 1$, $d_2 = 1$ ($d_1 + d_2 \leq 2$). Every probe can be simulated with $d_1 = 1$ share of each input, due to the randomness that is assigned before the internal synchronization.

- **Case 3.** $d_1 = 0$, $d_2 = 2$ ($d_1 + d_2 \leq 2$). Every probe can be simulated with $d_1 = 0$ shares of each input, due to the randomness that is assigned before the internal synchronization.

Higher-order Multiplication with one register layer using Offset Remasking. We need to be careful when generalizing this construction to higher orders. We show what can go wrong in a 3rd-order multiplication. As we now have $d+1 = 4$ shares, we only give a limited set of terms before and after the internal synchronization.

$$
\begin{aligned}
z_1 = {} & [l_1 x_1 y_1 + l_1^{-1} r_1 + l_1^{-1} r_{16}]_{reg} + \\
& [l_2 x_1 y_2 + l_1^{-1} r_2 + l_1^{-1} r_1]_{reg} + \\
& [l_3 x_1 y_3 + l_1^{-1} r_3 + l_1^{-1} r_2]_{reg} + \\
& [l_4 x_1 y_4 + l_1^{-1} r_4 + l_1^{-1} r_3]_{reg} \\
& \ldots \\
z_4 = {} & [l_1 x_4 y_1 + l_4^{-1} r_{13} + l_4^{-1} r_{12}]_{reg} + \\
& [l_2 x_4 y_2 + l_4^{-1} r_{14} + l_4^{-1} r_{13}]_{reg} + \\
& [l_3 x_4 y_3 + l_4^{-1} r_{15} + l_4^{-1} r_{14}]_{reg} + \\
& [l_4 x_4 y_4 + l_4^{-1} r_{16} + l_4^{-1} r_{15}]_{reg}
\end{aligned}
$$

Using the d-probing model we can show that this straightforward extension to a higher orders is not secure. The output value of z_1 can be written as $z_1 = x_1 y + l_1^{-1} r_{16} + l_1^{-1} r_4$. With two extra probes, one on the last contribution of z_1 ($u_{1,4}$) to obtain $l_1^{-1} r_4$ and one on the last contribution of z_4 ($u_{4,4}$) to retrieve $l_1^{-1} r_{16}$, we can recover information on the sensitive value y. With 3 probes we can thus break the supposedly 3rd-order secure multiplication.

The correct way to add the randomness is with so-called offset remasking. It differs from ring refreshing in the index the two rings of fresh masks are offset with. This construction was independently presented by Barthe et al. [13]. If we shift the rings with an offset of 2 in our 3rd-order masked multiplier we can restore the SCA security.

$$
\begin{aligned}
z_1 = [l_1 x_1 y_1 + l_1^{-1} r_1 + l_1^{-1} r_{15}]_{reg} \\
[l_2 x_1 y_2 + l_1^{-1} r_2 + l_1^{-1} r_{16}]_{reg} \\
[l_3 x_1 y_3 + l_1^{-1} r_3 + l_1^{-1} r_1]_{reg} \\
[l_4 x_1 y_4 + l_1^{-1} r_4 + l_1^{-1} r_2]_{reg} \\
\ldots \\
z_4 = [l_1 x_4 y_1 + l_4^{-1} r_{13} + l_4^{-1} r_{11}]_{reg} \\
[l_2 x_4 y_2 + l_4^{-1} r_{14} + l_4^{-1} r_{12}]_{reg} \\
[l_3 x_4 y_3 + l_4^{-1} r_{15} + l_4^{-1} r_{13}]_{reg} \\
[l_4 x_4 y_4 + l_4^{-1} r_{16} + l_4^{-1} r_{14}]_{reg}
\end{aligned}
$$

Due to the extra random terms in the output value of $z_1 = x_1 y + l_1^{-1} r_{15} + l_1^{-1} r_{16} + l_1^{-1} r_3 + l_1^{-1} r_4$, an attacker can not break this 3rd-order multiplication with 3 or less probes.

Higher-order Multiplication with one register layer using Non-Complete Compression. Another way to achieve security in the higher-order multiplication example is with so-called non-complete compression. Instead of altering the offset in the remasking, it reorders the terms in the compression part of the algorithm with non-completeness in mind. We can bring 1st-order non-completeness in the compression layer while keeping correctness by slightly altering the multiplications with the l_i factors. The resulting outputs look as

follows.

$$
\begin{aligned}
z_1 = &[l_1 x_1 y_1 + l_1^{-1} r_1 + l_1^{-1} r_{16}]_{reg} \\
&[l_2 x_1 y_2 + l_1^{-1} r_2 + l_1^{-1} r_1]_{reg} \\
&[l_3 x_1 y_3 + l_1^{-1} r_3 + l_1^{-1} r_2]_{reg} \\
&[x_2 y_1 + l_1^{-1} r_5 + l_1^{-1} r_4]_{reg} \\
&\ldots \\
z_4 = &[l_1 x_4 y_1 + l_4^{-1} r_{13} + l_4^{-1} r_{12}]_{reg} \\
&[l_2 x_4 y_2 + l_4^{-1} r_{14} + l_4^{-1} r_{13}]_{reg} \\
&[l_4 x_4 y_4 + l_4^{-1} r_{16} + l_4^{-1} r_{15}]_{reg} \\
&[x_1 y_4 + l_4^{-1} r_4 + l_4^{-1} r_3]_{reg}
\end{aligned}
$$

An attacker can again not break this 3rd-order multiplication with 3 probes or less due to the introduced non-completeness in the output shares and the resulting extra scrambling of random masks. Rigorous proofs of the offset remasking and non-complete compression are part of our ongoing work.

3.8 Conclusion

In this chapter, we related Boolean, TI-like masking, Inner-Product masking and Polynomial masking. We drew inspiration from the ISW multiplication to reduce the cost of the multiplication in Polynomial masking, which is commonly implemented with the more costly BGW protocol. A similar optimization was recently applied to multiplication in the Inner-Product masking scheme [10]. While the relations between the masking schemes we point out are simple, they lead to new insights in the relation between masking schemes. This work builds further on the consolidating work initiated by Reparaz et al. in [117].

We initiated the chapter by surveying known adversary models and we chose the hardware implementation aware robust probing model to argue about the security of the algorithms we proposed. After relating Boolean, TI-like masking, Inner-Product masking and Polynomial masking, we extracted a generalized structure for masking schemes and explored trade-offs in implementation cost.

Throughout this chapter we discussed several open problems. We summarize them here as directions for future work.

1. **Security Level.** A first question is which masking schemes offer a higher level of SCA resistance. Additionally, how does the choice of public coefficients influence the SCA resistance? Answering these questions from the perspective of a real world adversary, i.e. by mounting actual key retrieval experiments, is advantageous for practical purposes.

   A second question is what benefits can be obtained from the differences in initial sharing. Inner-Product masking and Polynomial masking share the same masked multiplication algorithm, but their initial sharings differ. Can this difference lead to advantages, or can it be leveraged when considering an active adversary? The BGW protocol offers advantages in that light.

2. **Further Generalizing Masking Schemes.** Can other masking schemes, e.g. affine masking or (orthogonal) Direct-Sum masking, be included in a generalized structure for hardware? What benefits can be extracted from them and translated to other schemes?

3. **Optimizing the Implementation Cost.** Can the masked multiplications be reduced in circuit complexity, and more specifically, can the heavy d^2 cost of the Internal Synchronization Layer be alleviated? What trade-offs would lead to this reduction?

In the next chapter, we illustrate how the most compact of masking schemes, the Boolean, TI-like masking scheme can be applied to AES to achieve small SCA resistant AES implementations. Their low circuit complexities are partially achieved by using the single register layer 1st-order and 2nd-order (1,0,1)-robust SNI multiplications we presented in this chapter.

Chapter 4

Securing AES with Boolean, TI-Like Masking

In this chapter, we take some of the Boolean, TI-like masking schemes described in the previous chapter on masking theory and apply them to one of the most widely deployed block ciphers, namely the Advanced Encryption Standard. This way we can investigate various trade-offs designers can make when considering Boolean, TI-like masking schemes to achieve side-channel security in hardware implementations. We evaluate the side-channel security through leakage detection tests rather than investigating the success rate or the ease of mounting key-retrieval attacks.

This chapter is structured as follows. We first provide an overview of the unmasked AES implementation we use as basis for our masked implementations in Section 4.1. We proceed by presenting several side-channel secure AES realizations in Section 4.2: one first-order implementation using the theoretical minimum number of $s_{in} = d + 1 = 2$ input shares, one second-order implementation with the minimum number $s_{in} = d + 1 = 3$ of input shares, and a second-order secure implementation with $s_{in} = s_{out} = 6$ shares. We evaluate the side-channel security of these implementations using leakage detection tests on a SAKURA-G side-channel analysis evaluation board and report on the results in Section 4.3. In Section 4.4, we present and discuss the implementation cost. We finally conclude with a brief summary of our findings and provide directions for future work in Section 4.5.

This work is based on publications presented at the Smart Card Research and Advanced Application Conference (CARDIS 2015) [42] and the Conference on

Cryptographic Hardware and Embedded Systems (CHES 2016) [47]. It includes extensions drawn from developments made by the community during the course of this PhD, particularly from the publication by Arribas et al. [5].

Before we proceed, we stress that the results of the leakage detection tests found here differ significantly from the ones reported in the CHES 2016 publication [47]. The figures obtained for the CHES publication were the results of a long series of heuristic variations in the implementation style of the HDL, in the synthesis parameters, and a great deal of other random factors. We stopped this experimentation once we reached our desired result, i.e. leakage free t-tests, and only proceeded with investigating the effect of some of these parameters later on. This positive bias is a classic mistake a researcher can make and to rectify this, we repeated the measurements here with a clear documentation of our practices.

4.1 An Unprotected Implementation of AES

In what follows we describe a compact implementation of AES in hardware and focus specifically on the data path. We additionally give details on the S-box, as it is the most complex and critical block for a compact and side-channel secure implementation.

4.1.1 A Very Compact Hardware Implementation

The basic principles of the compact AES architecture are described here for one round. For the full details we refer to the original publication [103]. In this discussion, we assume the computation of one S-box takes 6 clock cycles, which is the case in all three implementations we present. The hardware diagrams of the State and Key Arrays are depicted in Figure 4.1.

Clock Cycles 1-16. In each of the first sixteen clock cycles, one byte is sent to the S-box. The first output of the S-box is received in the sixth clock cycle and clocked back in the State Array register S_{33}. From that clock cycle on, we receive back one byte and store it in register S_{33}, while the other registers operate in shift register mode. At clock cycle sixteen, the S-box lookup of byte 11 is ready.

During these first sixteen clock cycles, the roundkey is loaded in the Key Array. When the bytes are shifted out, the Sel signal is active during all but the 4th,

8th, 12th and 16th clock cycles. After the 16th clock cycle, the *Sel* signal is deactivated until the next round.

Clock Cycles 17-20. The following four clock cycles load bytes from the Key Array registers to the S-box. In the Key Array, no registers are clocked in the 17th clock cycle. During clock cycles 18 and 19, the registers K_{13}, K_{23} and K_{33} are not clocked.

Clock Cycle 21. From this point on, all bytes required for the SubBytes operation have been sent to the S-box. The S-box now receives bytes of value zero as input.

Clock Cycles 22. The S-box operation on the last byte of the State Array is finished and its value is stored in S_{33}. Because of the shift register mode that was active, all bytes are in the correct place to activate the alternate mode for the ShiftRows operation, which is performed in this clock cycle.

Meanwhile, the first S-box lookup of the Key byte is finished. The round constant *rcon* is XORed to the S-box output before it is stored in register K_{30}.

Clock Cycles 23-26. The last four clock cycles of the AES round perform the linear MixColumns operation. The third mode of operation is activated in the State Array for this purpose.

During the cycles 23 to 25, the S-box computation of the Key bytes are finished and XORed to the corresponding K_{00} byte. The registers are not clocked in the 26th clock cycle. For the next round, the value in K_{00} is XORed with S_{00} before going to the S-box input.

4.1.2 Canright's Very Compact AES S-box

The AES S-box is an 8-bit permutation composed of a multiplicative inversion in $\mathbb{GF}(2^8)$ followed by a $\mathbb{GF}(2)$-affine transformation [40]. Side-channel resistant implementations of this S-box are commonly based on subfield arithmetic as proposed by Rijmen [119] and explored by Canright [33]. This approach typically produces low-area circuits. It takes its name from the recursive decomposition of the S-box into computations in smaller fields. Namely, the $\mathbb{GF}(2^8)$ inversion is first decomposed into arithmetic operations in $\mathbb{GF}(2^4)$, and in turn the nonlinear operations are performed in the subfield $\mathbb{GF}(2^2)$. The resulting computation is

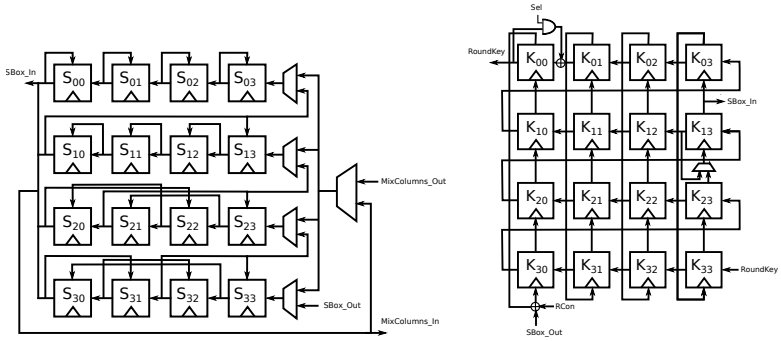

Figure 4.1: State and Key Array

composed of a $\mathbb{GF}(2)$-linear (LM) and inverse $\mathbb{GF}(2)$-linear map (ILM), several $\mathbb{GF}(2^2)$ multiplications, bitwise XORs and multiple instantiations of linear operations in $\mathbb{GF}(2^2)$. The diagram for the unprotected S-box is shown in Figure 4.2. For a detailed description of the individual operations, we refer to the original work [33].

4.2 Masking AES at Different Orders

In this section we first explain why linear operations are easy to mask. Nonlinear operations require a more subtle approach for the masking scheme to preserve its dth-order security, which justifies our major focus on masking the S-box. We then have a look at various ways to decompose the Canright S-box, and explain how we decide upon a structure that is advantageous for achieving a low circuit complexity (our main goal). Our three implementations are described

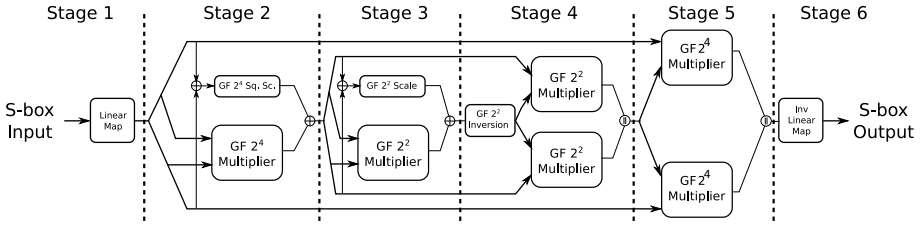

Figure 4.2: Operations in the Canright AES Sbox. The lines depict pipeline registers to keep the non-completeness in masked scenarios

afterwards: the first-order implementation with $s_{in} = 2 = d + 1$ shares, the second-order implementation with $s_{in} = 3 = d + 1$ shares and the second-order implementation with $s_{in} = 6 > td + 1$ shares.

4.2.1 Linear Operations

With Boolean masking, it is easy to see that securely evaluating linear and affine functions is straightforward. The function $f(x, y) = ax + by + c$ with a, b and c constants in $\mathbb{GF}(2^m)$ and variables $x, y \in \mathbb{GF}(2^m)$ can be implemented in a dth-order secure way as follows.

$$f_1 = ax_1 + by_1 + c$$

$$f_i = ax_i + by_i \qquad , 2 \leq i \leq d + 1$$

The correctness of these component functions is easily checked as $f(x, y) = \bigoplus_{i=1}^{d+1} f_i = a \bigoplus_{i=1}^{d+1} x_i + b \bigoplus_{i=1}^{d+1} y_i + c = ax + by + c$.

The masking of the linear components of AES such as ShiftRows, MixColumns and AddRoundKey are achieved by instantiating $d + 1$ state and key arrays. Each pair of state and key array is responsible for one single share of the plaintext and key.

4.2.2 Redefining the S-box Decomposition

Converting the Canright S-box to a threshold implementation can be achieved on several levels. Each individual block can be composed with the neighbouring blocks or decomposed into smaller sub-blocks to attain different trade-offs between circuit complexity, speed and randomness. We acknowledge that randomness requirements also (indirectly) affect the circuit complexity. Hence, we strive for an implementation with low circuit complexity, and at the same time we try to keep the randomness requirements as low as possible.

For the discussion of our shared AES S-box, we rely on Figure 4.2. We choose to implement the square scale and multiplication operations in $\mathbb{GF}(2^4)$ as done by Bilgin et al. [22]. This adaptation requires less randomness and clock cycles than sharing their subfield functions in $\mathbb{GF}(2^2)$ since some of the refreshing and registering that must follow the nonlinear operation is avoided. The inversion in $\mathbb{GF}(2^4)$ is of algebraic degree three. This would lead to $(d + 1)^3$ output shares and fresh random masks which blows up the circuit complexity and randomness consumption. Additionally, for our implementation with $s_{in} = s_{out} = 6$ shares, no compact second-order non-complete sharing of such a function has yet

been proposed. In all three implementations, we therefore share its subfield decomposition, which consists of a linear operation and three multiplications in $\mathbb{GF}(2^2)$. Although this increases the number of clock cycles by one, it keeps the area and randomness contained.

We split the calculation of the S-box into six stages. All stages are separated by pipeline registers, indicated by the vertical lines in Figure 4.2. The registers between the stages keep the non-completeness within each stage satisfied.

4.2.3 Implementation 1: First-Order TI of the AES S-box with $d + 1 = 2$ Shares

In what follows, we describe in detail how the AES is masked with 2 shares using TI to achieve first-order security. The same principle applies to higher orders, but care is required when applying the refreshing and compression layer [117] (see Chapter 3 for more details). We will discuss the higher-order refreshing in more detail when describing the second-order implementations.

We now go over the masked design in a stage by stage manner.

First Stage. The first operation occurring in the decomposed S-box performs a change of basis through a linear map. Its masking requires instantiating this linear map once for each share i. This mapping is implemented in combinational logic and it maps the 8-bit input $(x_i^1, \ldots, x_i^8)$ to the 8-bit output $(y_i^1, \ldots, y_i^8)$ for each share i as follows:

$$y_i^1 = x_i^8 \oplus x_i^7 \oplus x_i^6 \oplus x_i^3 \oplus x_i^2 \oplus x_i^1 \qquad y_i^5 = x_i^8 \oplus x_i^5 \oplus x_i^4 \oplus x_i^2 \oplus x_i^1$$

$$y_i^2 = x_i^7 \oplus x_i^6 \oplus x_i^5 \oplus x_i^1 \qquad y_i^6 = x_i^1$$

$$y_i^3 = x_i^7 \oplus x_i^6 \oplus x_i^2 \oplus x_i^1 \qquad y_i^7 = x_i^7 \oplus x_i^6 \oplus x_i^1$$

$$y_i^4 = x_i^8 \oplus x_i^7 \oplus x_i^6 \oplus x_i^1 \qquad y_i^8 = x_i^7 \oplus x_i^4 \oplus x_i^3 \oplus x_i^2 \oplus x_i^1$$

Note that synchronizing the output values of the first stage with registers is required for security. For simplicity, we explain what can go wrong in the absence of these registers for the first-order case, but the same can be expressed for any order d. Let's consider the y^2 and y^6 bits of the output of the linear map. The two shares corresponding to those bits are then given by (y_1^2, y_2^2) and (y_1^6, y_2^6) respectively. These two bits will go through the AND gates of the subsequent $\mathbb{GF}(2^4)$ multiplier, which leads to the following term being computed

at one point:

$$y_1^2 y_2^6 = (x_1^7 + x_1^6 + x_1^5 + x_1^1)x_2^1$$

If there is no register between the linear map and the $\mathbb{GF}(2^4)$ multiplier, the above expression is realized by combinational logic, which deals with x_1^1 and x_2^1 in a nonlinear way and causes leakage on $\mathbf{x}^1 = (x_1^1, x_2^1)$. Note that the problem mentioned above does not happen in TIs with $s_{in} = td + 1$ shares, since the original non-completeness condition imposes that each component function is independent of at least one share of *each* input (for $d = 1$). Hence, linear functions before and after nonlinear component functions can be used without synchronization. No remasking is required after this stage since the computed function is linear.

Second Stage. We consider the parallel application of the nonlinear multiplication and the affine Square Scaling (Sq. Sc.) as one single function $\mathbf{z} = \mathbf{x} \otimes \mathbf{y} \oplus SqSc(\mathbf{x} \oplus \mathbf{y})$. For the first order, the resulting equations are given by:

$$\mathbf{z}_1 = \mathbf{x}_1 \otimes \mathbf{y}_1 \oplus SqSc(\mathbf{x}_1 \oplus \mathbf{y}_1)$$

$$\mathbf{z}_2 = \mathbf{x}_1 \otimes \mathbf{y}_2$$

$$\mathbf{z}_3 = \mathbf{x}_2 \otimes \mathbf{y}_1$$

$$\mathbf{z}_4 = \mathbf{x}_2 \otimes \mathbf{y}_2 \oplus SqSc(\mathbf{x}_2 \oplus \mathbf{y}_2)$$

Where the Square Scaling operation $y = SqSc(x)$ is given by:

$$y^1 = (x^1 \oplus x^3) \qquad y^3 = (x^1 \oplus x^2)$$

$$y^2 = (x^2 \oplus x^4) \qquad y^4 = x^1$$

The multiplication between shares $(z^1, z^2, z^3, z^4) = (x^1, x^2, x^3, x^4) \otimes (y^1, y^2, y^3, y^4)$ is performed as follows.

$$z^1 = x^1y^1 \oplus x^3y^1 \oplus x^4y^1 \oplus x^2y^2 \oplus x^3y^2 \oplus x^1y^3 \oplus x^2y^3 \oplus x^3y^3 \oplus x^4y^3 \oplus x^1y^4 \oplus x^3y^4$$

$$z^2 = x^2y^1 \oplus x^3y^1 \oplus x^1y^2 \oplus x^2y^2 \oplus x^4y^2 \oplus x^1y^3 \oplus x^3y^3 \oplus x^2y^4 \oplus x^4y^4$$

$$z^3 = x^1y^1 \oplus x^2y^1 \oplus x^3y^1 \oplus x^4y^1 \oplus x^1y^2 \oplus x^3y^2 \oplus x^1y^3 \oplus x^2y^3 \oplus x^3y^3 \oplus x^1y^4 \oplus x^4y^4$$

$$z^4 = x^1y^1 \oplus x^3y^1 \oplus x^2y^2 \oplus x^4y^2 \oplus x^1y^3 \oplus x^4y^3 \oplus x^2y^4 \oplus x^3y^4 \oplus x^4y^4$$

It is important to add the affine contribution from the Square Scaling to the multiplier output in such a way that the non-completeness property is not broken, which leaves only one possibility for the construction. In previous works [22, 23, 103] these two functions are treated separately, leading to more outputs at this stage. By approaching the operations in the second stage as one, we obtain two advantages. Firstly, we omit the extra registers for storing the outputs of both sub-functions separately. Secondly, less randomness is required to achieve uniformity for the inputs of the next stage.

Before the new values are clocked in the register, we need to perform a mask refreshing to make the next stage's inputs uniform. To this end we add a sharing of zero to the outputs which keeps the correctness property and requires $3 \times 4 = 12$ bits of randomness.

Third Stage. This stage is similar to the second stage. Here, the received nibbles are split in 2-bit couples for further operation. The Scaling operation (Sc) replaces the similar affine Square Scaling and is executed alongside the multiplication in $\mathbb{GF}(2^2)$. By combining both operations, we can share the total function by taking again the non-completeness into account. Since the nonlinear multiplication outputs four 2-bit values, a total of 6 bits of randomness is consumed for remasking.

The scaling operation $y = Sc(x)$ is given by:

$$y^1 = x^1$$

$$y^2 = x^1 \oplus x^2$$

The multiplication between two elements $(z^1, z^2) = (x^1, x^2) \otimes (y^1, y^2)$ is performed as follows.

$$z^1 = (x^1 \oplus x^2)(y^1 \oplus y^2) \oplus x^1 y^1$$

$$z^2 = (x^1 \oplus x^2)(y^1 \oplus y^2) \oplus x^2 y^2$$

Fourth Stage. The fourth stage is composed of an inversion and two parallel multiplications in $\mathbb{GF}(2^2)$. The inversion in $\mathbb{GF}(2^2)$ is linear and is implemented by swapping the bits using wires and comes at no additional cost. The outputs of the multiplications are concatenated, denoted by ⓒ in Figure 4.4, to form 4-bit values in $\mathbb{GF}(2^4)$. The concatenated 4-bit values of the 4 outputs of the multipliers are remasked with a total of 12 fresh random bits.

Fifth Stage. Stage 5 is similar to Stage 4. The difference of the two stages lies in the absence of the inversion operation and the multiplications being performed in $\mathbb{GF}(2^4)$ instead of $\mathbb{GF}(2^2)$. The concatenation of its outputs results in byte values, which are remasked with 24 fresh random bits.

Sixth Stage. In the final stage of the S-box, the inverse linear map is performed. By using a register between Stage 5 and Stage 6, we can remask the shares and perform a compression before the inverse linear map is performed resulting in only two instead of four inverse linear map instances. As with the linear map, no uniform sharing of its inputs is required for security. However, in the full AES, this output will at some point contribute to a new input of the S-box, where it undergoes nonlinear operations again. We insert the remasking for this reason.

An output y of the inverse linear map given input x is computed as:

$$y^1 = x^6 \oplus x^4 \qquad y^5 = x^8 \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^4$$

$$y^2 = x^8 \oplus x^4 \qquad y^6 = x^7 \oplus x^6 \oplus x^4 \oplus x^3 \oplus x^1$$

$$y^3 = x^7 \oplus x^1 \qquad y^7 = x^6 \oplus x^5 \oplus x^2$$

$$y^4 = x^8 \oplus x^6 \oplus x^4 \qquad y^8 = x^7 \oplus x^5 \oplus x^2$$

First-Order Mask Remasking. The structure of the additive remasking we use is shown on the left in Figure 4.3. We need 3 units of randomness per remasking. We recall that one unit of randomness is defined as a set of independent and uniformly distributed bits with the field size of the wire as its cardinality.

4.2.4 Implementation 2: Second-Order TI of the AES S-box with $d + 1 = 3$ Shares

To achieve a very compact second-order TI, we can use the same the stage-wise structure as the first-order implementation and increase the number of input shares s_{in} from 2 to 3. We will have to increase the number of consumed random units from 3 to 9, and will apply them in a ring (shown on the right in Figure 4.3). The reason for this increase is twofold: we need to make the inputs to the next stage uniform, and we need to provide multivariate security.

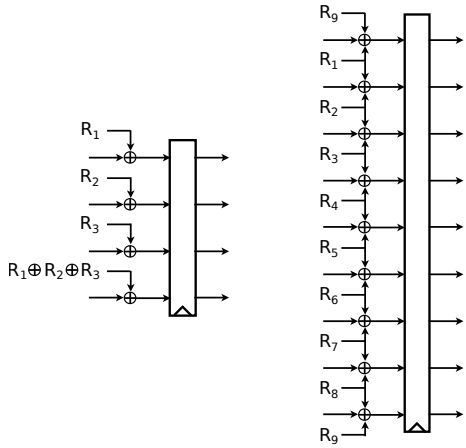

Figure 4.3: Additive remasking for the first-order implementation (left), ring remasking for the second-order implementations (right)

We start with the previous construction but with an increased number of shares: $s_{in} = 3$ instead of $s_{in} = 2$. The main change that occurs, appart from the mask refreshing structure is found in the Stages 2 and 3.

Parallel operations. The parallel linear and nonlinear operations from Stage 2 and Stage 3 are altered in the following way. For the second order, the resulting equations are given bellow. It is clear that non-completeness is respected.

$$\mathbf{z}_1 = \mathbf{x}_1 \otimes \mathbf{y}_1 \oplus SqSc(\mathbf{x}_1 \oplus \mathbf{y}_1) \qquad \mathbf{z}_6 = \mathbf{x}_2 \otimes \mathbf{y}_3$$

$$\mathbf{z}_2 = \mathbf{x}_1 \otimes \mathbf{y}_2 \qquad \mathbf{z}_7 = \mathbf{x}_3 \otimes \mathbf{y}_1$$

$$\mathbf{z}_3 = \mathbf{x}_1 \otimes \mathbf{y}_3 \qquad \mathbf{z}_8 = \mathbf{x}_3 \otimes \mathbf{y}_2$$

$$\mathbf{z}_4 = \mathbf{x}_2 \otimes \mathbf{y}_1 \qquad \mathbf{z}_9 = \mathbf{x}_3 \otimes \mathbf{y}_3 \oplus SqSc(\mathbf{x}_3 \oplus \mathbf{y}_3)$$

$$\mathbf{z}_5 = \mathbf{x}_2 \otimes \mathbf{y}_2 \oplus SqSc(\mathbf{x}_2 \oplus \mathbf{y}_2)$$

The complete masked S-box is depicted in Figure 4.4.

4.2.5 Implementation 3: Second-Order TI of the AES S-box with $6 > td + 1$ Shares

We create another second-order TI, but this time we use the "classic" number of shares for a TI. This number is related to the algebraic degree t of the function we want to mask and is commonly referred to as a $td+1$ implementation. There are three known implementations that use $s_{in} \geq d + 1$. A first sharing uses $s_{in} = 5$ and $s_{out} = 10$ and was used in a second-order implementation of the KATAN block cipher by Bilgin et al. [21]. A second sharing employs $s_{in} = 6$ and $s_{out} = 7$ and was used by De Cnudde et al. to implement a second-order AES S-box [42]. More recently, a third sharing was proposed by Arribas et al. which reduces the number of output shares to obtain $s_{in} = 6$ and $s_{out} = 6$ [5]. We use the latter sharing as it results in the lowest number of random masks when using our ring refreshing on the output shares.

$(6, 6)$-**Sharing.** The $s_{in} = s_{out} = 6$ sharing is given below.

$$z_1 = x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus x_1y_3 \oplus x_3y_1$$

$$z_2 = x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus x_2y_4 \oplus x_4y_2 \oplus x_3y_4 \oplus x_4y_3$$

$$z_3 = x_3y_3 \oplus x_3y_5 \oplus x_5y_3 \oplus x_3y_6 \oplus x_6y_3$$

$$z_4 = x_4y_4 \oplus x_1y_4 \oplus x_4y_1 \oplus x_1y_6 \oplus x_6y_1 \oplus x_4y_6 \oplus x_6y_4$$

$$z_5 = x_5y_5 \oplus x_1y_5 \oplus x_5y_1 \oplus x_4y_5 \oplus x_5y_4$$

$$z_6 = x_6y_6 \oplus x_2y_5 \oplus x_5y_2 \oplus x_2y_6 \oplus x_6y_2 \oplus x_5y_6 \oplus x_6y_5$$

The implementation follows the same guidelines as the previous implementations.

4.3 Leakage Detection

Once our implementations are coded, we can proceed with their synthesis and load them on our Spartan-6 FPGA for evaluation. The results of their evaluation are described in what follows. The measurement setup we used is described in Chapter 2.

4.3.1 Implementation 1: (2,4)-Sharing

PRNG Off. The result of the leakage detection tests for our first-order AES with biased masks is shown in Figure 4.5. With 5k traces, the t-value goes beyond the confidence interval of ±4.5 and we can conclude that our measurement setup is sound.

PRNG On. The result of the leakage detection test on our first-order AES with the activated PRNG is shown in Figure 4.5. Clear leakage is present in the second-order t-test (we get a t-score of up to 300). Moreover, a small number of peaks fall outside the confidence interval in the first-order. Our first-order AES implementation does not achieve the targeted first-order security with 50 million traces, even though this is expected by the theory.

4.3.2 Implementation 2: (3,9)-Sharing

PRNG Off. The result of the leakage detection tests for our (3,9)-sharing second-order AES with biased masks is shown in Figure 4.6. As expected, the t-value goes beyond the confidence interval of ±4.5 with 5k traces and we proceed with our evaluation.

PRNG On. The result of the leakage detection test on our second-order (3,9)-sharing AES with the activated PRNG is shown in Figure 4.6. We would expect clear leakage in the third-order t-test, but the noise in the signal leads to a leakage free third-order statistical moment with 50M traces. In the first-, and second-order t-test we expect no leakage but get points that fall outside the confidence interval nonetheless. Our second-order AES implementation with the (3,9)-sharing does not achieve the targeted first-, and second-order security with 50M traces. It remains to be clarified whether or not the points of leakage indicate that our implementation can actually be attacked with a first-, or second-order key retrieval though.

4.3.3 Implementation 3: (6,6)-Sharing

PRNG Off. The result of the leakage detection tests for our (6,6)-sharing second-order AES with biased masks is shown in Figure 4.7. With 5M traces, the t-value goes beyond the confidence interval of ±4.5 and we can conclude that our measurement setup is sound.

PRNG On. The result of the leakage detection test on our second-order (6,6)-sharing AES with the activated PRNG is shown in Figure 4.7. As we expect, we have leakage in the third-order t-test, but it is rather small due to noise in the signal. In the first- and second-order t-test we expect no leakage but get points that fall outside the confidence interval in both orders. We conclude that our second-order AES implementation with the (6,6)-sharing does not achieve the targeted first- and second-order security with 50M traces, but again, it remains to be clarified how easy these leakages can be exploited.

4.4 Implementation Cost

Table 4.1 lists the area costs of the individual components of our designs. Table 4.2 gives the full implementation costs of our designs and of related TIs. The circuit complexity estimations are obtained with Synopsys 2010.03 and the NanGate 45nm Open Cell Library [81].

Table 4.1: Circuit complexity of different functions of the masked AES

	Circuit Complexity [GEs] Compile	Compile Ultra
First-order TI $s_{in} = 2$, $s_{out} = 4$		
S-box	1977	1872
AES Key & State Array	4472	4238
Total AES	6681	6340
Second-order TI $s_{in} = 3$, $s_{out} = 9$		
S-box	3796	3662
AES Key & State Array	6287	6258
Total AES	10449	10276
Second-order TI $s_{in} = 6$, $s_{out} = 6$		
S-box	10492	9381
AES Key & State Array	6411	6144
Total AES	17355	17087

Discussion

We now discuss the increase in implementation costs when going from first- to second-order security and compare the results with similar designs. This discussion does not necessarily apply to other ciphers or implementations. Lightweight block ciphers with small S-boxes for instance might benefit from keeping $s_{in} \geq td + 1$ in nonlinear functions.

Circuit Complexity. Moving from first-order to second-order security requires an increase of 50% in Gate Equivalents (GEs) for linear functions and an increase of around 100% for nonlinear functions. The larger increase for nonlinear functions stems from the quadratic increase of output shares as function of an increment in input shares, resulting in more registers per stage.

Speed. The number of clock cycles for one AES encryption is equal for our first- and second-order implementations. All previous first-order TIs have a faster encryption because they have less pipeline stages in the S-box. The number of stages in our S-box is smaller than the S-box by Gross et al. [71] due to the trade-off in randomness and clock cycles we presented in Chapter 3. We note that our second-order S-box with $s_{in} = 6$ shares can be executed in 4 clock cycles by merging the linear map and inverse linear map to their adjacent stages. Such a merging is used in the S-box of Bilgin et al. [22] that uses $s_{in} > td + 1$ shares.

Randomness. Our first-order AES requires 54 bits of randomness per S-box execution. For our second-order $d + 1$ implementation, this number increases to 162 bits of randomness. This number is reduced to 132 bits per clock cycle in our (6,6)-sharing second-order AES, as the number of output shares to refresh is $s_{out} = 6$ as opposed to $s_{out} = 9$ in the $d + 1 = 3$ case. These 132 bits consist of 3×8 bits for increasing the number of shares from 3 at the state and key array to 6 at the S-box input, and 108 bits of randomness for the actual S-box.

The randomness requirement of our first-order AES is higher than the AES implementations by Bilgin et al. [23]. The reason is that in our minimal sharing, we have no correction terms that can help regain the uniformity of the output shares. For the second-order implementation, even more randomness is required per output share to achieve bivariate security. All shares of one stage require randomness for both satisfying the uniformity and for statistical independence of its following stage. By trading off more stages in the S-box for randomness, the S-box of Gross et al. [71] achieves a significantly lower number of consumed randomness.

Table 4.2: Implementation cost of different TIs of AES

AES	Circuit Complexity AES [GE]*	S-box [GE]*	S-box Stages	Randomness† [bit]	Clock Cycles
Unprotected					
[103]	2601/2421	233	1	-	226
1st-order					
[103]	11114/11031	- /4244	5	48	266
[22]	9102/8172	3708/3004	4	44	246
[23]	11221/10167	3653/2949	4	44	246
[23]	8119/7282	2835/2224	4	32	246
[71]	7.1k/-	2.6k/-	8	18	246
Impl. 1	6681/6340	1977/1872	6	54	276
2nd-order					
[42]‡	18602/14872	11174/7849	6	126	276
[71]	11.9k/-	5.3k/-	8	54	246
Impl. 2	10449/10276	3796/3662	6	162	276
Impl. 3	17355/17087	10492/9381	6	132	276

*: Using `compile` / `compile_ultra`[1] option.

†: Per S-box lookup

‡: Area estimation of a non-tested AES with tested S-box

4.5 Conclusion

In this chapter we described three hardware implementations of AES secure against differential power analysis attacks at various orders. We presented two implementations using the theoretical minimum number of shares, one with first-order security and one with second-order security. Additionally, we presented a second-order implementation using $s_{in} = s_{out} = 6$ inputs shares. The security of all designs were tested through leakage detection tests in lab conditions. We noticed that all our implementations leak unexpectedly, even though all security critical properties of the masking schemes were satisfied. It is conceivable to attribute these unexpected leakages to the foundational underlying assumption all masking schemes make, i.e. "the total leakage of all the shares is a linear combination of the leakages of all shares individually". We investigate this more thoroughly in the next chapter.

[1] The `compile_ultra` option requires careful application. To avoid optimizing over share boundaries, each submodule is compiled using `compile_ultra`. The resulting netlists are then given to a top module and synthesized with the regular `compile` option. This way, the gates from the ASIC library are instantiated conform to the `KEEP_HIERARCHY` option.

We conclude this chapter by providing directions for future work. The required randomness of our implementations is substantial. Investigating ways of reducing the randomness is essential for lightweight application. In future work, paths leading to minimizing this cost can be researched.

Another direction for future work is to compare the security in terms of number of traces required to perform a successful key retrieval. This can lead to better insights in the trade-off between security and implementation costs for masked implementations with $s_{in} = d + 1$ and $s_{in} \geq td + 1$ shares.

Different approaches to reach the smallest masked AES can be investigated. Ghoshal and De Cnudde investigated a different S-box architecture than the Canright S-box to this aim [64], namely the Boyar-Peralta S-box [30]. In recent work, a first-order side-channel secure implementation consuming no fresh randomness was obtained [139] by using this architecture in combination with a new approach to mask refreshing [39].

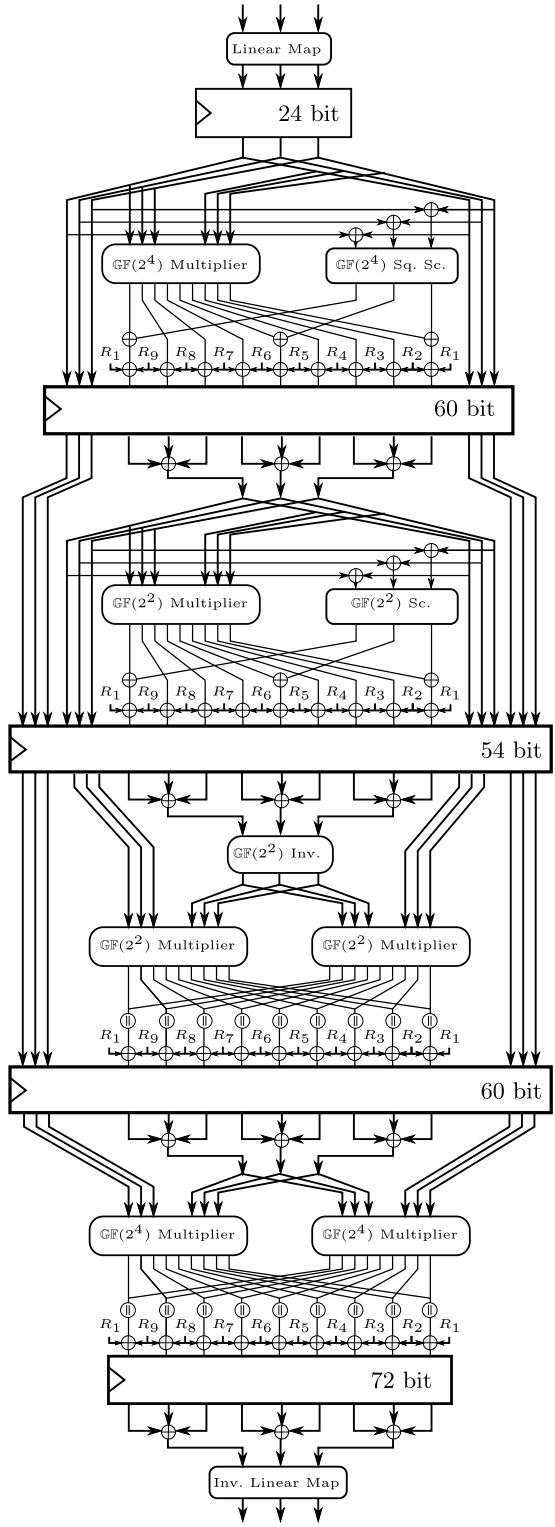

Figure 4.4: Structure of the second-order TI of the AES S-box

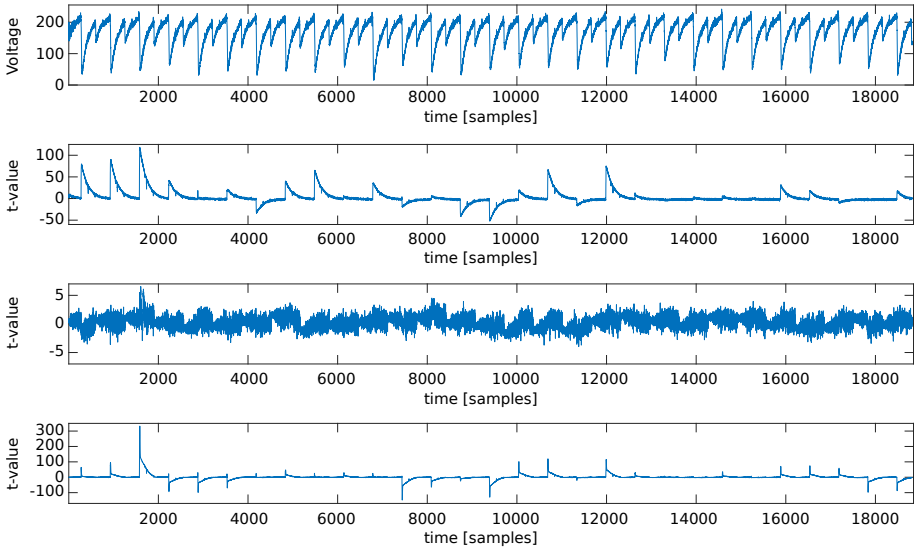

Figure 4.5: Masked AES with (2,4)-sharing, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 5k traces, first-order t-test with uniform masks using 50M traces, second-order t-test with uniform masks using 50M traces

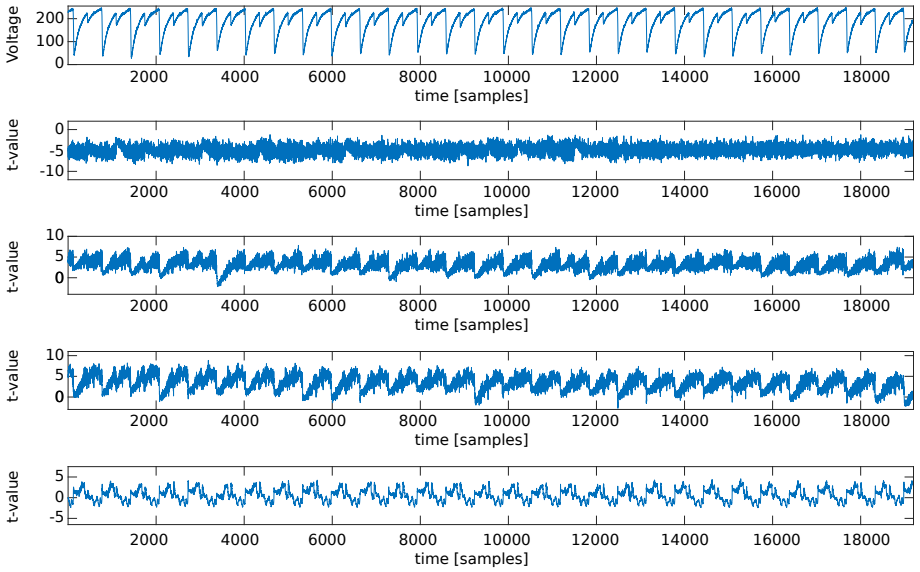

Figure 4.6: Masked AES with (3,9)-sharing, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 5k traces, first-order t-test with uniform masks using 50M traces, second-order t-test with uniform masks using 50M traces, third-order t-test with uniform masks using 50M traces

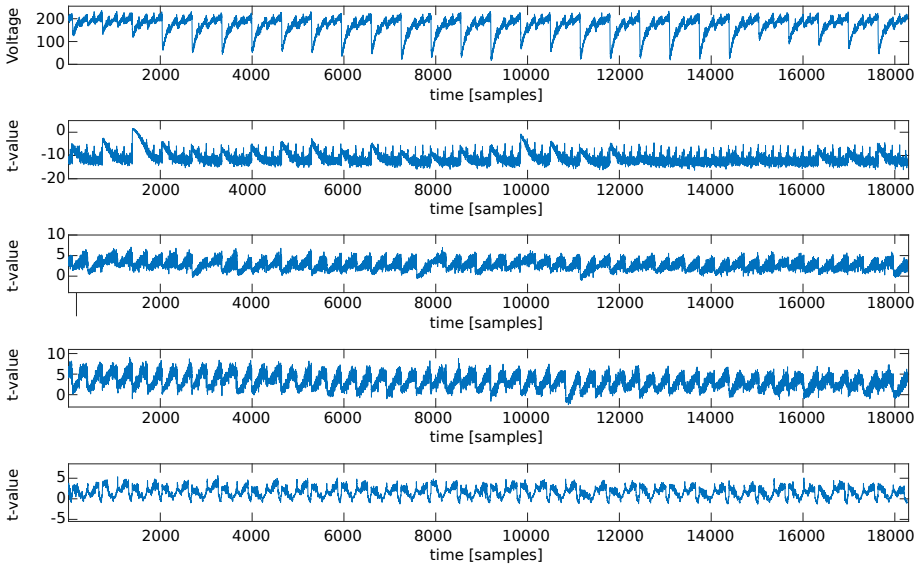

Figure 4.7: Masked AES with (6,6)-sharing, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 5M traces, first-order t-test with uniform masks using 50M traces, second-order t-test with uniform masks using 50M traces, third-order t-test with uniform masks using 50M traces

Chapter 5

Evidence of Leakage from Coupling in Masked Implementations

We have seen in previous chapters that masking schemes can exhibit exploitable leakage when their properties and assumptions do not translate correctly from the theory to the implementation. For the ISW masking scheme and the Trichina masked AND-gate, an important underlying assumption is that all gates execute sequentially. An essential assumption made by most masking schemes is that the total leakage of the device is a linear combination of the leakages of the individual shares. If this assumption is violated, a degradation of the security order can be expected, and one way this assumption can be nullified is through coupling.

In this chapter, we investigate the effect of coupling on the side-channel security through the physical placement of share domains. In order to differentiate other masking failures, we make sure all other requirements of the masking scheme hold. We use a Pseudorandom Number Generator (PRNG) for the initial shares and we make sure our share domains are isolated using the "Keep Hierarchy" option. We choose the lightweight KATAN-32 [32] as our target block cipher as we expect coupling effects to be more prominent in a low-switching-noise setting. We use leakage assessment on power measurements obtained from the Virtex-2 Pro VP7 FPGA of the SASEBO-G side-channel analysis evaluation board. We show that we can observe differences in—possibly—exploitable leakage by placing functions corresponding to different shares of a masked implementation

in close proximity.

In Section 5.1, we describe what can go wrong when out-of-model leakage is present and we give an overview of the internal mechanism of two out-of-model leakage sources. In Section 5.2 we describe how these mechanisms negatively affect the security in a first-order TI with $s_{in} = 3$ shares. We revisit the KATAN-32 threshold implementation in Section 5.3. In Section 5.4, we describe and evaluate two leakage scenarios on the KATAN-32 threshold implementation. We give a brief discussion in Section 5.5 before we conclude in Section 5.6.

The work presented in this chapter was presented at the International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2017) [41].

5.1 Sources of Out-of-Model Leakage

We now revisit what can go wrong when leakage assumptions of masking schemes fail. We also look at the conditions for masking from a power consumption point of view and give simplified models of physical phenomena that are known to lead to out-of-model leakage [90].

5.1.1 Failure of Independent Leakage

The theoretical security of masking schemes degrades when the leakage signals of different shares get influenced by each other nonlinearly. The amount of this security reduction has been investigated theoretically in [49] with respect to the strength of joint leakage in comparison to independent leakages (called the flaw constant) and to noise level. It has been shown that mutual information increases together with the flaw constant. It is also shown that second-order leakage can become easier to detect than first-order leakage as the noise increases given enough dependent leakage.

In practice, Hamming Distance (HD) leakage from one share to another and glitchy gates are natural and visible examples of non-independent leakage. It is shown in [12] that a theoretically d^{th}-order secure implementation can be attacked using $d/2^{\text{th}}$-order attack in practice due to HD leakage if the security proofs assume Hamming Weight (HW) leakage. Moreover, Boolean masking without non-completeness is shown to be futile in circuits using CMOS-like technology [92]. The temporally separated version of the masking scheme of Prouff and Roche [113], where shares interleave their computations, has also been argued to be vulnerable when static leakage is measurable [98].

Another example of non-independent leakage is crosstalk, which originates from coupling capacitors between circuit wires, and between circuit wires and ground. Only a few publications have investigated the effect of crosstalk within the field of SCA attacks. In [37], Chen et al. used SPICE simulations to show that the leakage intensity of glitches and the leakage caused by inter-wire capacitance are comparable. They then demonstrated a successful key retrieval attack on a masked implementation with dual-rail pre-charge logic. This logic style was thought to avoid non-independent leakages caused by glitching implying crosstalk to be the main leakage leading to the attack. However, the success of their attack is more likely to be attributed to the issue of early propagation in implementations using these logic styles [140] rather than crosstalk itself. Later, Dyrkolbotn considered the layout dependent phenomena of capacitive crosstalk in [50, 51] in order to derive a more precise leakage model. They showed that the detection performance of values on an 8-bit data bus increases from 2.5-bits of information per sample with a Hamming Distance detector to a theoretical 5.7-bit and simulated 5-bit of information per sample with a crosstalk based detector. Power-supply noise or IR drop, another coupling effect in circuits, was also shown to have a negative impact on the security of countermeasures by relating independent logic gates through the power supply line [125, 144]. Finally, Schmidt et al. performed successful key-retrieval attacks by measuring the power consumption on input or output peripherals instead of using the regular power supply lines [126]. The success of their method originates from the coupling between pins of an Integrated Circuit (IC).

To conclude, there is no definitive report on the observability of non-independent leakage originating from coupling on a real-world device when masking is considered. In order to distinguish between non-independent leakage originating from e.g. HD or glitches, and leakage originating from coupling, we will refer to the latter as out-of-model-leakage.

5.1.2 Power Consumption in Masking Schemes

From a power-consumption perspective, a first-order masked implementation requires the following condition to hold: the mean power consumption for each unmasked sensitive value should be equal. One way to achieve this requirement is by using Boolean masking with masks drawn from a uniform random distribution.

If we mask a one-bit secret value x with a one-bit random mask r as $\mathbf{x} = (s_1, s_2) = (x \oplus r, r)$ and denote the probability of $r = i$ by K_i, we can formalize

the condition for the uniformity of the masks as:

$$K_0 = K_1 = \frac{1}{2}$$

The expected power consumption P w.r.t. the unmasked value x can then be expressed as:

$$P(x = 0) = K_0 P(s_1 = 0, s_2 = 0) + K_1 P(s_1 = 1, s_2 = 1)$$

$$P(x = 1) = K_0 P(s_1 = 1, s_2 = 0) + K_1 P(s_1 = 0, s_2 = 1)$$

The condition for first-order Boolean masking is then formalized by the following equation:

$$P(s_1 = 0, s_2 = 0) + P(s_1 = 1, s_2 = 1) = P(s_1 = 0, s_2 = 1) + P(s_1 = 1, s_2 = 0)$$

In this example, first-order vulnerabilities occur in the masking scheme when this condition is violated. The effect of out-of-model leakage from coupling on the security of the masking scheme can be understood by analyzing the power consumption P. The instantaneous power consumption P_{inst} represents a sample of a SCA measurement trace:

$$P_{inst} = I_{inst} V_{inst}$$

Where I_{inst} and V_{inst} denote the instantaneous current and instantaneous voltage respectively.

5.1.3 Crosstalk

Crosstalk is the result of capacitive coupling between adjacent wires. Figure 5.1 shows two adjacent wires, each with a parasitic capacitance to the IC substrate and an inter-wire capacitance between them. When a wire (the aggressor) switches the value it carries, another wire in its vicinity (the victim) will be influenced through the inter-wire capacitance $C_{1,2}$ between the aggressor and the victim. This influence can range from an increased delay of a signal to traverse the wire, through a wrong value being temporarily induced on the victim. The reduction in SCA security introduced by crosstalk can be explained as follows. A typical first-order masked implementation represents a sensitive variable by two randomized shares, such that the mean power consumption of either share is independent of the other share. If two wires belonging to different shares are coupled, the mean power consumption of one share depends jointly on both a neighboring aggressor share and itself. The masked implementation may hence be rendered insecure.

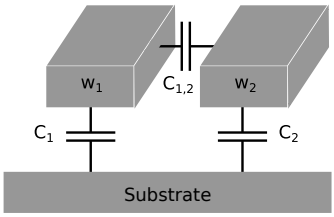

Figure 5.1: Crosstalk between two wires w_1 and w_2 originates from the inter-wire capacitance $C_{1,2}$

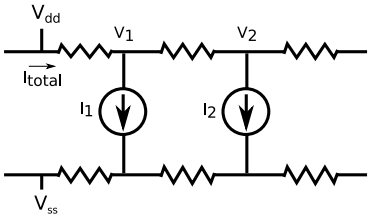

Figure 5.2: Static and dynamic IR drop occurs from the non-zero resistance of conductive supply voltage and ground wires

5.1.4 IR Drop

IR drop or power supply noise originates from the finite conductance of wires in the Power Distribution Network (PDN) of ICs. Every wire segment has a small resistance associated with it leading to a drop in the power supply voltage when a current flows through that wire [114]. Both static and dynamic IR drop can lead to coupling between shares and hence to out-of-model leakage. A simplified model is given in Figure 5.2.

The effects of both crosstalk and IR drop get worse with shrinking technology nodes [114]. At the time of writing, the effect of IR drop on the SCA security had not yet been investigated. More recently, this topic was researched by Schellenberg et al. [125].

5.2 Coupling in Threshold Implementations

We now look into the effects of the previously discussed coupling mechanisms on a first-order TI masking scheme with $s_{in} = 3$ shares.

5.2.1 Crosstalk

Since we are interested in the effect of out-of-model leakage on the first-order SCA security of the KATAN-32 TI with three shares, we first provide a discussion of the effect crosstalk can introduce in that setting.

Masking the secret value x yields $\mathbf{x} = (s_1, s_2, s_3) = (x \oplus r_1 \oplus r_2, r_1, r_2)$, where the masks r_i are drawn from a uniform random source to satisfy the uniformity condition of masking, i.e.

$$K_{0,0} = K_{0,1} = K_{1,0} = K_{1,1} = \frac{1}{4}$$

where $K_{i,j}$ denotes the probability of $r_1 = i$ and $r_2 = j$.

The masking condition $P(x = 0) = P(x = 1)$ on the expected power consumption P w.r.t. the unmasked value x is then expressed as:

$$\begin{aligned} &P(s_1 = 0, s_2 = 0, s_3 = 0) + P(s_1 = 0, s_2 = 1, s_3 = 1) \\ &+P(s_1 = 1, s_2 = 0, s_3 = 1) + P(s_1 = 1, s_2 = 1, s_3 = 0) \\ &= P(s_1 = 0, s_2 = 0, s_3 = 1) + P(s_1 = 0, s_2 = 1, s_3 = 0) \\ &+P(s_1 = 1, s_2 = 0, s_3 = 0) + P(s_1 = 1, s_2 = 1, s_3 = 1) \end{aligned}$$

In order to examine the influence of out-of-model leakage on the SCA security of the masking scheme, we need to find whether or not a dependence exists between the instantaneous power consumption $P_{inst} = I_{inst}V_{inst}$ and the unmasked value x. In order to perform this exemplary analysis, we rely on a data bus model [48]. The relation between the instantaneous power and the consumed energy can be derived from the expression for the energy needed to charge a wire i on a bus from $V_i(t^-) = 0$ to $V_i(t^+) = V_{dd}$:

$$E_{rise,i} = \int_{t^-}^{t^+} V_{dd} \cdot I_i(t)dt$$

The total energy consumption to change a three-wire bus can be written as:

$$E_{total} = \sum_{i=0}^{3}(1 + 2\lambda - \lambda\delta_{i,i-1} - \lambda\delta_{i,i+1}) \cdot C_L \cdot V_{dd} \cdot V_i$$

where $\lambda = C_I/C_L$, and the inter-wire capacitances and wire-substrate capacitances are chosen as $C_I = C_{1,2} = C_{1,3}$ and $C_L = C_1 = C_2 = C_3$

respectively. The equality of the inter-wire capacitances and of the wire-substrate capacitances are justified by the data bus model, where the wires are assumed to have the same dimensions and are assumed to be equidistant both from each other and from the substrate. Furthermore, $\delta_{i,j} \in \{-1, 0, 1\}$ is the normalized relative voltage change of the jth line w.r.t. the ith line, V_{dd} is the supply voltage and V_j is the final voltage on the jth line.

We can now group and calculate the total energy transitions per unmasked value using values from [97] for $C_L = 400fF$, $C_I = 250fF$ and $V_{dd} = 3V$:

$$E_{total,0\to0} = 0.430nJ \qquad E_{total,1\to0} = 0.529nJ$$

$$E_{total,0\to1} = 0.475nJ \qquad E_{total,1\to1} = 0.498nJ$$

The difference in total energy per unmasked value is analytically distinguishable and hence the masking scheme is not secure in the presence of crosstalk.

5.2.2 IR Drop

Power supply noise or IR drop is a result of the finite conductance of wires from the power delivery network in ICs. Figure 5.3 shows a simplified version of this effect that focuses on shared subcircuits [19]. The influence of IR drop on the security of the masking scheme is best understood by looking at the instantaneous power consumption $P_{inst} = I_{inst}V_{inst}$ on the voltage nodes V_1, V_2 and V_3:

$$V_1 = V_{dd} - (I_1 + I_2 + I_3)R_1$$

$$V_2 = V_{dd} - (I_1 + I_2 + I_3)R_1 - (I_2 + I_3)R_2$$

$$V_3 = V_{dd} - (I_1 + I_2 + I_3)R_1 - (I_2 + I_3)R_2 - I_3R_3$$

We can now write the instantaneous power consumption of all the shares $P_{inst,Share1}$, $P_{inst,Share2}$ and $P_{inst,Share3}$ as:

$$P_{inst,Share1} = I_1V_1 = V_{dd}I_1 - I_1^2R_1 - I_1I_2R_1 - I_1I_3R_1$$

$$P_{inst,Share2} = I_2V_2 = V_{dd}I_2 - I_1I_2R_1 - I_2^2R_1 - I_2I_3R_1 - I_2^2R_2 - I_2I_3R_2$$

$$P_{inst,Share3} = I_3V_3 = V_{dd}I_3 - I_1I_3R_1 - I_2I_3R_1 - I_3^2R_1 - I_2I_3R_2 - I_3^2R_2 - I_3^2R_3$$

The power consumption of any one share thus theoretically depends on adjacent shares and, in other words, the non-completeness property is violated. The masking scheme is hence not secure in the presence of IR drop.

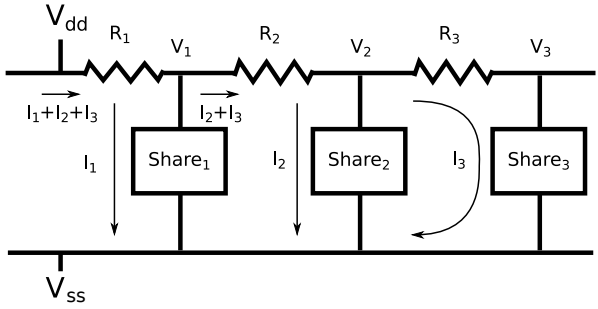

Figure 5.3: Power supply noise or IR drop in the PDN couples shares

5.3 KATAN-32 and Its Threshold Implementation

KATAN is a set of block ciphers designed specifically for lightweight applications [32]. Its efficiency in hardware translates to a small area and a low power consumption. Three options are available for the state size: 32-, 48- or 64-bit. All options use an 80-bit key, making the security independent of the state size.

The diagram of the KATAN-32 round function is shown in Figure 5.4. The 32-bit plaintext is stored in a state that consists of two shift registers: a 13-bit right shifting register $L1$ and a 19-bit left shifting register $L2$. The cipher processes the state by applying a round operation 254 times. The round operation consists of a small number of AND and XOR gates and is performed on several bits in order to update the first bits of the shift registers $L1$ and $L2$. The function is of the form $A = f(X, Y, Z) = X \oplus YZ$. The IR (irregular update) bit represents the last bit of the round counter which enables or disables the fourth bit of $L1$ in the round operation. The bits k_{2i} and k_{2i+1} are the $2i^{th}$ and $(2i + 1)^{th}$ bits of the 80-bit key for rounds $i \leq 40$. In rounds $i > 40$, they are derived from the original key by a Linear Feedback Shift Register (LFSR). The full description can be found in [32].

The round operation is susceptible to glitching, making TI a natural choice for a masked implementation. This was shown by Bilgin et al. in [21] where a first-, a second- and a third-order threshold implementation of KATAN-32 were presented. We now revisit their first-order TI of KATAN-32.

The focus of this description lies in the sharing of the nonlinear round function since sharing nonlinear operations is more involved than sharing linear ones. An unshared key and key schedule are used, such that the key addition only needs to be performed on one of the share of the state.

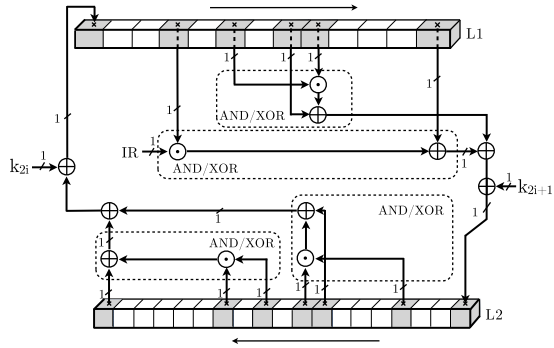

Figure 5.4: KATAN-32 consists of two sets of shift registers and four groups of nonlinear operations (Source: [21])

A first-order three-share TI of a single AND gate with uniform outputs can only be achieved with remasking [105]. To avoid using extra randomness, the AND gates of the round operations are always grouped with an XOR gate and masked using the uniform three-share TI of the function $A = f(X, Y, Z) = X \oplus YZ$. This approach results in the following first-order non-complete and uniform sharing:

$$a_1 = x_1 \oplus (y_1 z_1 \oplus y_1 z_2 \oplus y_2 z_1)$$

$$a_2 = x_2 \oplus (y_2 z_2 \oplus y_2 z_3 \oplus y_3 z_2)$$

$$a_3 = x_3 \oplus (y_3 z_3 \oplus y_3 z_1 \oplus y_1 z_3)$$

Since the round counter (and resultantly the irregular update IR) is not key dependent and hence not shared, IR is added to the AND/XOR blocks in the following way:

$$a_i = x_i + IR \times y_i, \quad i \leq s_{in}$$

The number of state shares is chosen to be three and follows the number of shares of the nonlinear function.

5.4 Coupling in a TI of KATAN-32 with 3 Shares

We now practically investigate what effect coupling has on the first-order side-channel leakage of the KATAN-32 threshold implementation with three shares.

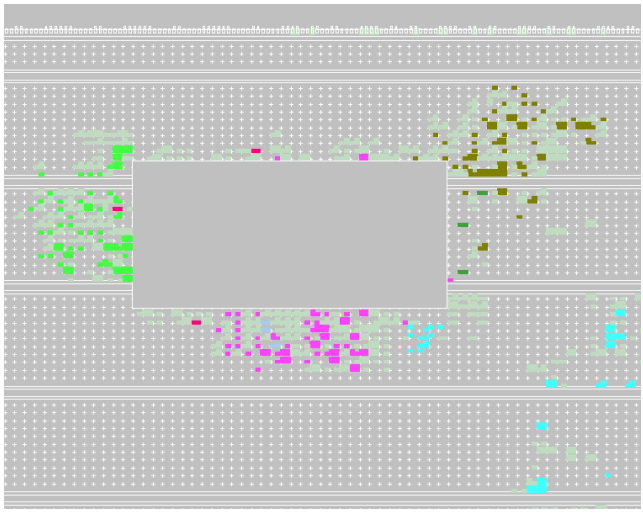

Figure 5.5: Placing the individual shares far apart leads to a secure design

In a first experiment, we measure the side-channel resistance of a regular threshold implementation of KATAN-32, for which we follow the design rules mentioned in the literature, i.e. setting the "Keep Hierarchy" constraint. In a second experiment, we show that placement has an influence on the leakage of the same (netlist-wise) KATAN-32 TI. We use the constraints described in the preliminaries (Chapter 2) to create two variants of the KATAN-32 TI: a regular implementation which we expect to be secure (Figure 5.5), and an implementation where the shares are forced to be placed close to one another (Figure 5.6).

5.4.1 Secure Threshold Implementation of KATAN-32

To achieve a secure Threshold Implementation, we set the "Keep Hierarchy" synthesis option to true globally, as is done in related practical TIs [22, 103, 110]. Resulting from the hierarchy in both the synthesis and place and route phases, the individual shares are automatically placed apart on the Virtex-2 Pro floorplan and can be clearly distinguished. Figure 5.5 shows the separation of the three individual shares on the floorplan of the FPGA, the three different shares are shown in magenta, light green and dark green.

We proceed with leakage assessment to evaluate whether or not the out-of-model leakage from coupling can be detected. To detect leakage in higher-order

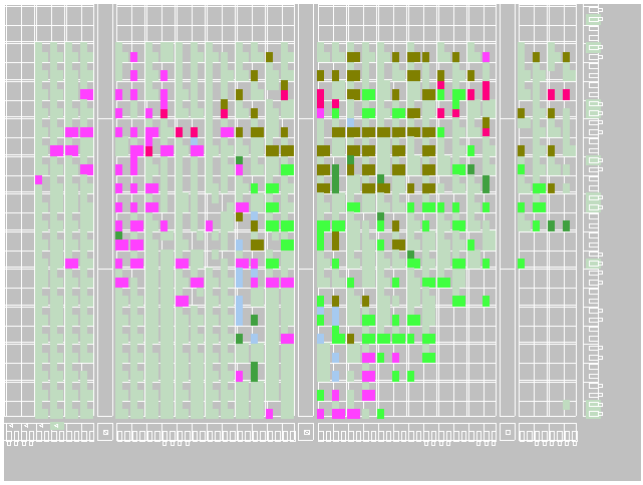

Figure 5.6: Placing all shares in close proximity leads to a design that leaks

moments, we run the t-test on preprocessed traces. The measurement setup and methodology is similar to previous experiments. A difference here is that we opt for a lower noise platform (Virtex-2 Pro FPGA on the SASEBO-G board [1]) to provide even more favorable measurement conditions.

PRNG Off. The result of the leakage detection test with the masks turned off is shown in Figure 5.7. As expected, the t-value threshold of ± 4.5 is exceeded meaning the design with disabled masks leaks with 20k traces. We conclude that the measurement setup is sound.

PRNG On. Turning the masks on results in the first- and second-order leakage detection tests in respectively the middle and bottom graphs shown in Figure 5.7. The expected second-order leaks are present and suggest that we have enough measurements to be able to detect leakage in lower-order moments, if any would be present.

The dashed line in Figure 5.9 shows the evolution of the point of maximum absolute value of the first-order leakage in function of the number of traces in 1M increments. The maximum of the absolute t-value fluctuates around the threshold but no steady increase in the maximum value is recognizable. Based on the t-test statistic we consider a $\sqrt{N}$ increase in the t-value with an increasing number of traces N as leakage. We therefore conclude that no out-of-model leakage is observable with 100M traces.

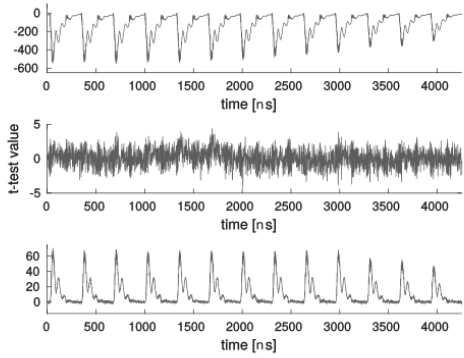

Figure 5.7: Leakage detection tests of a secure KATAN-32 TI, 20k traces masks off (top), 100M traces masks on 1st-order (middle), 100M traces masks on 2nd-order (bottom)

5.4.2 Leaking Threshold Implementation of KATAN-32

To investigate the effect of the placement and its potentially inducing coupling, we first convert the NGC netlist back to an HDL file. Since the netlist is only produced after the synthesis step, and therefore is influenced by the "Keep Hierarchy" constraint, the resulting HDL file consists of Xilinx specific primitives grouped into separate modules that reflect the hierarchical structure of the secure KATAN-32 TI. By merging the resulting HDL modules and assigning the "Keep" constraint to all signals, we preserve the integrity of the secure implementation while dropping the placement constraints originating from the "Keep Hierarchy" constraint. We proceed by synthesizing the HDL file with "Keep Hierarchy" set to false and force the placement of the components to the lower right corner of the FPGA floorplan using the "Prohibit" constraints. Figure 5.6 shows the floorplan of the FPGA. The three individual shares are now placed in close proximity. The three different shares are again shown in magenta, light green and dark green.

We proceed with the leakage detection tests.

PRNG Off. The result of the leakage detection test with the masks turned off is shown in Figure 5.8. Since the masks of the Threshold Implementation are set to zero, the t-value threshold of ± 4.5 is exceeded with 20k traces.

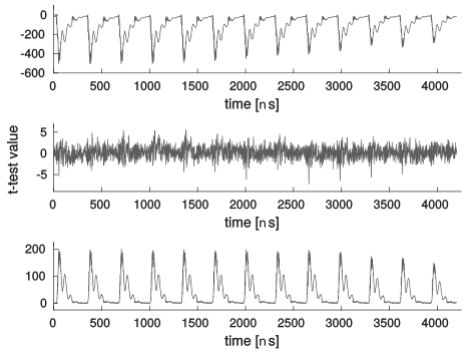

Figure 5.8: Leakage detection test of an insecure KATAN-32 TI, 20k traces masks off (top), 100M traces masks on 1st-order (middle), 100M traces masks on 2nd-order (bottom)

PRNG On. The middle and bottom graphs in Figure 5.8 show the result of the first- and second-order leakage detection tests with 100M traces respectively. Small, periodic first-order leaks are visible and indicate the presence of out-of-model leakage.

The solid line in Figure 5.9 shows the evolution of the point of maximum first-order leakage for the leaking KATAN-32 TI. Unlike the uncertain fluctuation around the ± 4.5 threshold for the secure KATAN-32 TI, we now see a steady increase in the maximum of the absolute t-value. As this t-value follows a $\sqrt{N}$ increase with an increasing number of traces N, we have confidence that it is caused by actual leakage. To increase the confidence in our observations, we repeated the experiments with a different fixed plaintext value (chosen randomly as $087D2EC1_{hex}$) as opposed to the plaintext value zero. Figure 5.10 shows the maximum value of the absolute t-value of the insecure design to be increasing, whereas this value does not exceed the ± 4.5 threshold in the secure design. We conclude that out-of-model leakage, albeit small, is observable.

5.5 Discussion

A Note on "Keep Hierarchy". In the majority of the threshold implementations literature, the "Keep Hierarchy" constraint is attributed with the function to keep the synthesis phase from optimizing over share boundaries. While this

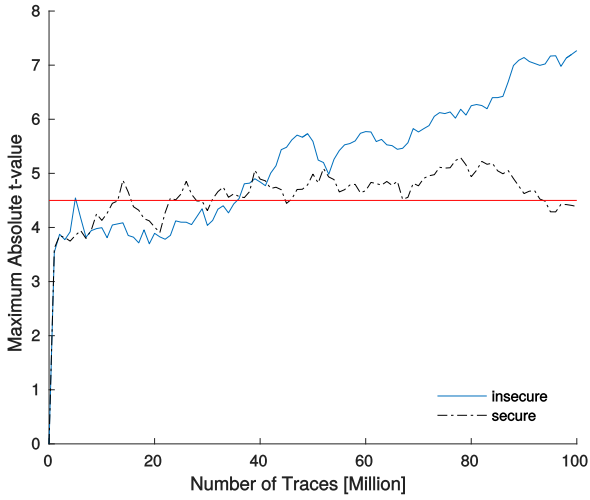

Figure 5.9: Evolution of the points of maximum absolute values of the leakage with increasing number of traces for the secure and insecure KATAN-32 TI and plaintext value 00000000_{hex}

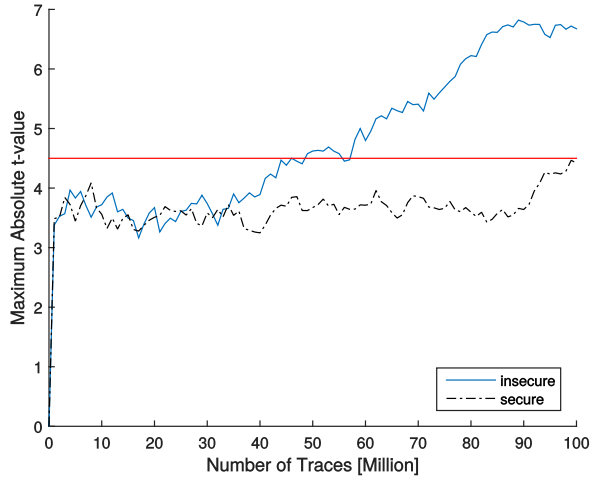

Figure 5.10: Evolution of the points of maximum absolute values of the leakage with increasing number of traces for the secure and insecure KATAN-32 TI and plaintext value $087D2EC1_{hex}$

explanation is correct and different shares are indeed prevented from being merged in the same LUT, "Keep Hierarchy" also serves the purpose of not packing different shares in the same FPGA slice, which is shown to potentially cause observable leakage. In FPGAs the "Keep Hierarchy" constraint is used to avoid optimizations that could merge shares and invalidate the non-completeness property. We achieve the same effect in our ASIC toolchain by first compiling different submodules with the "Compile Ultra" option and then compiling the resulting netlists using the regular "Compile" process. We have shown that care might still be required to avoid standard cells belonging to different shares to be placed in the vicinity of each other. Similarly, wires belonging to different shares should not be routed parallel.

A Note on the Measurement Platform. Our measurement setup is a low-noise platform based on a Virtex-2 Pro FPGA. The 90nm technology node it uses delivers clean power traces with rather large amplitudes. As the effects of out-of-model leakage we observed might not be as prominent with a 90nm technology as with smaller nodes, other side-channel evaluation boards will need evaluation. As crosstalk and IR drop are known to become more prominent with smaller technology nodes, the 65nm technology of the Virtex-5 on the SASEBO-GII platform [2], the 45nm technology of the Spartan-6 on the SAKURA-G board [74] and the 28nm technology of the Kintex-7 on the SAKURA-X platform form targets for further investigation.

5.6 Conclusion

In this chapter we showed evidence of coupling as a potential issue in masking schemes. We rule out the effects of glitches and early propagation by using threshold implementations to make sure the leakage we induced in our experiments originates from coupling effects only. We achieve a secure KATAN-32 TI using the state-of-the-art "Keep Hierarchy" implementation technique and show its security using state-of-the-art leakage detection methods. Afterwards, we induce out-of-model leakage by placing the gates and registers of the secure design in close proximity. This way we mimic real-world designs where shares would be placed close to each other on the chip to e.g. reduce timing of not only the masked core but other IPs as well. The leakage detection shows this new design to leak and leads us to the following conclusion. Leakage from coupling can be induced deliberately or "by accident" in masking schemes by placing shares in the vicinity of each other. While we only confirmed this effect on a Virtex-2 Pro FPGA, similar behavior can be expected on other FPGAs as well.

As is shown from the related TI FPGA implementations using the "Keep Hierarchy" option that pass the leakage detection test [21, 42, 128], this out-of-model leakage does not necessarily occur. From our experience, masked implementations go through several design iterations before the final security evaluation is obtained and included in a submission. It is not inconceivable that the problem we report has been observed in previous work.

Since this problem can be caused on an FPGA however, we believe that careful examination of other environments is required. We can not draw conclusions in general scenarios where shares of a masking scheme might be densely packed, e.g. in cryptographic ASIC implementations. The number of traces required for the leakage to be noticeable is high for our 90nm platform. Smaller process technologies are known to be more susceptible to crosstalk and IR drop coupling [114] and can lead to more leakage, and hence possibly insecure designs. The actual source of the observed leakage (e.g. crosstalk, IR drop, ...) is nontrivial to isolate and more targeted experiments are required, possibly in conjunction with SPICE simulations. Once these sources have been isolated, avoiding these problems earlier in the design cycle becomes feasible. This early detection of leakage sources could reduce the number of iterations needed to achieve secure designs.

Masked implementations have passed leakage detection tests even when the number of shares is decreased to the theoretical minimum number of shares $(d + 1)$ [36]. The "Keep Hierarchy" constraint is often mentioned as an important factor in achieving leakage free measurements. As we have seen in the previous chapter, this constraint is however not a guarantee for a leakage free implementation. Since these implementations use a lower number of shares, less noise is present and coupling can lead to leakages with a lower number of traces. These implementations are expected to be more favorable targets for actual key recovery attacks and form a solid subject for future work. In recent joint work with Maik Ender and Amir Moradi [44] the sources of these leakages are investigated in more detail. Whether these sources can be exploited to amplify the leakage and ultimately lead to key retrieval attacks remains to be seen in future work.

Chapter 6

Glitch-Resistant Masking Schemes Prevent FSA

We have spent the last three chapters exclusively discussing masking schemes. We change the focus slightly in this chapter, and investigate how some masking schemes can provide some resistance against active attacks natively. In particular we show that, under certain assumptions, masking schemes that are not vulnerable to glitches resist a specific type of "active side-channel attack", namely Fault Sensitivity Analysis (FSA).

FSA retrieves secret keys from the correlation of processed data values with their Fault Sensitivity (FS) [87]. The fault sensitivity is the intensity of an injected fault at which the device just starts malfunctioning. It is found by starting from normal, unfaulty operation of a device and gradually incrementing the intensity of a fault e.g. the supply voltage and/or clock frequency. The first point at which the device starts outputting either wrong or unintelligible results is the fault sensitivity. In the original work of Li, it was argued that masking schemes could provide resistance against FSA [87]. After several masking schemes were broken using FSA [96,101,102], they were discarded as FSA countermeasure and dedicated countermeasures against FSA started being researched. These can be categorized into gate-level countermeasures and RTL-level countermeasures.

The first gate-level countermeasure against FSA was proposed by Ghalaty et al. [62]. Their method is based on rearranging the gates of a circuit in order to balance out the sources of FSA leakage. An alternative approach at counteracting FSA attacks is through RTL-level countermeasures [86]. By only enabling stable outputs of the combinational logic to the register inputs, the

fault sensitivity is made constant and equal to the arrival of the enabling signal. Endo et al. choose the enable signal to be greater than the largest propagation delay of the circuit during post-manufacturing reconfiguration [54, 55].

While these countermeasures have a reasonably small impact on the circuit complexity and throughput, the resulting circuits are still vulnerable to passive side-channel attacks. It is reasonable to expect a circuit to require both protection against SCA and FAs. Therefore, combinations of threshold implementations and FA countermeasures have been investigated in more recent work: ParTI [129] and Private Circuits II enhanced TI (which we discuss in the next chapter) [45]. We will show in this chapter that both methods counteract FSA under certain assumptions since both approaches keep TI's non-completeness property.

The relation between FSA and power analysis attacks has been studied qualitatively by Li et al. [85]. Similar to their work, we analyze the relation between FSA and power analysis through the t-test leakage detection method. We put their work in a different light by showing that FSA and leakage detection tests are closely related.

This brief chapter is structured as follows. We provide a more detailed description of the Fault Sensitivity Analysis attack in Section 6.1. We proceed with a list of assumptions and their justification in Section 6.2. In Section 6.3 we argue that under our given assumptions, TI and the Roche and Prouff masking scheme resist FSA natively, and that the reason for this desired characteristic lies in their glitch-resistance. To this end, we highlight a relation between the t-test based leakage detection tests and FSA. We conclude the chapter in Section 6.4 and provide several directions for future work.

This chapter presents the main idea behind the work that was presented at FDTC 2018 [6] and coauthored with Arribas and Šijačić.

6.1 Fault Sensitivity Analysis

An FSA attack is mounted in two phases: a collection phase and a key retrieval phase. In the collection phase, fault sensitivities of input values are collected. During each encryption, the intensity of the applied fault FI is gradually increased. An encryption of a plaintext PT under fault intensity FI is denoted by $Enc(PT, FI)$. The first fault intensity at which the output becomes faulty is noted as the fault sensitivity for that input. This process of retrieving the FS is repeated for several different plaintexts. The algorithm for this procedure is given in Algorithm 7.

In the subsequent key retrieval phase, the attacker starts by making a (sub)key guess K_g. The attacker uses a function $f_{FS_g}(CT, K_g)$ to predict the fault intensity corresponding to this key guess FS_g and the ciphertext CT. An example of the function f_{FS_g} is the Hamming weight of the guessed intermediate value at the point of attack. The absolute value of the correlation ρ between the predicted fault sensitivities FS_g and the fault sensitivities of the implementation FS are computed. The highest correlation then leads to the correct key [87]. The algorithm for the key retrieval is given in Algorithm 8.

Two factors that codetermine the success of an FSA attack are the resolution of the fault injection the attacker can apply and whether or not a model or function $f_{FS_g}(CT, K_g)$ for the FS can be extracted. A clock glitch generator has shown to provide enough resolution for successfully mounting FSA attacks [56, 57]. The Hamming weight model has been used with success [87], but has been shown to not work on all S-boxes. One S-box for which the Hamming weight model can fail is the AES S-box with a field decomposition architecture [93] e.g. the Canright S-box.

Algorithm 7 Collection of the Fault Sensitivities

Input: The number of plaintexts N
Output: The list of corresponding ciphertexts $CT[i]$ and the associated list of Fault Sensitivities $FS[i]$

```
for i = 1 to N do
    PT[i] ← rand()
    FI ← 0
    CT[i] ← Enc(PT[i], FI)
    while Enc(PT[i], FI) = CT[i] do
        Increment Fault Intensity FI
    end while
    FS[i] ← FI
end for
```

6.2 Assumptions on the Masked Implementation and FSA Attack

In order to facilitate our explanation, we make the following assumptions.

1. The attacker does not exploit the faulty ciphertexts.
2. The attacker actually measures the timing properties of the circuit, i.e. the propagation delay through his FSA attack.

Algorithm 8 Key retrieval

Input: Length t of the (sub)key in bits, list of ciphertexts $CT[i]$, list of Fault Sensitivities $FS[i]$
Output: Key
  for $K_g = 0$ **to** $2^t - 1$ **do**
    for $i = 1$ **to** N **do**
      $FS_g[i] \leftarrow f_{FS_g}(CT[i], K_g)$
    end for
    $Cor[K_g] \leftarrow \rho(FS, FS_g)$
  end for
  Key $\leftarrow K_g$ for which $Cor[K_g]$ is maximum

3. The total leakage of the circuit is a linear combination of the leakages of the players or component functions, and the requirement for the masking schemes are satisfied.

4. The circuit starts from a uniformly drawn, random state before every encryption.

FSA was introduced as an attack to bypass fault detection mechanisms that withhold the output to prevent Differential Fault Analysis and other attacks that require faulty ciphertexts. Our first assumption is made with this specific scenario in mind, where fault detection is used to prevent DFA. We show that the FSA attack fails when the implementation is masked properly.

The validity of the second assumption can be explained by looking at the physical effect of the FSA attack. The attacker applies a fault and gradually increases its intensity, each time noting whether the output was correct or incorrect (or not released). After m injections, the attacker holds a list of m responses $[(\Delta_1, FI_1), ..., (\Delta_m, FI_m)]$ [93], symbolizing whether or not the fault injection led to correct or incorrect outputs under fault intensity FI_i. If $\Delta_i = 1$, we consider the computation to have finished correctly under fault intensity FI_i and $\Delta_i = 0$ otherwise. The fault intensity $\overline{FI}$ for which all $FI_i > \overline{FI}$ lead to $\Delta_i = 1$ is defined as the fault sensitivity for input x. This point can be considered as the last point in time for which not all output bits were valid and stable yet, which is essentially the propagation delay for the circuit's state and newly applied input.

The requirement of the third assumption means that, from a side channel point of view, the power consumption of the n different players of the masking scheme are decomposable. For simplicity, we limit our focus to the power consumption related to the players rather than considering the power consumption of the

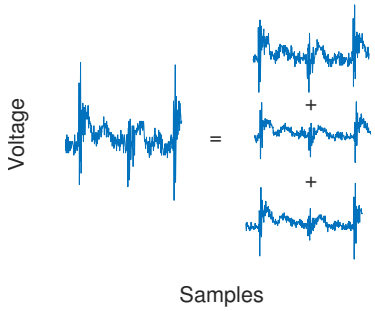

Figure 6.1: When the leakage requirement holds, the total power consumption trace can be decomposed into power traces of the different shared sub-circuits

whole circuit. In case the players operate in a parallel way (such as e.g. in our previously presented TIs), this means the power consumption of the players can be decomposed into separate instantaneous power consumption traces as shown in Figure 6.1 for a first-order TI with $n = 3$ players. When the players operate in a serial way (such as e.g. in the implementations of the Roche and Prouff masking scheme [43, 99]), the instantaneous power consumption of the n players are decomposed in their temporally separated power traces. As we have seen in previous chapters, this assumption is not always satisfied.

The fourth assumption is a "hidden" requirement in the ISW masking scheme [77]. The result of this assumption is that we can consider the fault sensitivity of the circuit to only depend on the new input instead of on both the previous input (or current state of the circuit) and the new input. A very low fault sensitivity would occur if the previous input and the new input are identical, as there is no propagation delay in that case. This can be exploited [96].

6.3 Glitch Resistance and Fault Sensitivity

In this section, we explain qualitatively why glitch resistant masking schemes resist FSA. For this purpose we first explain why, under our assumptions, the information from the propagation delay of the combinational part of a circuit is included in the t-test leakage detection. This justifies to look at FSA as an actively triggered side-channel attack when the faulty ciphertexts are not exploited.

6.3.1 Extending the Relation Between FSA and Power Analysis

The relation of the leakage in the power consumption side channel and FSA has been explored qualitatively in [85]. It was concluded that FSA and power analysis have partially overlapping leakage, but might contain different information. We build upon their initial investigation and argue that exploitable FSA information in the classical sense, i.e. the information of the propagation delay excluding the knowledge of the faulty outputs for a given input value, can be revealed with the t-test leakage detection test.

For every input x, a list $[(\Delta_{1,x}, FI_{1,x}), ..., (\Delta_{m,x}, FI_{m,x})]$ can be created by profiling the device. For an FSA attack to succeed, the threshold Fault Intensity $\overline{FI}$ for different inputs should be distinguishable. This means that if we would apply the t-test on this list, the confidence threshold should be exceeded in at least one point.

We can also look at the instantaneous power consumption traces as a leakage function of the circuit: $P_{inst}(t) = L(Circuit, t)$. Once the computation for input value x finishes, i.e. when the outputs are stable, we can note down the last point of activity as $P_{inst}(t_{PD,x})$. The t-test of an insecure, unmasked implementation would have points crossing the confidence threshold. For a masked implementation with 3 players or component functions, the instantaneous power consumption can be viewed as:

$$P_{inst}(t) = L(Player_1, t) + L(Player_2, t) + L(Player_3, t)$$

If no leakage is detected in the t-test of this implementation, we can conclude that no sample of the instantaneous power consumption depends on the sensitive data. Therefore, the largest propagation delay of all the sub-circuits or component functions does not reveal any information about the sensitive value. We conclude that an implementation that does not show leakage in the first-order t-test is secure in the FSA side channel. The sample rate of the oscilloscope with which the instantaneous power consumption trace is measured and the resolution of e.g. the clock glitch generator should be similar for this translation to hold.

6.3.2 Threshold Implementations Resist FSA

We use the first-order TI of an AND-gate ($z = xy$) with three shares to explain that FSA will not succeed on a threshold implementation if the faulty outputs are unexploited. Given the shares of the two inputs x and y as (x_1, x_2, x_3) and

(y_1, y_2, y_3), then the shares (z_1, z_2, z_3) of the output z are computed as:

$$z_1 = f_1(x_1, x_2, y_1, y_2) = x_1y_1 + x_1y_2 + x_2y_1$$

$$z_2 = f_2(x_2, x_3, y_2, y_3) = x_2y_2 + x_2y_3 + x_3y_2$$

$$z_3 = f_3(x_1, x_3, y_1, y_3) = x_3y_3 + x_3y_1 + x_1y_3$$

The three circuits f_i are evaluated in parallel as shown in Figure 6.2. If an attacker would now apply the procedure for FSA, a faulty output would be detected at some fault intensity. As the component functions f_i generally get different input values and are physically different instantiations, their fault sensitivities are not necessarily equal. Their relation can be set forth as:

$$FS_a(x_{a,b}, y_{a,b}) \leq FS_b(x_{b,c}, y_{b,c}) \leq FS_c(x_{a,c}, y_{a,c})$$

where the actual values of a, b and c depend on the state of the circuits f_i and their new inputs. The first fault intensity at which faults are detected is $FS_c(x_{a,c}, y_{a,c})$ and only provides information on the shares x_a, x_c, y_a and y_c. This information is insufficient to reconstruct the sensitive values x or y. An alternative explanation is that a first-order TI can be seen as a $(3, 3)$-thresholding scheme where all shares are needed to unmask the secret value. We argue that even if the attacker would know which component function is faulted, it remains unfeasible to gain enough information on the unmasked inputs, and therefore, TI resists FSA.

6.3.3 The Roche and Prouff Masking Scheme Resists FSA

We now qualitatively explain the resistance of the Roche and Prouff masking scheme against FSA using a first-order secured multiplication in $\mathbb{GF}(2^m)$. Note that Shamir's Secret Sharing is not defined in $\mathbb{GF}(2)$ so we can not use a single AND-gate as an example here. The shares of the two inputs x, y are given as (x_1, x_2, x_3) and (y_1, y_2, y_3). To obtain the shares (z_1, z_2, z_3) of the output z, the shared multiplication is performed in three steps following the BGW protocol, which is represented in Figure 6.3.

1. Each player i first computes t_i:

$$t_i = x_iy_i$$

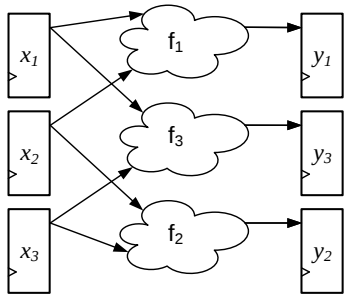

Figure 6.2: General structure of the threshold implementation example

2. Each player i then randomly selects an evaluation point r_i and remasks t_i:

$$q_{i,1} = t_i + (r_i \alpha_1)$$

$$q_{i,2} = t_i + (r_i \alpha_2)$$

$$q_{i,3} = t_i + (r_i \alpha_3)$$

   Each $q_{i,j}$ with $i \neq j$ is subsequently sent to the corresponding player j.

3. The outputs $q_{i,j}$ are then reconstructed by each player i as:

$$z_i = (q_{1,i}\lambda_1) + (q_{2,i}\lambda_2) + (q_{3,i}\lambda_3)$$

In order to avoid deterioration of the scheme from glitches, steps 1 and 2 are separated from step 3 using registers. In Figure 6.3, steps 1 and 2 of player i are performed by the combinational circuit f_i, and step 3 of player i is computed in circuit f_{i+3} after the register layer.

In published implementations of the Roche and Prouff scheme [43, 99], each player executes in a sequential fashion. An attacker can learn information on at most one player when mounting an FSA with a fault injection in one clock cycle. Since a first-order Roche and Prouff masked implementation is a $(3, 2)$-thresholding scheme, at least two shares are required to reconstruct the sensitive data. In order to successfully retrieve the key, an attacker thus needs to perform the FS collection phase of the FSA attack in at least two different clock cycles that process related shares. This is similar to SCA, where a second-order attack can theoretically break a first-order SCA resistant implementation.

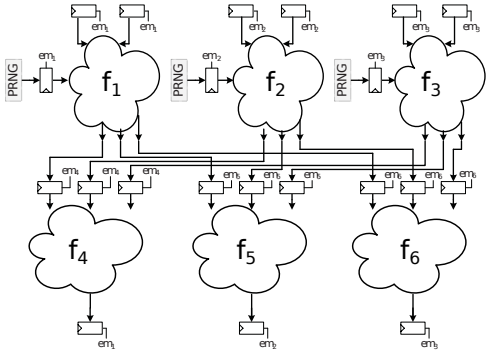

Figure 6.3: General structure of the Rivain and Prouff masked multiplier

6.4 Conclusion

In this chapter we investigated the resistance of two glitch-resistant masking schemes against the FSA attack. We explained that both the threshold implementations and the Roche and Prouff masking schemes never reveal information about any unmasked sensitive inputs through the propagation delay of the combinational circuit. To this end, we used both the threshold cryptographic nature of the masking schemes for the theoretical point of view, as well as the t-test leakage detection for our more practical argument. The resistance from FSA comes directly from the glitch-resistant nature of the schemes, which sets them apart from the masked AES implementations that were broken by Moradi et al. [101, 102]. It is therefore plausible that other glitch-resistant masking schemes, e.g. masking scheme using $d + 1$ input shares [47, 71, 120], show the same resistance. We argue that as long as the t-test leakage detection test is passed with a sample rate comparable to the resolution of a clock glitch, FSA resistance is implied when the faulty ciphertexts remain unexploited.

To conclude this chapter we propose following directions for future work. As a first direction, we can simulate and mount practical attacks using a clock glitch generator [56, 57] to solidify our claims in practice. We can additionally quantify how glitch-resistant masking schemes hold against Timing Violation Vulnerability Factor [143] for extra support.

A second direction is to relate the t-test (and the power side-channel in general) and FSA in a more quantitative way, further extending the work of Li et al. in [85]. This path could lead to alternative evaluation strategies to test the correct implementation of masking schemes in hardware. Additionally,

translating higher-order multivariate attacks to the FSA scenario by injecting glitches over multiple clock cycles can lead to interesting results.

As final direction we propose to investigate the resistance of these masking schemes to more advanced FSA-based attacks, e.g. Differential Fault Intensity Analysis [63]. While we limited our research to first-order glitch-resistant masking schemes, their higher-order implementations might provide increased resistance against these more advanced attacks. This track could lead to improved and more efficient countermeasures against combined passive attacks and these actively triggered type of side-channel attacks.

Chapter 7

Protecting PRESENT Against Combined SCA & Arbitrary Fault Injections

In the previous chapter, we have shown that under certain assumptions glitch-resistant SCA countermeasures protect against FSA natively. FSA has loose requirements on the type of injected faults for a successful key retrieval and the attack can be mounted with relatively cheap equipment. When the implementation is deployed with a glitch-resistant countermeasure, an attacker will have to move on to more expensive attacks to break the cryptosystem.

In this chapter, we turn our focus to countering more expensive attacks. Building again on a SCA resistant implementation, the goal is to provide additional protection against Fault Attacks. We specifically look at FAs that use more restrictive (and thus more expensive and powerful) fault patterns compared to the random model in FSAs. One example of a class of such powerful attacks is Differential Fault Analysis (DFA). DFA requires knowledge of the fault model, and the more accurate the fault model, the more powerful the attack. Another class of very potent active attacks are fault injections against the control logic, where a whole encryption could be bypassed to output (parts of) the key. As the overview in Chapter 2 shows, countermeasures against FAs generally rely on some form of redundancy, e.g. area or time redundancy, appending error correction or detection codes, physically shielding parts of the Integrated Circuit, or equipping the chip with fault injection detectors. In case tampering is detected, an alarm can be triggered to withhold the faulty ciphertexts from

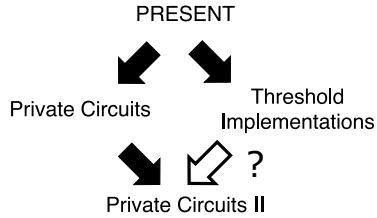

Figure 7.1: In this chapter we search for an alternative, more effective way to achieve a Private Circuits-II implementation of PRESENT

being output to prevent DFA.

Countermeasures against FAs have generally been researched separately from countermeasures against SCA, and can hence lead to a blowup in implementation cost when applied together. An exception can be found in Private Circuits II (PC-II) [76], an approach towards provable resistance against combined passive and active attacks. It is an extension of Private Circuits (which we abbreviate as ISW) [77], on which it relies to thwart passive attacks. As we discussed in Chapter 2, a practical drawback of the ISW algorithm is that its security relies on ideal gates, i.e. gates that evaluate only once per clock cycle and in the right order. Satisfying the ideal gate requirement in CMOS logic is costly and failing to do so leads to a deterioration in the security of the masking scheme through glitches and early evaluation [91]. As an alternative to ISW, the threshold implementations (TI) masking scheme [21, 105, 106] has gained in popularity for hardware applications since it does not require ideal gates. A first move towards a PC-II implementation with tamper-resistance against reset attacks was made by Rakotomalala et al. [115]. Their design is based on a manually coded, fully combinational ISW implementation followed by an encoding using FPGA specific primitives. In addition, an assessment of the security against timing violations is provided. The SCA security of its underlying ISW implementation is however degraded by early propagation and glitches [66, 123].

In this chapter we search for an alternative, more effective way to achieve a Private Circuits-II implementation of PRESENT (Figure 7.1). To this end, we first establish that threshold implementations are more cost effective than Private Circuits. We then proceed from the TI to apply the transformations responsible for the resistance against arbitrary fault injections.

We first provide details on the theoretical background w.r.t. the PRESENT algorithm, the ISW masking scheme and the PC-II combined countermeasure. In Section 7.2, we compare and evaluate both ISW and TI countermeasures

to mask the PRESENT S-box. The smallest resulting countermeasure is then chosen as basis for the PC-II implementation of which we detail its application in Section 7.3. We discuss the resistance against FAs in Section 7.4 and conclude this chapter in Section 7.5.

The content of this chapter is a combination of my contributions presented at FDTC 2016 [45] and published in IEEE Transactions on VLSI [46].

7.1 Background

7.1.1 PRESENT Block Cipher

The PRESENT symmetric key block cipher [25] is designed with the heavy constraints on area and performance of lightweight hardware applications in mind. As a result, it is aggressively optimized for hardware environments and forms an ideal candidate for Internet-of-Things applications. It was made an ISO standard in 2012 [78]. Its block length equals 64-bits and key lengths of 80- and 128-bits are supported, referred to as PRESENT-80 and PRESENT-128 respectively. PRESENT-80 is recommended for lightweight applications and is the target for our implementation. The PRESENT cipher iterates through 31 rounds followed by a final key whitening stage. Each round consists of a roundkey addition and a substitution-permutation network. The permutation layer is a bitwise rewiring governed by $p_{out}(i) = p_{in}(16i \bmod 63)$ and comes at no cost in hardware. The substitution layer applies a 4-bit S-box $S : \mathrm{GF}(2^4) \rightarrow \mathrm{GF}(2^4)$ on each nibble of the state registers. We refer to the original work for the full details [25].

7.1.2 Private Circuits or ISW

A Private Circuit [77] distinguishes between two transformations to achieve d-probing security. Both transformations retain the correctness of the original circuit.

A first, stateless transformation considers a circuit C as a directed acyclic graph where vertices are Boolean gates and the edges are wires. A second stateful transformation allows the inclusion of memory elements. It extends the graph to contain cycles as long as every cycle traverses at least one register. The addition of an initial state s_0 and external input and output wires completes the definition. We elaborate on the latter transformation as their stateful nature

makes them more practical in the context of cryptosystems. We refer to them as $T_{ISW}^{(d)}$ [1].

The transformation of a stateful circuit $C[s_0]$ to a d-probing secure circuit $C'[s'_0]$ is achieved through the following steps.

1. **Input Encoding.** An input bit a of the unprotected circuit C is transformed into a set of s input bits $\mathbf{a}$ to C' by means of uniform Boolean masking.

2. **Gates Encoding.** The $T_{ISW}^{(d)}$ construction relies on replacing the universal NOT and AND gates in C to NOT and AND gadgets[2] in C'. In addition, we describe the well-known XOR gadget since it leads to more efficient implementations.

   - **NOT Gate.** Transforming the NOT gate in C to a NOT gadget is achieved by inverting any uneven number of shares of the input $\mathbf{a} = (a_1, a_2, ..., a_s)$.
   - **AND Gate.** The AND operation $c = f(a, b) = ab$ in C is transformed into an AND gadget which performs the following sequential steps.
     (a) First, random bit values $r_{i,j}$ with $i \neq j$ and $1 \leq i \leq j \leq 2d + 1$ are generated.
     (b) Then, $r_{j,i} = (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$ with $i \neq j$ and $1 \leq i \leq j \leq 2d+1$ are computed.
     (c) Finally, the output bits are computed as $c_i = a_i b_i \bigoplus_{j \neq i} r_{i,j}$, with $1 \leq i \leq 2d + 1$ and $1 \leq j \leq 2d + 1$.
   - **XOR Gate.** The XOR gate $c = f(a, b) = a \oplus b$ is trivially transformed to an XOR gadget that implements $c_i = a_i \oplus b_i$, $i \in (1, ..., s)$ on the individual shares.

3. **Register Encoding.** Each register value x in C is stored in C' in its encoded form $\boldsymbol{x}$. After passing through a circuit of gadgets, the next state of the registers is still encoded and stored at the next clock cycle.

4. **Output Decoding.** An output bit is reconstructed from the shared bits by unmasking $c = \bigoplus c_i$.

[1] In this chapter we describe $T_{ISW}^{(d)}$ using $s = 2d + 1$ shares, but $s = d + 1$ shares are sufficient after a small modification [77].

[2] $T_{ISW}^{(d)}$ can achieve a higher efficiency when applied on larger Field multiplications instead of single AND gates [120].

7.1.3 Private Circuits II

Private Circuits II extends the Private Circuits masking scheme to add protection against an adversary capable of modifying values on a chosen number of wires in a circuit [76]. In contrast to other fault attack countermeasures [61] no part of the circuit needs to be completely free from tampering.

Private Circuits II comes in two styles, offering one of the following options.

1. Tamper resistance against an unbounded number of adaptive reset-only wire faults.
2. Tamper resistance against a bounded number e of arbitrary wire faults (set, reset or toggle) per clock cycle.

In both constructions, an (optionally infective [142]) circuit is achieved which resists both fault attacks and SCA attacks. Two transformations are carried out to transform a circuit C to a tamper resistant circuit C': one for the circuit itself and one for the data. The starting point is a side-channel resistant circuit. We investigate both ISW and TI to fit this role and choose the most efficient one as starting point for the second transformation.

Since we intend to protect the PRESENT block cipher against any type of fault, we limit our overview to the more general PC-II construction.

Tamper Resistance Against General Attacks on Wires

In this model the adversary can set the values in any of the circuit's wires to 0 or 1, as well as toggle their value. A limit e on the number of newly targeted wires per clock cycle is imposed but the attacker is allowed to release the perturbations without restrictions on the number or time of release. Hence, persistent or permanent faults are only counted on their introduction in the circuit.

To achieve the tamper resistance a circuit is first transformed into a dth-order SCA resistant circuit. Afterwards, a $2de$ repetition encoding is applied to all data values and the gates are replaced by so-called gadgets operating on these encoded values.

These actions are formalized as:

1. **Input Encoding.** The encoder transforms bit value 0 to a $2de$ vector $0^{2de} = (0, 0, ..., 0)$ and bit value 1 to a $2de$ vector $1^{2de} = (1, 1, ..., 1)$, where

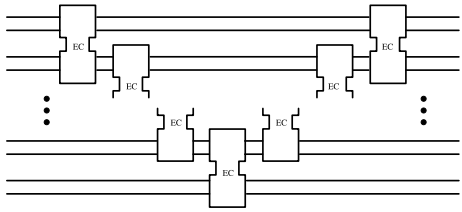

Figure 7.2: The error cascading stage propagates any e-bit fault to all wires before values are registered (illustrated here for $e = 1$)

d is the SCA security order and e is the limit on the number of tolerable faults. All other values are considered invalid ($\perp$). Furthermore, a special value $\perp^*$ is defined as $0^{de}1^{de}$.

2. **Gates Encoding.** The gates of the SCA resistant circuit are replaced by the gadgets of which the truth tables are listed in Table 7.1. Their implementation is of the form OR of ANDs of the input wires or their NOTs, or the NOT of such a circuit. Standard OR and AND gates can be used for constructing the gadgets, whereas the NOT gates should be reversible so that faults occurring on their output side propagate to their input side.

3. **Error Cascading.** Before values are registered or output, their wires go through an error cascading gadget. This way, detected errors will propagate and erase the data in the circuit and at the output in an infective way. An error is detected when an invalid encoding occurs in a wire vector. Fully infective behavior within one clock cycle is achieved by following the structure shown in Figure 7.2, where the error cascading gadget is listed in Table 7.1. We recall that this stage is optional [76].

4. **Output Decoding.** The final output can be decoded by ignoring all but one of the $2de$ output wires.

7.2 The Masking Process

In this section, we first compare the PRESENT S-box masked with ISW and with threshold implementations. Once our choice for TI is motivated, we reproduce and evaluate the PRESENT-TI proposed by Poschmann et al. [110] in order to assert a sound, SCA resistant basis for our PC-II implementation. We implement the first-order secure PRESENT with both a masked state and key, and test the result with state-of-the-art leakage detection tests.

Table 7.1: Truth tables for the gadgets, where d is the order of SCA security and e is the number of tolerable faults

AND Gadget		
Input $\mathbf{a}$	Input $\mathbf{b}$	Output $\mathbf{c} = \mathbf{ab}$
0^{2de}	0^{2de}	0^{2de}
0^{2de}	1^{2de}	0^{2de}
1^{2de}	0^{2de}	0^{2de}
1^{2de}	1^{2de}	1^{2de}
...	...	$0^{de}1^{de}$

XOR Gadget		
Input $\mathbf{a}$	Input $\mathbf{b}$	Output $\mathbf{c} = \mathbf{a} \oplus \mathbf{b}$
0^{2de}	0^{2de}	0^{2de}
0^{2de}	1^{2de}	1^{2de}
1^{2de}	0^{2de}	1^{2de}
1^{2de}	1^{2de}	0^{2de}
...	...	$0^{de}1^{de}$

NOT Gadget	
Input $\mathbf{a}$	Output $\mathbf{c} = \neg\mathbf{a}$
0^{2de}	1^{2de}
1^{2de}	0^{2de}
...	$0^{de}1^{de}$

Error Cascading Gadget			
Input $\mathbf{a}$	Input $\mathbf{b}$	Output $\mathbf{c}$	Output $\mathbf{d}$
0^{2de}	0^{2de}	0^{2de}	0^{2de}
0^{2de}	1^{2de}	0^{2de}	1^{2de}
1^{2de}	0^{2de}	1^{2de}	0^{2de}
1^{2de}	1^{2de}	1^{2de}	1^{2de}
...	...	$0^{de}1^{de}$	$0^{de}1^{de}$

7.2.1 ISW vs. TI

To our knowledge, the PRESENT S-box has not been implemented using ISW in hardware, whereas its TI has been published [110]. We therefore explore their difference in implementation cost based on a single AND-gate.

We first describe how to secure an AND gate using ISW when non-ideal gates are used. This approach was proposed by Reparaz et al. [117]. Security can be obtained while still keeping the integrity of ISW w.r.t. the operation order by inserting registers. In [123], a secure AND gadget was proposed by inserting registers behind every gate of a manually encoded ISW AND. The resulting impact on the area and performance is significant. A different approach for preventing leakage from glitches and early propagation is obtained by reasoning on the non-completeness property of TI. When investigating the different outputs $c_{1\leq i\leq 3}$ in the ISW AND gadget, it becomes clear that only one layer of registers is required to introduce non-completeness to the scheme.

The AND gadget is then executed in the following two clock cycles.

$$c_1 = [a_1 b_1 \oplus r_{1,2} \oplus r_{1,3}]_{reg}$$

$$c_2 = [a_2 b_2 \oplus a_2 b_1 \oplus r_{1,2} \oplus a_1 b_2 \oplus r_{2,3}]_{reg}$$

$$c_3 = [a_3 b_3]_{reg} \oplus [a_3 b_1 \oplus r_{1,3} \oplus a_1 b_3]_{reg} \oplus [a_3 b_2 \oplus r_{2,3} \oplus a_2 b_3]_{reg}$$

We opt for a first-order implementation with pipelining in mind. Therefore, the term $a_3 b_3$ gets computed in the first clock cycle to avoid storing the individual bits a_3 and b_3 separately.

In addition to the advantage of a lower circuit complexity, the AND gadget evaluation drops from the 9 clock cycles of [123] to 2. It was shown that the ISW AND gadget can be generalized to nonlinear operations in larger fields [120]. Similarly, the described partitioning of operations can be applied to achieve first-order secure multiplications in larger fields.

In comparison to the multi-stage ISW masked AND-gate, TI can achieve this in a single clock cycle, as it was designed to be free from a strict implementation sequence.

$$c_{1,out} = x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus r_1 \oplus r_2$$

$$c_{2,out} = x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus r_1$$

$$c_{3,out} = x_3y_3 \oplus x_3y_1 \oplus x_1y_3 \oplus r_2$$

We can compare these two masked AND-gates on their implementation cost. The ISW AND-gate needs two clock cycle, three units of randomness, nine multiplications and twelve additions. The TI AND-gate executes in one clock cycle, requires two units of randomness, nine multiplications and ten additions. The TI AND-gate is cheaper in many aspects, which is not surprising as it was designed specifically for hardware. It will be our choice of masking scheme on which we build our PC-II implementation. While this comparison is limited for simplicity, its conclusion stays valid when applied to larger structures. This is shown for the PRESENT S-box in our FDTC 2016 publication [45], where a more detailed discussion can be found.

7.2.2 Masking PRESENT with Threshold Implementations

The round key addition and permutation of the cipher can be performed on each share independently due to the linearity of these operations. The S-box operation is correspondingly harder to mask due to its nonlinearity. We therefore direct our attention to masking the nonlinear PRESENT S-box, which performs the substitution $S(x)$ given in Table 7.2.

A compact way of sharing the PRESENT S-box is proposed by Poschmann et al. [110]. They decompose the S-box $S(x)$ (of algebraic degree three) into two functions $g = G(x)$ and $f = F(x)$ (each of algebraic degree two) such that $S(x) = F(G(x))$. The S-box and its decomposed functions G and F are given in Table 7.2. In order to guarantee the uniformity at the input of F, the evaluation of the nonlinear functions G and F needs to be separated by registers. The total S-box is then computed in two clock cycles. The algebraic normal forms of $(g_3, g_2, g_1, g_0) = G(x_3, x_2, x_1, x_0)$ and $(f_3, f_2, f_1, f_0) = F(x_3, x_2, x_1, x_0)$ are

Table 7.2: 4-Bit to 4-Bit Substitution of the PRESENT S-Box [25] and a quadratic decomposition $F(G(x))$ [110]

x	0	1	2	3	4	5	6	7
$S(x)$	C	5	6	B	9	0	A	D
$G(x)$	7	E	9	2	B	0	4	D
$F(x)$	0	8	B	7	A	3	1	C
x	8	9	A	B	C	D	E	F
$S(x)$	3	E	F	8	4	7	1	2
$G(x)$	5	C	A	1	8	3	6	F
$F(x)$	4	6	F	9	E	D	5	2

listed below, where x_3, g_3 and f_3 represent the most significant bits.

$$g_3 = x_0 \oplus x_1 \oplus x_2$$

$$g_2 = 1 \oplus x_1 \oplus x_2$$

$$g_1 = 1 \oplus x_3 \oplus x_1 \oplus x_0x_2 \oplus x_0x_1$$

$$g_0 = 1 \oplus x_0 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2$$

$$f_3 = x_2 \oplus x_1 \oplus x_0 \oplus x_3x_0$$

$$f_2 = x_3 \oplus x_1x_0$$

$$f_1 = x_2 \oplus x_1 \oplus x_3x_0$$

$$f_0 = x_1 \oplus x_2x_0$$

For the shared versions of these equations, we refer to the original work [110].

7.2.3 Testing PRESENT-TI with Leakage Detection Tests

We perform the security evaluation on a SAKURA-G board [74] and proceed as described in the Preliminaries. For the PRESENT threshold implementation, this gives the following results.

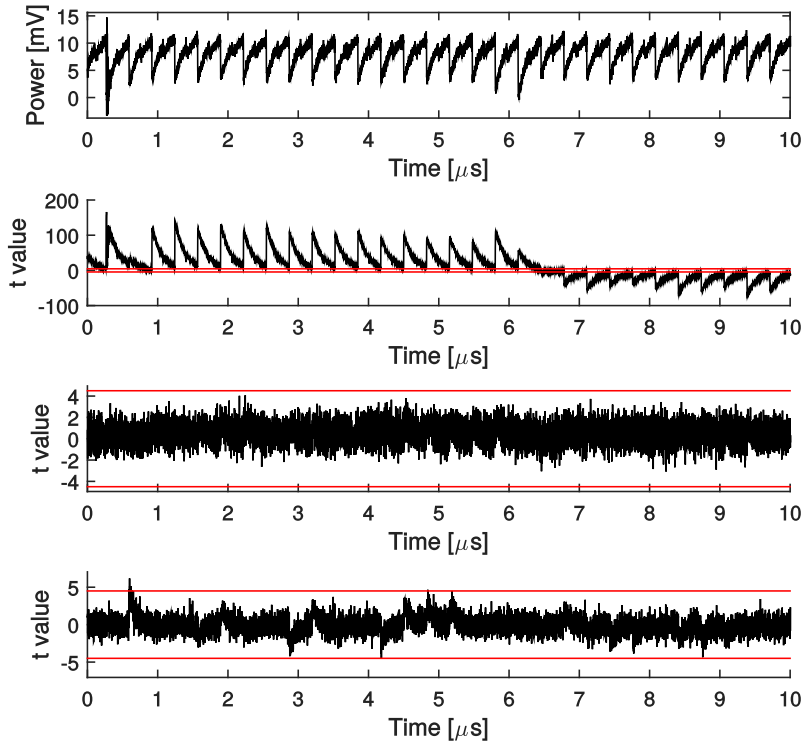

Figure 7.3: Masked PRESENT-TI, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 20k traces, first-order t-test with uniform masks using 100M traces, second-order t-test with uniform masks using 100M traces

PRNG Off. The result of the leakage detection tests for the PRESENT-TI with biased masks is shown in Figure 7.3. With 20k traces, the t-value goes beyond the confidence interval of ± 4.5 and we can conclude that our measurement setup is sound.

PRNG On. The result of the leakage detection test on PRESENT-TI with the activated PRNG is shown in Figure 7.3. Leaks are present in the second-

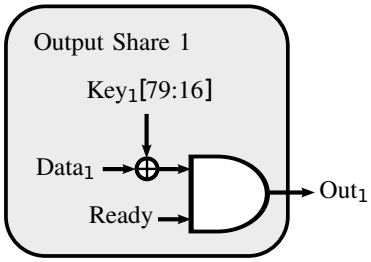

Figure 7.4: Output logic for PRESENT-TI

order t-test whereas no first-order t-values fall outside the confidence interval. Our PRESENT-TI implementation achieves the targeted first-order security with 100 million traces and provides us with a solid foundation for a PC-II implementation.

7.3 Applying PC-II

We proceed by applying the remaining PC-II transformations on the PRESENT-TI. While our model protects against an attacker injecting a 1-bit fault on a wire ($e = 1$) only, the explained process can be extended to any number of faulty bit injections.

7.3.1 Encoding

Transforming every wire into a $2de = 2$ wire pair is straightforward: all wires and registers are simply duplicated. In contrast to masking, where only the data dependent logic and registers must be protected, this duplication has to be applied to all wires and registers, including the ones responsible for the control of the data flow. If their protection is overlooked, intermediate states from which the key can trivially be retrieved can be made to prematurely appear at the outputs through a careful fault injection. This is exemplified in Figure 7.4: activating the ready signal to output the state at the start of an encryption would make a key retrieval straightforward for an attacker.

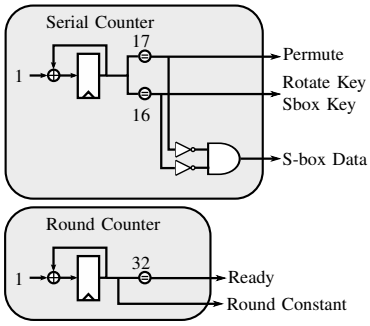

Figure 7.5: Control structure for PRESENT-TI

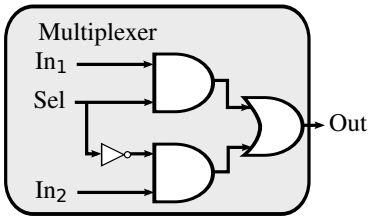

Figure 7.6: Logic structure of a multiplexer

7.3.2 Gates Encoding

After all wires and registers are encoded, we transform all gates in the circuit to their respective PC-II gadgets. In addition to the gates of the shared S-box, permutation and round key addition, this transformation has to include the internal gates of the two adders, the multiplexers and the comparators of the design also. An example of this can be found in the structure of the PRESENT-TI control logic, which is shown in Figure 7.5. The internal signals of e.g. the multiplexers (shown in Figure 7.6) have to undergo the encoding and the gates transformation as well.

In Section 7.2, it was noted that reversible NOT gates are required for PC-II in the general attack model. The reason for this is described in the original work [76] and boils down to being able to consider the NOT gates as part of atomic AND gates inside PC-II gadgets. We can omit the need for reversible NOT gates in our FPGA implementation by packing the logic of each single PC-II gadget inside individual LUTs of the Spartan-6 FPGA. With the original work's assumption that attacks are only performed on wires, the design remains a valid PC-II implementation as a whole LUT can be considered atomic. For

standard cell ICs, this can be achieved by placing the atomic standard cells related to the same gadget adjacently and keeping their connections on the lowest routing layers.

In our work we achieve this by expressing all operations in the Hardware Description Language of the PRESENT-TI using following atomic gates: AND, XAND[3], OR, NOR, XOR, XNOR and NOT gates. Since all these functions have at most four inputs and two outputs in their encoded form, we map these to the 6-input, dual output LUTs of the Spartan-6 FPGA using the Xilinx "*LUT_MAP*" constraint. This is an alternative to hard coding the LUT functionality as was done in the PC-II implementation of Rakotomalala et al. [115].

7.3.3 Error Cascading

Error cascades are nonlinear gadgets that forward the input unless an invalid encoding is detected at one of or both the inputs. In that case, both its encoded outputs are made invalid ($\perp^*$).

We provide a toy example to explain the effect of the error cascading stage on the SCA security. Assume we have two shares of a 1-bit value $s = s_1 \oplus s_2$ such that $s_1 = s \oplus r$ and $s_2 = r$, where r is a random bit drawn from a uniform distribution. These two values are passed through an error cascade, of which the function is given in Table 7.1. Its underlying circuit is of the form OR of ANDs of all the shares and leads to following four outputs.

$$s_{1,EC,1} = (s_{1,1}\neg s_{1,0}\neg s_{2,1}s_{2,0}) \oplus (s_{1,1}\neg s_{1,0}s_{2,1}\neg s_{2,0})$$

$$s_{1,EC,0} = (\neg s_{1,1}s_{1,0}\neg s_{2,1}s_{2,0}) \oplus (\neg s_{1,1}s_{1,0}s_{2,1}\neg s_{2,0})$$

$$s_{2,EC,1} = (\neg s_{1,1}s_{1,0}s_{2,1}\neg s_{2,0}) \oplus (s_{1,1}\neg s_{1,0}s_{2,1}\neg s_{2,0})$$

$$s_{2,EC,0} = (\neg s_{1,1}s_{1,0}\neg s_{2,1}s_{2,0}) \oplus (s_{1,1}\neg s_{1,0}\neg s_{2,1}s_{2,0})$$

For the error cascading stage to be effective, it should pass all pairs of wires as shown in Figure 7.2. This creates a combinational circuit that nonlinearly relates all shares, and invalidates the non-completeness property. When using CMOS,

[3] a XAND $b = (\neg a)b$

glitching and early propagation of values can cause the circuit to potentially leak. Since this stage is optional [76], we will omit its implementation.

7.3.4 Leakage Detection

The resulting implementation needs to satisfy first-order SCA security. We follow the same approach as with our previous security evaluations to test the security of the design.

PRNG Off. The result of the leakage detection tests for the biased PC-II protected PRESENT is shown in Figure 7.7. With 20k traces, the t-value goes beyond the confidence interval of ± 4.5 and we can conclude that our measurement setup is sound.

PRNG On. The results of the leakage detection tests on PC-II protected PRESENT with the activated PRNG is shown in Figure 7.7. Clear leaks are present in the second-order t-test while no first-order t-values fall outside the confidence interval. The PC-II protected PRESENT implementation achieves the targeted first-order SCA security with 100M traces.

7.3.5 Circuit Complexity

Table 7.3 lists the circuit complexities of the individual components of both the TI and PC-II implementation of PRESENT. Table 7.4 lists the circuit complexities of the individual PC-II gadgets we use in our PC-II design. All estimations are obtained with the Compile Ultra option in Synopsys 2010.03 and the NanGate 45nm Open Cell Library [81].

For ICs, when going from a first-order SCA resistant TI to a first-order SCA resistant PC-II implementation resisting 1-bit faults, the circuit complexity is increased with a factor of 8.75. The bulk of this factor originates from the increase in circuit complexity of the combinational gadgets compared to their corresponding gates. The duplication of the registers only leads to a duplication in circuit complexity for the sequential parts.

Table 7.5 lists the resources required for our implementations on the Xilinx Spartan-6 FPGA. While the ISW and PC-II version of Rokotomalala [115] have the same netlist, and result in a comparable resource consumption, this is not the case for our PRESENT. Instead, we notice a significant overhead for the PC-II implementations compared to the TI version: the number of slices is

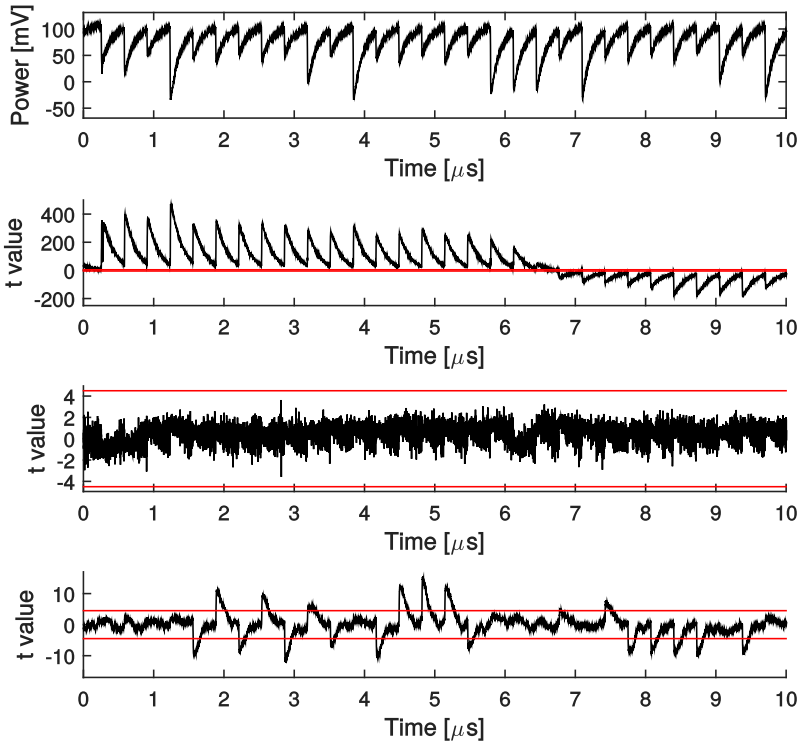

Figure 7.7: Masked PC-II protected PRESENT, from top to bottom: average power consumption trace of 1.5 rounds of a masked encryption, first-order t-test with biased masks using 20k traces, first-order t-test with uniform masks using 100M traces, second-order t-test with uniform masks using 100M traces

increased with factor 19.23. The first reason is that we did not manually code our TI. The FPGA tools can group several gates in a single LUT to reduce the total number of utilized slices. Because the LUTs are then maximally utilized, we were not able to simply adjust the LUT configuration to change the single output gates to double output gadgets. The second reason is that we can not use the standard multiplexers, but instead have to create PC-II style multiplexers from LUTs. This applies to other FPGA structures as well.

Table 7.3: Circuit Complexity of Different Functions of the PRESENT Designs

	Circuit Complexity [GEs]	
	TI	TI + PC-II
S-box $S(x) = F(G(x))$		
$F(x)$	75.06	533.16
$G(x)$	178.50	1016.19
Single Share	1619.23	14537.41
Total PRESENT-80	5236.49	45842.57

Table 7.4: Circuit Complexity of the Different PC-II Gadgets Used

PC-II Gadget	AND	XAND	OR	NOR	XOR	XNOR	NOT
GEs	7.41	7.41	7.41	7.74	9.09	10.10	2.02

7.4 Resistance Against Fault Attacks

Before concluding this chapter, we analyse the resistance of our implementation against known FAs. We start with an attack against the control structure and follow with two known DFAs on PRESENT.

Table 7.5: Resource comparison of the different first-order SCA resistant PRESENT versions on the Spartan-6 FPGA

	TI	TI + PC-II General Attack ($e = 1$)
Number of Slices	646	12422
Number of Slice Flip Flops	646	1294
Number of LUTs	889	12390
Consumed Random Bits	0	
Number of Clock Cycles	578	

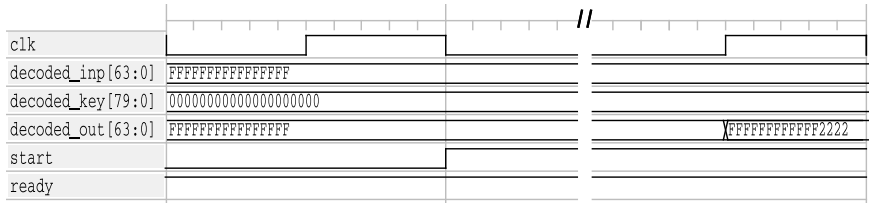

Figure 7.8: Traces of signals from the PRESENT-TI implementation with a set fault on the `ready` signal

7.4.1 Fault Attack Simulation

As mentioned in the example in Section 7.3.1, an attractive target for an attacker to inject a fault on is the ready signal. The ready signal guards the intermediate states from appearing prematurely at the output and activates the outputs only when the cipher has finished the right number of rounds. By validating this signal at the start of an encryption using a fault injection, intermediate states of the cipher become observable and cryptanalysis becomes feasible.

We now simulate this fault injection on the ready signal of the unprotected and PC-II protected PRESENT implementations. Figure 7.8 shows simulation traces of several signals of the unprotected PRESENT design. The `clk` signal represents the system clock, the `start` signal activates the start of an encryption and the `ready` signal indicates when the output is available. We show the decoded input, key and output shares by `decoded_inp`, `decoded_key` and `decoded_out` respectively. The `ready` signal in this example is stuck-at-1, i.e. every intermediate state is observable at the output of the cipher. By knowing the plaintext and observing the first intermediate result, the key can be obtained without much effort in an unprotected implementation. In Figure 7.9 we show the same simulation applied on the PC-II protected PRESENT. Since the `ready` signal and the bits of the state values pass through AND gadgets (Figure 7.4), their decoded output values become zero when a fault is present at their inputs. The invalid signal 01 on the encoded `ready` signal is detected, and the countermeasure renders the injected fault unexploitable.

Similarly, attacks on the data that are covered by our fault model will not succeed. Figure 7.10 shows a correct and faulty encryption, where the fault is generated by flipping an input bit of the first S-box look-up of the penultimate round. Figure 7.11 shows this same 1-bit fault injected in the PC-II protected PRESENT. While not all ciphertext bits are zeroized due to the absent infective error cascading stage, the injected fault is detected and invalid values are

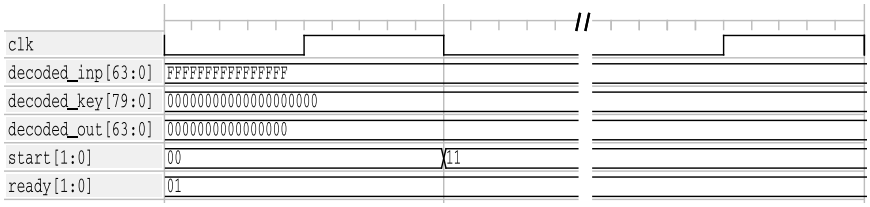

Figure 7.9: Traces of signals from the PC-II protected PRESENT implementation with a set fault on one of the wires of the encoded `ready` signal

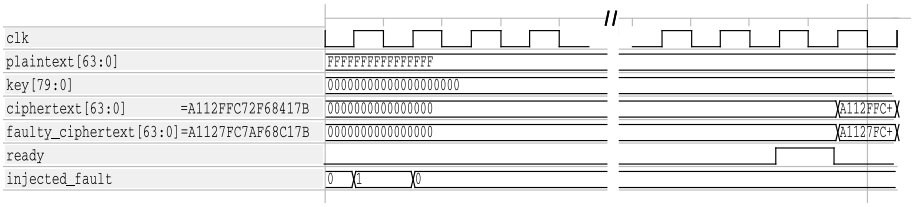

Figure 7.10: Traces of signals from the PRESENT-TI implementation with a set fault on the first share of the S-box input

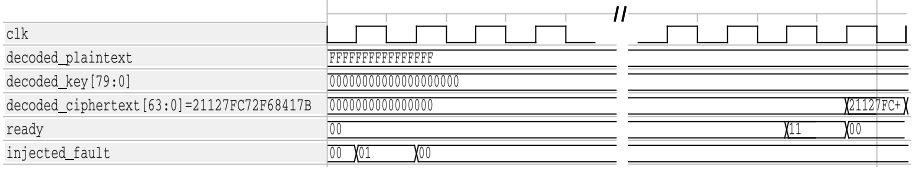

Figure 7.11: Traces of signals from the PC-II protected PRESENT implementation with a set fault on one of the wires of the encoded first share of the S-box input

propagated. This results in several bits turning zero: $A_{16} = (1010)_2$ in Figure 7.10 becomes $2_{16} = (0010)_2$ in Figure 7.11.

7.4.2 Increased Resistance Against Differential Fault Analysis

We now evaluate the increased resistance against two DFA attacks on PRESENT: firstly a DFA on the key schedule introduced by Wang et al. [138], and secondly a DFA on the internal state introduced by Bagheri et al. [8].

DFA on the PRESENT Key Schedule

The attack on the key schedule uses a fault model where 4-bit random faults are injected in either the 30th or 31st round key. Once an attacker obtains (on average) 64 pairs of correct and faulty ciphertexts, the secret key can be retrieved with a complexity of 2^{29}.

Since our implementation only provides protection against 1-bit faults, a successful key retrieval is possible when the attacker's power is outside the model. Some increased DFA resistance is however obtained from our implementation. Firstly, an attacker now has to inject an 8-bit fault instead of a 4-bit fault and needs to make sure the 8-bit fault is an encoded version of the 4-bit fault, otherwise the circuit will start to infectively erase values by propagating the invalid signal $\perp^*$. From a theoretical point of view, only 16 of the 256 8-bit values lead to a useful fault injection. As a result, the fault injection procedure will require 16 times more effort when the faults are random and uniformly distributed, leading to needing 1024 pairs of correct and faulty ciphertexts on average for a successful key retrieval.

DFA on the PRESENT State Registers

Bagheri et al. present two DFAs on the PRESENT state registers. The first uses a fault on a single bit of the intermediate state at the start of the S-box layer of the last round. This attack leads to a retrieval of the last subkey with an average of 48 correct and faulty ciphertext pairs. A second attack reduces the number of correct and faulty ciphertext pairs to 12, by injecting random 4-bit faults in the 29th round.

The first attack using the single bit fault falls within the bounds of what our implementation can secure and will always fail, as shown by Figure 7.11. The second attack will require the 4-bit random fault to be extended to a restricted 8-bit fault similar to the condition of the previously studied DFA on the key schedule. The fault injection procedure for this second attack to succeed will require 16 times more effort than the unprotected version when the faults are

random and uniformly distributed. On average it will then require 192 correct and faulty ciphertext pairs for a successful key retrieval.

7.5 Conclusion

In this chapter, we implemented and evaluated a Private Circuit II of the PRESENT block cipher. We used the serialized threshold implementation of Poschmann et al. [110] as starting point. We obtained an implementation that resists combined first-order side-channel attacks and arbitrary single-bit fault attacks. The circuit complexity compared to a side-channel resistant circuit increases with factor 8.75 and mainly originates from the complexity of the gadgets. While this cost is significant, our design benefits from an increased resistance against differential fault attacks even when faults fall outside the security model. In addition to the data path, the control logic is secured, which protects the implementation from trivial attacks, e.g. revealing intermediate state values at the output of the circuits by injecting a fault in a well chosen control signal. Our implementation was submitted to state-of-the-art leakage detection tests, which it passed with power consumption traces corresponding to 100 million encryptions. Previous work has shown that applying ISW or PC-II is not trivial and prone to many subtle mistakes that can lead to insecure designs. As the traps and pitfalls of ISW were covered in the work of Roy et al. [123], we directed our focus towards PC-II.

Researching practical but provable countermeasures against combined side-channel analysis and fault attacks is a relatively new research area. At the time of writing, only four published implementations have been tested and evaluated: ParTI [129], the PC-II enhanced TI presented in this work, CAPA [118], and an extension of the Prouff and Roche masking scheme [131]. Applying classical FA countermeasures on top of SCA resistant implementations is costly and inefficient, and a more integrated approach can prove beneficial. To this end, the field of Multi-Party Computation (MPC) can provide a solid starting point [118, 130]. Since our implementation does not consume any randomness during its execution, we propose to apply PC-II to a masked implementation that relies on a PRNG, and to measure the resulting overhead.

Chapter 8

Conclusions & Open Problems

In this chapter, we summarize the content of this work and formulate its main take-aways by answering the research questions we posed in the Introduction (Chapter 1). We conclude with directions for future work.

8.1 Summary

We described the research landscape at the onset of this PhD in the Introduction and presented our contributions in the chapters that followed. We now summarize our contributions to the field by answering our research questions.

We have addressed our first set of research questions that relate to the masking theory and its translation to practice with the presentation of state of the art masking schemes (Chapter 3) and our masked AES implementations (Chapter 4). By presenting side-channel secure multiplication algorithms for Boolean, TI-like masking, Inner-Product masking and a novel Polynomial masking construction next to one another, we were able to extract a generalized masking structure that extends the structure of the Consolidated Masking Scheme by Reparaz et al. [117]. We presented a trade-off that can be made: the number of clock cycles of an elementary multiplication can be reduced by increasing the randomness.

We proceeded with evaluating the security of several Boolean, TI-like masked implementations on an FPGA platform. We obtained unexpected results in the

t-test leakage detection tests: leakage was detected at orders where no leakage should be present. This led us to our next research question.

We presented evidence that leakage in masked implementations can be induced through the placement of shares on a Virtex-2 Pro VP7 FPGA platform. We distinguished two case studies in our conducted experiments. In a first test, we went through the regular design flow and were careful in that the masking scheme was translated to the FPGA correctly. In other words, no properties of the masking scheme were violated during the synthesis and mapping steps. We performed leakage detection tests and asserted the expected security. For the second test we took the technology netlist of this secure implementation, and pushed all components to one corner of the FPGA (or at least to one corner in the design tool's representation of the FPGA). During evaluation we detected first-order leakage in the first-order secure implementation. The only change between the two tests was the specific placement and routing. The reason for this behavior can be traced to the underlying assumption of masking schemes, i.e. that the total leakage of a device can be written as the linear combination of the leakages from the individual shares and component function. We have given two well-known coupling mechanisms that can violate this assumptions: crosstalk and power supply noise or IR drop.

After our research on the theory of masking schemes and their implementation, we directed our focus to extending masking to thwart a more powerful adversary. We considered an adversary that is not limited to passive attacks alone but can combine them with active attacks. We argued that under certain assumptions, glitch-resistant masking schemes are natively resistant against Fault Sensitivity Analysis, a type of actively triggered side-channel attack. We additionally uncovered a link between FSA and the t-test leakage detection test: when the precision of the fault injection is comparable to the sample rate of the measurement oscilloscope, the potentially data dependent fault sensitivity is encompassed in the t-test.

Since not all fault attacks can be protected against by masking schemes [29], we proceeded with a case study of a combined countermeasure. We implemented the PRESENT block cipher using the Private Circuits II (PC-II) countermeasure, which resists both passive side-channel attacks and active fault attacks. PC-II is built from a chain of transformations, of which the ISW masking scheme or Private Circuits is the first one. For hardware specifically, it is known that TI is more desirable as its security does not deteriorate in the presence of glitches. We obtained an overall more cost effective implementation resisting combined attacks by substituting the ISW scheme for TI. Still, the resulting implementation is shown to have a high circuit complexity and more research is needed to alleviate this overhead. We noticed that although Private Circuits II considers both side-channel analysis and fault attacks, their mitigation

strategies are still applied sequentially. A lower implementation overhead could be achieved when truly considering both attacks simultaneously in the design of the countermeasure. The redundancy introduced by the masking scheme could e.g. be leveraged by the FA countermeasure and vice versa. Another reason for the high implementation cost is the protection of the control logic against fault attacks. This is often omitted in related work [118, 130], where similar to masking only the datapath is protected, but has recently regained research attention [3].

8.2 Directions for Future Work

During our research we came across a multitude of additional questions. We categorize them in three main directions here and briefly discuss each.

Further Generalizing and Relating Schemes. We extracted a general structure from three masking schemes: Boolean, TI-like masking, Inner-Product masking and Polynomial masking. This generalization could be extended by looking at Multi-Party Computation (MPC). Multi-party computation, which shares similarities with masking, studies a.o. security in the presence of passive and/or active adversaries. Research that borrows techniques from MPC has been initiated [118, 130], but is possibly in too early a stage to be generalizable.

As the application of error correction and detection codes is an established countermeasures against FAs, looking at masking schemes from a coding theoretic perspective can lead to the design of cost-effective combined countermeasures. The link between coding theory and masking was studied by Castagnos et al. [34].

Correct Application of Countermeasures. A hardware designer implements a masking scheme in a certain Hardware Description Language (HDL) like Verilog or VHDL. The resulting code is then input to the design flow, a chain of algorithms that translates the code to a format related to the target platform. Many manipulations are performed on the code of the masking scheme during this translation, and security crucial properties can potentially be violated. One example is the merging of different operations in a single LUT. If the merged operations belong to different component functions, the non-completeness property is violated. This is generally mitigated by synthesizing with the "Keep Hierarchy" constraint. In addition to violating the properties of the masking scheme, assumptions on the leakage behavior can be violated as well.

An interesting question for future work is how to test whether a countermeasure was applied properly at different points in the design flow. Verification tools can operate on many levels. The uniformity of an implementation can be checked at the algorithmic level on a high-level programming language code of the implementation. The non-completeness can be checked on the Register-Transfer Level (RTL) with the HDL code of the implementation. The output files of the synthesis tools can be used to check the physical level for correct logic separation of primitives (e.g. slices in FPGAs or standard cells in ASICs).

More challenging is to check whether or not the underlying assumptions of the masking schemes are valid in the targeted hardware platform. For FPGA platforms, where details on the specific power consumption of primitives are often unknown, experiments will need to bring clarity about this potential issue. For ASICs, in case the details of the cell library are known, simulation tools can be created to analyze violated assumptions at design time before taping out the chip. This can lead to a significant reduction in the number of design iterations and can hence save both time and money during the design process.

A Move Towards System Security. The lab environment in which we test our designs differs in many aspects from the real-world environment an attacker would face. One difference is in how we provide the randomness to our masked core. We supply randomness obtained from an AES in OFB mode of which we do not measure the power consumption. In other words, the randomness we use for mask refreshing is of high quality and appears at the right moment and in the right quantity. A rather urgent direction for future work is to look into this assumption, and how we can evaluate more realistic scenarios. Questions that need answering in this respect are: "Where does the randomness come from?", "What requirements on the PRNG are needed for a certain degree of side-channel security?", "How does the circuit complexity, power consumption, latency and side-channel security scale with different PRNGs?", and "Can attacking the PRNG lead to an easier attack path than attacking the masking scheme itself?".

Due to its speed and ease of use, the t-test has been leveraged to argue the side-channel security of many recent implementations. The t-test is however very hard to satisfy and can detect leakage that might not be exploitable at all. Passing the t-test might not be necessary or even desired from a real-world perspective, as it entails a significant overhead in implementation cost. As the attacker's effort is ultimately what matters, an interesting question for future work is how the t-score relates to an attacker's effort to break a system? Research that combines the t-test and key retrieval is needed to isolate what leads to actually exploitable problems.

During my PhD I investigated the application and evaluation of the state-of-the-art theory on masking schemes. I came across unexpected behavior when testing the leakage. While the theory of the schemes and their security proofs are solid, their implementation can still show leakage. Hence, as a final conclusion of my PhD, I want to stress that the practical application of theory is crucial, as the Grey-Box model originates exactly from the application of theory in a real-world environment. While theoretic research is important and valuable, it only derives meaning in our field when applied and evaluated in practice.

Bibliography

[1] Research Center for Information Security, National Institute of Advanced Industrial Science and Technology, Side-channel Attack Standard Evaluation Board SASEBO-G Specification. `http://satoh.cs.uec.ac.jp/SASEBO/en/board/sasebo-g.html`.

[2] Research Center for Information Security, National Institute of Advanced Industrial Science and Technology, Side-channel Attack Standard Evaluation Board SASEBO-GII Specification. `http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-GII-en.html`.

[3] Aghaie, A., Moradi, A., Rasoolzadeh, S., Schellenberg, F., and Schneider, T. Impeccable Circuits. *IACR Cryptology ePrint Archive 2018* (2018), 203.

[4] Anderson, R., and Kuhn, M. Tamper Resistance – a Cautionary Note. In *In Proceedings of the Second USENIX Workshop On Electronic Commerce* (1996), pp. 1–11.

[5] Arribas, V., Bilgin, B., Petrides, G., Nikova, S., and Rijmen, V. Rhythmic Keccak: SCA Security and Low Latency in HW. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2018*, 1 (Feb. 2018), 269–290.

[6] Arribas, V., Cnudde, T. D., and Šijačić, D. Glitch-Resistant Masking Schemes as Countermeasure Against Fault Sensitivity Analysis. In *FDTC* (2018), IEEE Computer Society, pp. 1–8.

[7] Arribas, V., Nikova, S., and Rijmen, V. VerMI: Verification Tool for Masked Implementations. *IACR Cryptology ePrint Archive 2017* (2017), 1227.

[8] Bagheri, N., Ebrahimpour, R., and Ghaedi, N. New differential fault analysis on PRESENT. *EURASIP J. Adv. Sig. Proc. 2013* (2013), 145.

[9] Balasch, J., Faust, S., and Gierlichs, B. Inner Product Masking Revisited. In *EUROCRYPT (1)* (2015), vol. 9056 of *Lecture Notes in Computer Science*, Springer, pp. 486–510.

[10] Balasch, J., Faust, S., Gierlichs, B., Paglialonga, C., and Standaert, F. Consolidating Inner Product Masking. In *ASIACRYPT (1)* (2017), vol. 10624 of *Lecture Notes in Computer Science*, Springer, pp. 724–754.

[11] Balasch, J., Faust, S., Gierlichs, B., and Verbauwhede, I. Theory and Practice of a Leakage Resilient Masking Scheme. In *ASIACRYPT* (2012), vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 758–775.

[12] Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., and Standaert, F. On the Cost of Lazy Engineering for Masked Software Implementations. In *CARDIS* (2014), vol. 8968 of *Lecture Notes in Computer Science*, Springer, pp. 64–81.

[13] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Standaert, F., and Strub, P. Improved Parallel Mask Refreshing Algorithms: Generic Solutions with Parametrized Non-Interference & Automated Optimizations. *IACR Cryptology ePrint Archive 2018* (2018), 505.

[14] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., and Zucchini, R. Strong Non-Interference and Type-Directed Higher-Order Masking. In *ACM Conference on Computer and Communications Security* (2016), ACM, pp. 116–129.

[15] Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., and Strub, P. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT (1)* (2017), vol. 10210 of *Lecture Notes in Computer Science*, pp. 535–566.

[16] Becker, G., Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Kouzminov, T., Leiserson, A., Marson, M., Rohatgi, P., et al. Test vector leakage assessment (TVLA) methodology in practice.

[17] Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., and Vergnaud, D. Private Multiplication over Finite Fields. In *CRYPTO (3)* (2017), vol. 10403 of *Lecture Notes in Computer Science*, Springer, pp. 397–426.

[18] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC* (1988), ACM, pp. 1–10.

[19] Bhooshan, R., and Rao, B. P. Optimum IR drop models for estimation of metal resource requirements for power distribution network. In *VLSI-SoC* (2007), IEEE, pp. 292–295.

[20] Biham, E., and Shamir, A. Differential fault analysis of secret key cryptosystems. In *CRYPTO* (1997), vol. 1294 of *Lecture Notes in Computer Science*, Springer, pp. 513–525.

[21] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. Higher-Order Threshold Implementations. In *ASIACRYPT (2)* (2014), vol. 8874 of *Lecture Notes in Computer Science*, Springer, pp. 326–343.

[22] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. A more efficient AES threshold implementation. In *AFRICACRYPT* (2014), vol. 8469 of *Lecture Notes in Computer Science*, Springer, pp. 267–284.

[23] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Trans. on CAD of Integrated Circuits and Systems 34*, 7 (2015), 1188–1200.

[24] Bloem, R., Gross, H., Iusupov, R., Könighofer, B., Mangard, S., and Winter, J. Formal verification of masked hardware implementations in the presence of glitches. In *EUROCRYPT (2)* (2018), vol. 10821 of *Lecture Notes in Computer Science*, Springer, pp. 321–353.

[25] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. PRESENT: an ultra-lightweight block cipher. In *CHES* (2007), vol. 4727 of *Lecture Notes in Computer Science*, Springer, pp. 450–466.

[26] Boneh, D., DeMillo, R. A., and Lipton, R. J. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT* (1997), vol. 1233 of *Lecture Notes in Computer Science*, Springer, pp. 37–51.

[27] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knezevic, M., Knudsen, L. R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S. S., and Yalçın, T. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In *ASIACRYPT* (2012), vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 208–225.

[28] Bos, J. W., Hubain, C., Michiels, W., and Teuwen, P. Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough. In *CHES* (2016), vol. 9813 of *Lecture Notes in Computer Science*, Springer, pp. 215–236.

[29] Boscher, A., and Handschuh, H. Masking Does Not Protect Against Differential Fault Attacks. In *FDTC* (2008), IEEE Computer Society, pp. 35–40.

[30] Boyar, J., and Peralta, R. A small depth-16 circuit for the AES s-box. In *SEC* (2012), vol. 376 of *IFIP Advances in Information and Communication Technology*, Springer, pp. 287–298.

[31] Brier, E., Clavier, C., and Olivier, F. Correlation power analysis with a leakage model. In *CHES* (2004), vol. 3156 of *Lecture Notes in Computer Science*, Springer, pp. 16–29.

[32] Cannière, C. D., Dunkelman, O., and Knezevic, M. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In *CHES* (2009), vol. 5747 of *Lecture Notes in Computer Science*, Springer, pp. 272–288.

[33] Canright, D., and Batina, L. A Very Compact "Perfectly Masked" S-Box for AES. In *ACNS 2008* (2008), vol. 5037 of *Lecture Notes in Computer Science*, pp. 446–459.

[34] Castagnos, G., Renner, S., and Zémor, G. High-order Masking by Using Coding Theory and Its Application to AES. In *IMA Int. Conf.* (2013), vol. 8308 of *Lecture Notes in Computer Science*, Springer, pp. 193–212.

[35] Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO* (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 398–412.

[36] Chen, C., Farmani, M., and Eisenbarth, T. A tale of two shares: Why two-share threshold implementation seems worthwhile - and why it is not. In *ASIACRYPT (1)* (2016), vol. 10031 of *Lecture Notes in Computer Science*, pp. 819–843.

[37] Chen, Z., Haider, S., and Schaumont, P. Side-channel leakage in masked circuits caused by higher-order circuit effects. In *ISA* (2009), vol. 5576 of *Lecture Notes in Computer Science*, Springer, pp. 327–336.

[38] Coron, J., Prouff, E., and Roche, T. On the Use of Shamir's Secret Sharing against Side-Channel Analysis. In *CARDIS* (2012), vol. 7771 of *Lecture Notes in Computer Science*, Springer, pp. 77–90.

[39] Daemen, J. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In *CHES* (2017), vol. 10529 of *Lecture Notes in Computer Science*, Springer, pp. 137–153.

[40] Daemen, J., and Rijmen, V. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[41] De Cnudde, T., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., and Rijmen, V. Does Coupling Affect the Security of Masked Implementations? In *COSADE* (2017), vol. 10348 of *Lecture Notes in Computer Science*, Springer, pp. 1–18.

[42] De Cnudde, T., Bilgin, B., Reparaz, O., Nikov, V., and Nikova, S. Higher-Order Threshold Implementation of the AES S-Box. In *CARDIS* (2015), vol. 9514 of *Lecture Notes in Computer Science*, Springer, pp. 259–272.

[43] De Cnudde, T., Bilgin, B., Reparaz, O., and Nikova, S. Higher-Order Glitch Resistant Implementation of the PRESENT S-Box. In *BalkanCryptSec* (2014), vol. 9024 of *Lecture Notes in Computer Science*, Springer, pp. 75–93.

[44] De Cnudde, T., Ender, M., and Moradi, A. Hardware Masking, Revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2018*, 2 (2018), 123–148.

[45] De Cnudde, T., and Nikova, S. More Efficient Private Circuits II through Threshold Implementations. In *FDTC* (2016), IEEE Computer Society, pp. 114–124.

[46] De Cnudde, T., and Nikova, S. Securing the PRESENT Block Cipher Against Combined Side-Channel Analysis and Fault Attacks. *IEEE Trans. VLSI Syst. 25*, 12 (2017), 3291–3301.

[47] De Cnudde, T., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., and Rijmen, V. Masking AES with d+1 Shares in Hardware. In *CHES* (2016), vol. 9813 of *Lecture Notes in Computer Science*, Springer, pp. 194–212.

[48] Duan, C., LaMeres, B. J., and Khatri, S. P. *On and off-chip crosstalk avoidance in VLSI design*. Springer.

[49] Duc, A., Faust, S., and Standaert, F. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In *EUROCRYPT (1)* (2015), vol. 9056 of *Lecture Notes in Computer Science*, Springer, pp. 401–429.

[50] Dyrkolbotn, G. O., Wold, K., and Snekkenes, E. Security implications of crosstalk in switching CMOS gates. In *ISC* (2010), vol. 6531 of *Lecture Notes in Computer Science*, Springer, pp. 269–275.

[51] Dyrkolbotn, G. O., Wold, K., and Snekkenes, E. Layout dependent phenomena A new side-channel power model. *JCP 7*, 4 (2012), 827–837.

[52] Dziembowski, S., and Faust, S. Leakage-resilient circuits without computational assumptions. In *TCC* (2012), vol. 7194 of *Lecture Notes in Computer Science*, Springer, pp. 230–247.

[53] Dziembowski, S., and Pietrzak, K. Leakage-Resilient Cryptography. In *FOCS* (2008), IEEE Computer Society, pp. 293–302.

[54] Endo, S., Li, Y., Homma, N., Sakiyama, K., Ohta, K., and Aoki, T. An efficient countermeasure against fault sensitivity analysis using configurable delay blocks. In *FDTC* (2012), IEEE Computer Society, pp. 95–102.

[55] Endo, S., Li, Y., Homma, N., Sakiyama, K., Ohta, K., Fujimoto, D., Nagata, M., Katashita, T., Danger, J., and Aoki, T. A silicon-level countermeasure against fault sensitivity analysis and its evaluation. *IEEE Trans. VLSI Syst. 23*, 8 (2015), 1429–1438.

[56] Endo, S., Sugawara, T., Homma, N., Aoki, T., and Satoh, A. An on-chip glitchy-clock generator for testing fault injection attacks. *J. Cryptographic Engineering 1*, 4 (2011), 265–270.

[57] Endo, S., Sugawara, T., Homma, N., Aoki, T., and Satoh, A. A configurable on-chip glitchy-clock generator for fault injection experiments. *IEICE Transactions 95-A*, 1 (2012), 263–266.

[58] Faust, S., Grosso, V., Pozo, S. M. D., Paglialonga, C., and Standaert, F. Composable masking schemes in the presence of physical defaults and the robust probing model. *IACR Cryptology ePrint Archive 2017* (2017), 711.

[59] Ferrigno, J., and Hlavác, M. When AES blinks: introducing optical side channel. *IET Information Security 2*, 3 (2008), 94–98.

[60] Genkin, D., Shamir, A., and Tromer, E. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In *CRYPTO (1)* (2014), vol. 8616 of *Lecture Notes in Computer Science*, Springer, pp. 444–461.

[61] Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., and Rabin, T. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In *TCC* (2004), vol. 2951 of *Lecture Notes in Computer Science*, Springer, pp. 258–277.

[62] Ghalaty, N. F., Aysu, A., and Schaumont, P. Analyzing and eliminating the causes of fault sensitivity analysis. In *DATE* (2014), European Design and Automation Association, pp. 1–6.

[63] Ghalaty, N. F., Yuce, B., Taha, M. M. I., and Schaumont, P. Differential fault intensity analysis. In *FDTC* (2014), IEEE Computer Society, pp. 49–58.

[64] Ghoshal, A., and De Cnudde, T. Several Masked Implementations of the Boyar-Peralta AES S-Box. In *INDOCRYPT* (2017), vol. 10698 of *Lecture Notes in Computer Science*, Springer, pp. 384–402.

[65] Gilbert Goodwill, B. J., Jaffe, J., Rohatgi, P., et al. A testing methodology for side-channel resistance validation.

[66] Goddard, Z. N., LaJeunesse, N., and Eisenbarth, T. Power analysis of the t-private logic style for FPGAs. In *HOST* (2015), IEEE Computer Society, pp. 68–71.

[67] Goubin, L., and Martinelli, A. Protecting AES with Shamir's Secret Sharing Scheme. In *CHES* (2011), vol. 6917 of *Lecture Notes in Computer Science*, Springer, pp. 79–94.

[68] Goubin, L., and Patarin, J. DES and differential power analysis (the "duplication" method). In *CHES* (1999), vol. 1717 of *Lecture Notes in Computer Science*, Springer, pp. 158–172.

[69] Goudarzi, D., and Rivain, M. How Fast Can Higher-Order Masking Be in Software? In *EUROCRYPT 2017* (2017), vol. 10210 of *Lecture Notes in Computer Science*, pp. 567–597.

[70] Gross, H., and Mangard, S. Reconciling d+1 masking in hardware and software. In *CHES* (2017), vol. 10529 of *Lecture Notes in Computer Science*, Springer, pp. 115–136.

[71] Gross, H., Mangard, S., and Korak, T. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. *IACR Cryptology ePrint Archive 2016* (2016), 486.

[72] Gross, H., Mangard, S., and Korak, T. An efficient side-channel protected AES implementation with arbitrary protection order. In *CT-RSA* (2017), vol. 10159 of *Lecture Notes in Computer Science*, Springer, pp. 95–112.

[73] Grosso, V., Standaert, F., and Faust, S. Masking vs. multiparty computation: how large is the gap for aes? *J. Cryptographic Engineering 4*, 1 (2014), 47–57.

[74] Guntur, H., Ishii, J., and Satoh, A. Side-channel attack user reference architecture board SAKURA-G. In *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)* (2014), IEEE, pp. 271–274.

[75] Hutter, M., and Schmidt, J. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS* (2013), vol. 8419 of *Lecture Notes in Computer Science*, Springer, pp. 219–235.

[76] Ishai, Y., Prabhakaran, M., Sahai, A., and Wagner, D. Private Circuits II: keeping secrets in tamperable circuits. In *EUROCRYPT* (2006), vol. 4004 of *Lecture Notes in Computer Science*, Springer, pp. 308–327.

[77] Ishai, Y., Sahai, A., and Wagner, D. Private Circuits: Securing hardware against probing attacks. In *CRYPTO* (2003), vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 463–481.

[78] Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers. Standard, International Organization for Standardization, Geneva, CH, Jan. 2012.

[79] Karpovsky, M. G., Kulikowski, K. J., and Taubin, A. Differential fault analysis attack resistant architectures for the advanced encryption standard. In *CARDIS* (2004), vol. 153 of *IFIP*, Kluwer/Springer, pp. 177–192.

[80] Karri, R., and Wu, K. Algorithm level re-computing using implementation diversity: a register transfer level concurrent error detection technique. *IEEE Trans. VLSI Syst. 10*, 6 (2002), 864–875.

[81] Knudsen, J. Nangate 45nm open cell library.

[82] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. Spectre Attacks: Exploiting Speculative Execution. *meltdownattack.com* (2018).

[83] Kocher, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO* (1996), vol. 1109 of *Lecture Notes in Computer Science*, Springer, pp. 104–113.

[84] Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *CRYPTO* (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 388–397.

[85] Li, Y., Endo, S., Debande, N., Homma, N., Aoki, T., Le, T., Danger, J., Ohta, K., and Sakiyama, K. Exploring the relations between fault sensitivity and power consumption. In *COSADE* (2013), vol. 7864 of *Lecture Notes in Computer Science*, Springer, pp. 137–153.

[86] Li, Y., Ohta, K., and Sakiyama, K. Toward effective countermeasures against an improved fault sensitivity analysis. *IEICE Transactions 95-A*, 1 (2012), 234–241.

[87] Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., and Ohta, K. Fault sensitivity analysis. In *CHES* (2010), vol. 6225 of *Lecture Notes in Computer Science*, Springer, pp. 320–334.

[88] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. Meltdown. *meltdownattack.com* (2018).

[89] Lomné, V., Roche, T., and Thillard, A. On the Need of Randomness in Fault Attack Countermeasures - Application to AES. In *FDTC* (2012), IEEE Computer Society, pp. 85–94.

[90] Mangard, S., Oswald, E., and Popp, T. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[91] Mangard, S., Pramstaller, N., and Oswald, E. Successfully attacking masked AES hardware implementations. In *CHES* (2005), vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 157–171.

[92] Mangard, S., and Schramm, K. Pinpointing the side-channel leakage of masked AES hardware implementations. In *CHES* (2006), vol. 4249 of *Lecture Notes in Computer Science*, Springer, pp. 76–90.

[93] Melzani, F., and Palomba, A. Enhancing fault sensitivity analysis through templates. In *HOST* (2013), IEEE Computer Society, pp. 25–28.

[94] Menezes, A., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography*. CRC Press, 1996.

[95] Messerges, T. S. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *CHES* (2000), vol. 1965 of *Lecture Notes in Computer Science*, Springer, pp. 238–251.

[96] Mischke, O., Moradi, A., and Güneysu, T. Fault sensitivity analysis meets zero-value attack. In *FDTC* (2014), IEEE Computer Society, pp. 59–67.

[97] Moll, F., Roca, M., and Isern, E. Analysis of dissipation energy of switching digital CMOS gates with coupled outputs. *Microelectronics Journal 34*, 9 (2003), 833–842.

[98] Moradi, A. Side-channel leakage through static power - should we care about in practice? In *CHES* (2014), vol. 8731 of *Lecture Notes in Computer Science*, Springer, pp. 562–579.

[99] Moradi, A., and Mischke, O. On the simplicity of converting leakages from multivariate to univariate - (case study of a glitch-resistant masking scheme). In *CHES* (2013), vol. 8086 of *Lecture Notes in Computer Science*, Springer, pp. 1–20.

[100] Moradi, A., Mischke, O., and Eisenbarth, T. Correlation-Enhanced Power Analysis Collision Attack. In *CHES 2010* (2010), vol. 6225 of *Lecture Notes in Computer Science*, Springer, pp. 125–139.

[101] Moradi, A., Mischke, O., and Paar, C. One attack to rule them all: Collision timing attack versus 42 AES ASIC cores. *IEEE Trans. Computers 62*, 9 (2013), 1786–1798.

[102] Moradi, A., Mischke, O., Paar, C., Li, Y., Ohta, K., and Sakiyama, K. On the power of fault sensitivity analysis and collision side-channel attacks in a combined setting. In *CHES* (2011), vol. 6917 of *Lecture Notes in Computer Science*, Springer, pp. 292–311.

[103] Moradi, A., Poschmann, A., Ling, S., Paar, C., and Wang, H. Pushing the limits: A very compact and a threshold implementation of AES. In *EUROCRYPT* (2011), vol. 6632 of *Lecture Notes in Computer Science*, Springer, pp. 69–88.

[104] Nawaz, K., Kamel, D., Standaert, F., and Flandre, D. Scaling Trends for Dual-Rail Logic Styles Against Side-Channel Attacks: A Case-Study. In *COSADE* (2017), vol. 10348 of *Lecture Notes in Computer Science*, Springer, pp. 19–33.

[105] Nikova, S., Rechberger, C., and Rijmen, V. Threshold implementations against side-channel attacks and glitches. In *ICICS* (2006), vol. 4307 of *Lecture Notes in Computer Science*, Springer, pp. 529–545.

[106] Nikova, S., Rijmen, V., and Schläffer, M. Secure hardware implementation of non-linear functions in the presence of glitches. In *ICISC* (2008), vol. 5461 of *Lecture Notes in Computer Science*, Springer, pp. 218–234.

[107] Nikova, S., Rijmen, V., and Schläffer, M. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology 24*, 2 (2011), 292–321.

[108] Oswald, E., Mangard, S., Pramstaller, N., and Rijmen, V. A side-channel analysis resistant description of the AES s-box. In *FSE* (2005), vol. 3557 of *Lecture Notes in Computer Science*, Springer, pp. 413–423.

[109] Picek, S., Yang, B., Rozic, V., Vliegen, J., Winderickx, J., Cnudde, T. D., and Mentens, N. PRNGs for Masking Applications and Their Mapping to Evolvable Hardware. In *CARDIS* (2016), vol. 10146 of *Lecture Notes in Computer Science*, Springer, pp. 209–227.

[110] Poschmann, A., Moradi, A., Khoo, K., Lim, C., Wang, H., and Ling, S. Side-channel resistant crypto for less than 2,300 GE. *J. Cryptology 24*, 2 (2011), 322–345.

[111] Poussier, R., Guo, Q., Standaert, F., Carlet, C., and Guilley, S. Connecting and improving direct sum masking and inner product masking. In *CARDIS* (2017), vol. 10728 of *Lecture Notes in Computer Science*, Springer, pp. 123–141.

[112] Prouff, E., and Rivain, M. Masking against side-channel attacks: A formal security proof. In *EUROCRYPT* (2013), vol. 7881 of *Lecture Notes in Computer Science*, Springer, pp. 142–159.

[113] Prouff, E., and Roche, T. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In *CHES* (2011), vol. 6917 of *Lecture Notes in Computer Science*, Springer, pp. 63–78.

[114] Rabaey, J. M. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[115] Rakotomalala, H., Ngo, X. T., Najm, Z., Danger, J., and Guilley, S. Private Circuits II versus fault injection attacks. In *ReConFig* (2015), IEEE, pp. 1–9.

[116] Reparaz, O. Detecting flawed masking schemes with leakage detection tests. In *FSE* (2016), vol. 9783 of *Lecture Notes in Computer Science*, Springer, pp. 204–222.

[117] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., and Verbauwhede, I. Consolidating Masking Schemes. In *CRYPTO (1)* (2015), vol. 9215 of *Lecture Notes in Computer Science*, Springer, pp. 764–783.

[118] Reparaz, O., De Meyer, L., Bilgin, B., Arribas, V., Nikova, S., Nikov, V., and Smart, N. P. CAPA: the spirit of beaver against physical attacks. In *CRYPTO (1)* (2018), vol. 10991 of *Lecture Notes in Computer Science*, Springer, pp. 121–151.

[119] Rijmen, V. Efficient Implementation of the Rijndael S-box.

[120] Rivain, M., and Prouff, E. Provably secure higher-order masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings* (2010), S. Mangard and F. Standaert, Eds., vol. 6225 of *Lecture Notes in Computer Science*, Springer, pp. 413–427.

[121] Roche, T., and Prouff, E. Higher-order glitch free implementation of the AES using secure multi-party computation protocols - extended version. *J. Cryptographic Engineering 2*, 2 (2012), 111–127.

[122] Ronen, E., Shamir, A., Weingarten, A., and O'Flynn, C. IoT goes nuclear: Creating a zigbee chain reaction. *IEEE Security & Privacy 16*, 1 (2018), 54–62.

[123] Roy, D. B., Bhasin, S., Guilley, S., Danger, J., and Mukhopadhyay, D. From theory to practice of Private Circuit: A cautionary note. In *ICCD* (2015), IEEE Computer Society, pp. 296–303.

[124] Schellenberg, F., Finkeldey, M., Gerhardt, N., Hofmann, M., Moradi, A., and Paar, C. Large laser spots and fault sensitivity analysis. In *HOST* (2016), IEEE Computer Society, pp. 203–208.

[125] Schellenberg, F., Gnad, D. R. E., Moradi, A., and Tahoori, M. B. An inside job: Remote power analysis attacks on FPGAs. In *DATE* (2018), IEEE, pp. 1111–1116.

[126] Schmidt, J., Plos, T., Kirschbaum, M., Hutter, M., Medwed, M., and Herbst, C. Side-channel leakage across borders. In *CARDIS* (2010), vol. 6035 of *Lecture Notes in Computer Science*, Springer, pp. 36–48.

[127] Schneider, T., and Moradi, A. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In *CHES* (2015), vol. 9293 of *Lecture Notes in Computer Science*, Springer, pp. 495–513.

[128] Schneider, T., Moradi, A., and Güneysu, T. Arithmetic addition over boolean masking - towards first- and second-order resistance in hardware. In *ACNS* (2015), vol. 9092 of *Lecture Notes in Computer Science*, Springer, pp. 559–578.

[129] Schneider, T., Moradi, A., and Güneysu, T. ParTI - towards combined hardware countermeasures against side-channel and fault-injection attacks. In *CRYPTO (2)* (2016), vol. 9815 of *Lecture Notes in Computer Science*, Springer, pp. 302–332.

[130] Seker, O., Eisenbarth, T., and Steinwandt, R. Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks. *IACR Cryptology ePrint Archive 2017* (2017), 269.

[131] Seker, O., Fernandez-Rubio, A., Eisenbarth, T., and Steinwandt, R. Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 3 (2018), 394–430.

[132] Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (1979), 612–613.

[133] Skorobogatov, S. P., and Anderson, R. J. Optical fault induction attacks. In *CHES* (2002), vol. 2523 of *Lecture Notes in Computer Science*, Springer, pp. 2–12.

[134] Tiri, K., Akmal, M., and Verbauwhede, I. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Proceedings of the 28th European Solid-State Circuits Conference* (Sept 2002), pp. 403–406.

[135] Tiri, K., and Verbauwhede, I. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *DATE* (2004), IEEE Computer Society, pp. 246–251.

[136] Trichina, E., Korkishko, T., and Lee, K. Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results. In *AES Conference* (2004), vol. 3373 of *Lecture Notes in Computer Science*, Springer, pp. 113–127.

[137] Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., and Standaert, F. Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In *ASIACRYPT* (2012), vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 740–757.

[138] Wang, G., and Wang, S. Differential fault analysis on PRESENT key schedule. In *CIS* (2010), IEEE Computer Society, pp. 362–366.

[139] Wegener, F., and Moradi, A. A first-order SCA resistant AES without fresh randomness. In *COSADE* (2018), vol. 10815 of *Lecture Notes in Computer Science*, Springer, pp. 245–262.

[140] Wild, A., Moradi, A., and Güneysu, T. Evaluating the duplication of dual-rail precharge logics on fpgas. In *COSADE* (2015), vol. 9064 of *Lecture Notes in Computer Science*, Springer, pp. 81–94.

[141] Xilinx. Constraints guide 10.1. `http://www.xilinx.com/itp/xilinx10/books/docs/cgd/cgd.pdf`.

[142] Yen, S., Kim, S., Lim, S., and Moon, S. RSA speedup with residue number system immune against hardware fault cryptanalysis. In *ICISC* (2001), vol. 2288 of *Lecture Notes in Computer Science*, Springer, pp. 397–413.

[143] Yuce, B., Ghalaty, N. F., and Schaumont, P. TVVF: estimating the vulnerability of hardware cryptosystems against timing violation attacks. In *HOST* (2015), IEEE Computer Society, pp. 72–77.

[144] Zussa, L., Exurville, I., Dutertre, J., Rigaud, J., Robisson, B., Tria, A., and Clédière, J. Evidence of an information leakage between logically independent blocks. In *CS2@HiPEAC* (2015), ACM, pp. 25–30.

List of Publications

Journals

1. De Cnudde, T., and Nikova, S. Securing the PRESENT Block Cipher Against Combined Side-Channel Analysis and Fault Attacks. *IEEE Trans. VLSI Syst. 25*, 12 (2017), 3291–3301

Conferences

1. Arribas, V., Cnudde, T. D., and Šijačić, D. Glitch-Resistant Masking Schemes as Countermeasure Against Fault Sensitivity Analysis. In *FDTC* (2018), IEEE Computer Society, pp. 1–8

2. De Cnudde, T., Ender, M., and Moradi, A. Hardware Masking, Revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2018*, 2 (2018), 123–148

3. Ghoshal, A., and De Cnudde, T. Several Masked Implementations of the Boyar-Peralta AES S-Box. In *INDOCRYPT* (2017), vol. 10698 of *Lecture Notes in Computer Science*, Springer, pp. 384–402

4. De Cnudde, T., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., and Rijmen, V. Does Coupling Affect the Security of Masked Implementations? In *COSADE* (2017), vol. 10348 of *Lecture Notes in Computer Science*, Springer, pp. 1–18

5. Picek, S., Yang, B., Rozic, V., Vliegen, J., Winderickx, J., Cnudde, T. D., and Mentens, N. PRNGs for Masking Applications and Their Mapping to Evolvable Hardware. In *CARDIS* (2016), vol. 10146 of *Lecture Notes in Computer Science*, Springer, pp. 209–227

6. De Cnudde, T., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., and Rijmen, V. Masking AES with d+1 Shares in Hardware. In *CHES* (2016), vol. 9813 of *Lecture Notes in Computer Science*, Springer, pp. 194–212

7. De Cnudde, T., and Nikova, S. More Efficient Private Circuits II through Threshold Implementations. In *FDTC* (2016), IEEE Computer Society, pp. 114–124

8. De Cnudde, T., Bilgin, B., Reparaz, O., Nikov, V., and Nikova, S. Higher-Order Threshold Implementation of the AES S-Box. In *CARDIS* (2015), vol. 9514 of *Lecture Notes in Computer Science*, Springer, pp. 259–272

9. De Cnudde, T., Bilgin, B., Reparaz, O., and Nikova, S. Higher-Order Glitch Resistant Implementation of the PRESENT S-Box. In *BalkanCryptSec* (2014), vol. 9024 of *Lecture Notes in Computer Science*, Springer, pp. 75–93



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
IMEC-COSIC
Kasteelpark Arenberg 10, box 2452
B-3001 Leuven
thomas.decnudde@esat.kuleuven.be