Threshold Implementations
As Countermeasure Against
Higher-Order Differential Power Analysis

Begül Bilgin

Threshold Implementations
As Countermeasure Against Higher-Order Differential
Power Analysis

Begül Bilgin

Composition of the Examination Committee:

Prof. dr.	P. Wollants	KU Leuven (chairman)
Prof. dr.	H. Brinksma	Universiteit Twente (chairman)
Prof. dr. ir.	V. Rijmen	KU Leuven (supervisor)
Prof. dr.	P.H. Hartel	Universiteit Twente (supervisor)
Dr.	S. Nikova	KU Leuven (co-supervisor)
Prof. dr.	S. Etalle	Universiteit Twente and
		Technische Universiteit Eindhoven
Prof. dr.	W. Jonker	Universiteit Twente
Prof. dr. ir.	B. Preneel	KU Leuven
Prof. dr. ir.	F. X. Standaert	Université Catholique de Louvain
Prof. dr. ir.	L. Van der Perre	Imec and KU Leuven
Prof. dr. ir.	I. Verbauwhede	KU Leuven

THRESHOLD IMPLEMENTATIONS
As Countermeasure Against Higher-Order Differential Power Analysis

DISSERTATION

to obtain

the degree of doctor at the University of Twente and KU Leuven
on the authority of the rector magnificus,
prof. dr. H. Brinksma and prof. dr. R. Torfs,
on account of the decision of the graduation committee,
to be publicly defended
on Wednesday, 13$^{\text{th}}$ of May 2015 at 12.45

by

Begül Bilgin

born on 28$^{\text{th}}$ of October 1986,
in Ankara, Turkey

The dissertation is approved by:

Prof. dr.   P.H. Hartel (promotor)
Prof. dr. ir.   V. Rijmen (promotor)

*"We must find time to stop and
thank the people who make a
difference in our lives."*
                — John F. Kennedy

Acknowledgments

When I learned that I have been given the opportunity to start this journey, I knew that the following four years will be challenging, enlightening and stormy (literally). However, I could not have imagined to be surrounded by such amazing people. Thank you all who influenced me one way or another on the way to this dissertation. That said, I believe some people deserve a special acknowledgment.

Dear Svetla, as my daily supervisor you have always believed in me. This journey would not be as smooth as it was if you did not have my back. It was very comforting to know that I can ask for help not only to solve our Boolean puzzles but also to deal with any off-research problem since our first tea session in Hotel Drienerburght.

Dear Vincent, thank you for always pushing me to the right direction in your uniquely sarcastic way and for your supportive supervision. You have provided me with unique opportunities and trusted me (maybe even more than I deserve). I will always (not try but) do my best not to disappoint you.

Dear Pieter, you have always challenged me with the most unimaginable questions. I will always be grateful for the freedom you have offered.

And Benedikt, my honorary supervisor, thank you for teaching me to always read with criticism. Your patience and guidance taught me that each failure takes me one step closer to success.

I think I was extremely lucky to have not one but four incredible supervisors who read my each and every sentence, taught me how to do research and how to write and had their doors open for brainstorming all the time.

I would like to extend my gratitude to the committee members, especially Ingrid, for taking their time to read my thesis and provide me with valuable comments and to my co-authors for all the fruitful discussions.

I had the opportunity to share offices with whom I enjoyed their friendship and equally respect. COSIC 01.62 members who showed me the beauty of hands-on analysis and answered all my engineering questions patiently and my Turkish bride and girl power Dina from EWI 3032, I will always be in your debt.

Leaving all the paper work aside, it was a pleasure to be a part of two (research) families and to live in this beautiful city. An attempt to count everyone would be foolish. I will rather thank my lunch buddies, Friday beer and secret mojito fans, Turkish mafia, ex-Co6, chocolate and candy lovers, conference companions, my dearest Pela and 15h coffee mates from DIES.

İlk öğretmenlerim saydığım annem, babam, anneannem ve dedem; bu başarıların hiç biri sizin verdiğiniz emek ve eğitim olmasa olmazdı. Ne kadar zor olursa olsun bu yolu sizden uzak da olsa ilerlememde en büyük destekçim olduğunuz için size minnettarım. Bahadır, örnek olma uğruna en büyük çalısma motivasyonum, favori erkeğim, kardeşim, anime ortağım, . . . ODTÜ'yü beraber yaşayamayacak olmamıza rağmen "git" dediğin için teşekkürler. Dayım, en başta Turgut hoca ile beraber bana kriptografi yolunu gösterdiğiniz için ve her zaman bir mail uzağımda olduğunuzu hissettirdiğiniz için teşekkürler.

Gökhan'ım; Paulo Coelho "She didn't need to understand the meaning of life; it was enough to find someone who did, and then fall asleep in his arms and sleep as a child sleeps, knowing that someone stronger than you is protecting you from all evil and all danger" dediğinde bizi anlatmış sanki. Mutlu, sinirli, heyecanlı, stresli her türlü anımda yanımda olduğun, bana güvendiğin ve her şeyden önemlisi buralara geldiğin için teşekkürler. Eşsizsin . . .

Sevgili Öztürk ailesi, verdiğimiz kararları her zaman desteklediğiniz için sizlere çok teşekkür ederim.

<div align="right">

Begül Bilgin
April 2015, Leuven

</div>

Abstract

Embedded devices are used pervasively in a wide range of applications some of which require cryptographic algorithms in order to provide security. Today's standardized algorithms are secure in the black-box model where an adversary has access to several inputs and/or outputs of the algorithm. However, sensitive information, such as the secret key used in the algorithm, can be derived from the physical leakage of these devices in the so called gray-box model. In a passive, non-invasive attack scenario, this physical leakage can be execution time, power consumption or electromagnetic radiation. The most common attack based on these leakages is differential power analysis (DPA) since the equipment required for such an attack is relatively cheap and the success rate of the attack is high on unprotected implementations. DPA exploits the correlation between the instantaneous power consumption of a device and the intermediate results of a cryptographic algorithm.

Different countermeasures applied on various levels of the circuit have been proposed to prevent DPA. Some of these countermeasures focus on limiting the amount of power traces gathered from the cryptographic algorithm under attack using the same key. Some others aim at decreasing the signal-to-noise ratio in order to make the aforementioned correlation invisible. The final countermeasure group, which we study, randomizes the leakage depending on the sensitive information by randomizing the intermediate values of an algorithm in order to break the correlation. This powerful approach, which is called masking, provides provable security under certain leakage assumptions even if infeasibly many number of traces are analyzed. In standard masking, the model requires that there is no occurrence of unintended switching at the input or output of logic gates, the so called glitch. However, glitches are unavoidable in circuits using standard cells based on, for example, the most common hardware technology CMOS. This glitchy behavior typically results in the leakage of unintended information. There exist only two masking schemes that are proven secure even in the presence of glitches so far, namely by Nikova et

al. from ICICS'06 and by Prouff et al. from CHES'11. The former, named threshold implementation, requires significantly smaller area and uses much less randomness compared to the method by Prouff et al.

Threshold implementation (TI) is based on secret sharing and multi-party computation in which sensitive variables and functions using these variables are divided into $s > d$ shares, such that knowledge of d of these shares does not reveal the secret information. An analysis using a nonlinear combination of leakages derived from d of these shares or their calculation is called a dth-order DPA. TI relies on four properties, namely correctness, non-completeness, uniformity of the shared variables and uniformity of the shared functions. It provides provable security even in the presence of glitches given the assumption that the overall leakage of the device is a linear combination of leakages caused by different shares and their calculation. This is both a common and realistic assumption made by most of the masking schemes. Achieving all four properties of TI for linear functions is straight-forward. On the other hand, it can be a challenging task when nonlinear functions, such as the S-boxes of symmetric key algorithms, are considered. Satisfying all the properties can impose using extra randomness or increasing the number of shares. Both of these solutions imply an increase of resources required by TI.

The contribution of this thesis is two-fold. In the first part of the thesis, we introduce the theory for generating d^{th}-order TI which can counteract d^{th}-order DPA. The early works of TI provide security against first-order DPA attacks. However, it has been shown that second-order attacks are also feasible even though the amount of traces required for a successful attack increases exponentially in the noise standard deviation. Therefore, increasing the security using higher-order TI is valuable. In addition, we confirm the claimed security by analyzing a second-order TI of the block cipher KATAN.

The resource requirements form a limiting factor for countermeasures especially on lightweight devices. In the second part of the thesis, we examine area-randomness-security trade-offs during a TI. In order to do that, we first investigate all 3×3 and 4×4, and some cryptographically significant classes of 5×5 and 6×6 invertible S-boxes. We use the gathered knowledge to choose S-boxes during the designs of the authenticated encryption algorithms FIDES and PRIMATEs such that the area footprints of their TIs are small. Then, we extend our research to the TIs of standardized symmetric-key algorithms AES and SHA-3 with detailed investigation on the trade-offs.

Beknopte samenvatting

Geïntegreerde elektronica wordt tegenwoordig gebruikt in een breed scala aan toepassingen. Sommige van die toepassingen vereisen cryptografische algoritmes voor beveiliging. Gestandaardiseerde cryptografische algoritmes die tegenwoordig gebruikt worden zijn veilig in het zwarte doos model, waarbij een aanvaller enkel toegang heeft tot de inputs en/of outputs van het algoritme. Gevoelige informatie, zoals de geheime sleutel die door het algoritme wordt gebruikt, kan echter afgeleid worden uit de fysisch gelekte informatie van een apparaat in het zogenaamde grijze doos model. In een passief, niet-invasief aanvalsscenario, kan deze fysische informatie bestaan uit bijvoorbeeld uitvoeringstijd, vermogensverbruik of elektromagnetische straling. De meest voorkomende aanval die gebruik maakt van zulke lekken is differentiële vermogensanalyse (DPA), omdat de toestellen die nodig zijn om zo een aanval uit te voeren relatief goedkoop zijn. DPA maakt gebruik van de correlatie tussen het ogenblikkelijk vermogensverbruik van het toestel en de tussenresultaten in het cryptografisch algoritme.

Er zijn verschillende voorstellen voor tegenmaatregelen toegepast op verscheidene circuitniveaus om DPA tegen te gaan. Sommige van deze methodes trachten de hoeveelheid vermogensmetingen te beperken, terwijl het algoritme een en dezelfde sleutel gebruikt. Andere trachten de signaal-ruisverhouding van het circuit te verlagen, zodat de eerder vermelde correlaties niet meer meetbaar zijn. De laatste groep bestudeerde tegenmaatregelen doet de tussentijdse resultaten van het algoritme willekeurige waardes aannemen afhankelijk van de gevoelige informatie, om zo de correlaties te verbreken. Deze krachtige methode, die masking genoemd wordt, kan bewijsbaar veilige bescherming bieden onder bepaalde aannames in verband met de lekken, zelfs indien een heel groot aantal metingen op het circuit gedaan worden. Bij standaard masking gaat men er in het model van uit dat logische poorten in het circuit geen ongewenste overgangen maken, zogenaamde glitches. Glitches zijn echter niet te voorkomen met standaard cellen gebaseerd op, bijvoorbeeld, de meest gebruikte hardware technologie, CMOS. Deze glitches zorgen er doorgaans voor dat er ongewenste informatie uitlekt. Momenteel zijn er slechts twee masking schema's die bewijsbaar veilig zijn zelfs bij het voorkomen van glitches, namelijk

door Nikova et al. van ICICS '06 en door Prouff et al. van CHES '11. Dat eerste schema, zogenaamde threshold implementatie, vereist significant minder oppervlakte en gebruikt veel minder willekeurige data vergeleken met het schema van Prouff *et al.*

De threshold implementatie (TI) is gebaseerd op het delen van geheimen en meerdere partijen berekeningen, waarbij gevoelige variabelen en functies die deze gebruiken gesplitst worden in $s > d$ delen, op zo een manier dat kennis van maximum d delen niet vrijgeeft wat de geheime informatie is. Een analyse die gebruik maakt van niet-lineaire combinaties van lekken afgeleid van d delen of hun berekeningen wordt d^{de} orde DPA genoemd. TI is gebaseerd op vier eigenschappen, zijnde correctheid, niet-compleetheid, uniformiteit van de gedeelde variabelen en uniformiteit van de gedeelde functies. Het biedt bewijsbare veiligheid, zelfs in de aanwezigheid van glitches, gegeven de aanname dat de gelekte informatie van het apparaat een lineaire combinatie van de lekken van de verschillende delen en hun berekening. Dit is een standaard-, en realistische, aanname die voor de meeste masking schema's gemaakt wordt. Het is makkelijk om aan de vier eigenschappen van TI te voldoen voor lineaire functies. Voor niet-lineaire functies, zoals S-boxes in symmetrische sleutelalgoritmes, kan het echter moeilijk zijn. Om aan al deze voorwaarden te voldoen kan het nodig zijn om het aantal delen te verhogen of extra toevalbits te gebruiken. Deze beide oplossingen vereisen extra middelen voor de implementatie van TI.

De bijdrage van deze thesis is tweeledig. In het eerste deel van de thesis introduceren we de theorie nodig om een d^{de} orde TI te genereren die d^{de} orde DPA kan weerstaan. Eerder gepubliceerde versies van TI kunnen eerste orde DPA weerstaan. Het is echter aangetoond dat zulke implementaties vatbaar zijn voor tweede orde aanvallen. Het benodigde aantal metingen voor een succesvolle aanval stijgt in dat geval wel exponentieel in de standaardafwijking van de ruis. Het heeft dus zin om de veiligheid te verhogen door middel van hogere orde TI. Daarenboven bewijzen we de beloofde veiligheid door analyse van een tweede orde TI implementatie voor het blokcijfer KATAN.

De benodigde middelen, zoals bijvoorbeeld oppervlakte, zijn een beperkende factor voor tegenmaatregelen, vooral voor geïntegreerde elektronica. In het tweede deel van de thesis onderzoeken we trade-offs tussen oppervlakte, toevalsbits en veiligheid in TI. Om dat te kunnen doen, onderzoeken we eerst alle 3×3, 4×4 en enkel cryptografisch belangrijke 5×5 en 6×6 inverteerbare S-boxes. We gebruiken de vergaarde kennis bij het uitkiezen van S-boxes voor het ontwerp van de geauthentiseerde encryptie algoritmes Fides en PRIMATEs, zodat de grootte van de TI implementatie klein is. Daarna breiden we ons onderzoek uit naar TIs voor de gestandaardiseerde symmetrische sleutel algoritmes AES en SHA-3, met een gedetailleerd onderzoek naar de trade-offs.

Contents

List of Figures

List of Tables

Abbreviations

AES	Advanced Encryption Standard
ASIC	Application-Specific Integrated Circuit
CEPACA	Correlation-Enhanced Power Analysis Collision Attack
CMOS	Complementary Metal Oxide Semiconductor
CPA	Correlation Power Analysis
CT	Correction Terms
DES	Data Encryption Standard
DFF	D Flip-Flop
DoM	Difference of Means
DPA	Differential Power Analysis
EM	ElectroMagnetic
FPGA	Field-Programmable Gate Array
GE	Gate Equivalent
GND	Ground
HD	Hamming Distance
HW	Hamming Weight
I/O	Input / Output
PRNG	Pseudo-Random Number Generator
RFID	Radio-Frequency IDentification
RNG	Random Number Generator

SCA	Side-Channel Analysis
SFF	Scan Flip-Flops
SNR	Signal-to-Noise Ratio
SPA	Simple Power Analysis
SPN	Substitution Permutation Network
TI	Threshold Implementation
WDDL	Wave Dynamic Differential Logic

"If you can't explain it simply, you
don't understand it well enough."
— Albert Einstein

1
Introduction

There exist about 30 embedded devices per person [50] in a developed country. Each car and electronic household alone possess more than 20 such devices in addition to computing devices, phones and payment cards. Predictions show that the use of embedded devices will increase 10% every year parallel to the increase in commercial use of smart objects. Some of these embedded devices such as *Radio-Frequency IDentification* (RFID) tags are wireless. Moreover, these devices can become extremely *lightweight* by being low-powered and very small in area. Smart cards, for instance NXP Semiconductors' Mifare SmartCard series (Mifare Classic, Mifare DESFire) which celebrated its twentieth anniversary last year with over 5 million components sold [78], are only one lightweight and battery-less example.

Depending on the application, these cards can provide confidentiality, privacy, data integrity, authentication and many other security functions. For example, the secure series of smart cards chips are used in ID cards, passports, smart meters, key cards; and can handle micro-payments. Moreover, RFID tags are widely used for medical and military purposes, and for tracking commercial products and even people. The security backbone of these these devices is the ancient art of *cryptology* (hidden word) which dates back to 2000BC [57]. Its subfields *cryptography* (hidden writing) and *cryptanalysis* act as the Yin and Yang of modern security. Advancement in one brings the necessity of advancement in the counter party.

Modern cryptographic algorithms can be viewed as mathematical functions which use an input text mostly together with an input key in order to produce a random looking string that can not be correlated with the inputs. If these algorithms use at most one (secret) key, they are called *symmetric (secret)-key algorithms*. On the other hand, if they require a second (public) key in addition to the secret key which complements it, then they are referred to as *asymmetric (public)-key algorithms*. Throughout this thesis, we consider symmetric-key algorithms.

1.1 Adversary Models

A cryptographic algorithm provides security even if all the details except the secret key is known to an adversary as suggested by the Kerckhoffs' principle [60]. Hence, the key space should be big enough to make an exhaustive search infeasible for revealing this key. The attacker's goal is to find the key which he can use to deceive the system about his identity or to capture confidential information. He can also attempt to break the system without recovering the key, however such attacks are out of the scope of this thesis.

We can classify the adversaries depending on the amount of information they have access to. In the first adversary model, the attacker approaches a cryptographic algorithm as a purely mathematical object which gives the name *black-box* to the adversary model. The attacker can use the knowledge about the algorithm together with several of its inputs and/or outputs, in order to find a weakness and reveal the key with less complexity than an exhaustive key search. This oldest adversary model, unlike the others, is independent of the implementation of the algorithm and its platform. This attack strategy, of which differential and linear cryptanalysis are famous examples [55], is a wide and still evolving research area that is not considered in this thesis. However, we note that the standardized algorithms which we work with are secure against this model with today's knowledge.

The second adversary model assumes that an attacker has access to the software implementation and has control over the platform. All the information except the secret key is transparent to the attacker naming the model *white-box* cryptography. It is even assumed that the adversary has the ability to observe the exact intermediate values of the algorithm in addition to the capability to access the memory where the secret-key is stored. This model which dates back to only 2002 [34] is out of the scope of this thesis.

In the last adversary model, the attacker targets the implementation by analyzing the device behavior during a cryptographic operation. This *gray-box* attack

model dates back to 1965 [100] and assumes that the attacker has physical access to the device. The analysis can range from tampering with the device, by temperature or voltage changes [7], or making permanent changes on the circuit [64], to simply observing the physical behavior such as timing, power consumption or electromagnetic (EM) emission [51, 62, 63]. Many modern devices include sensors to detect the former active and/or (semi)-invasive techniques upon which they kill the chip or revoke the key. An attack using the latter passive non-invasive analysis, which is called *Side-Channel Analysis* (SCA) [62], is relatively hard to detect, hence advantageous from the attacker's point of view. In this thesis, we mainly consider power analysis attacks with the note that this work can be extended to study timing and EM analysis under similar leakage assumptions.
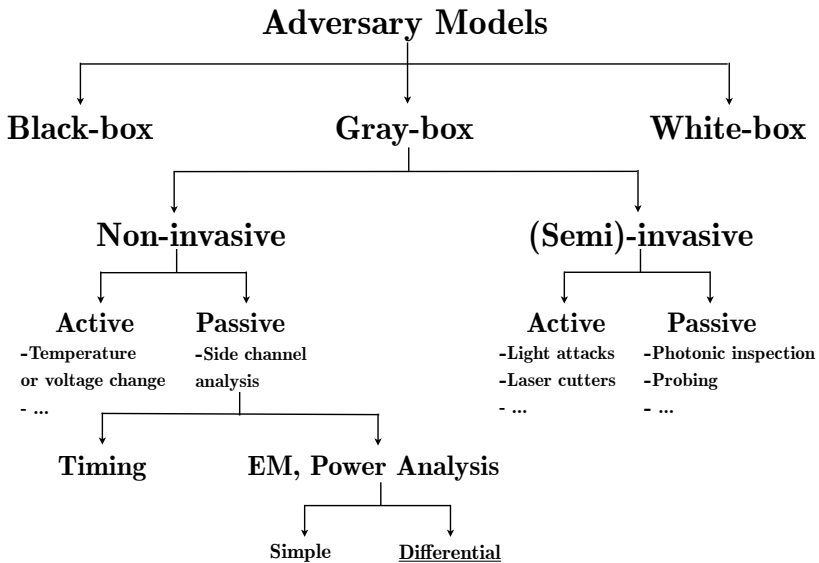
Figure 1.1: Overview of adversary models

If an adversary analyses power trace(s) from the cryptographic device collected using the same input, it is called a *Simple Power Analysis* (SPA) [63]. An adversary, using this strong model against a symmetric-key algorithm, requires a high signal-to-noise ratio (SNR) hence, can only tolerate minimal noise. Furthermore, this attack is typically impractical on hardware without a profiling phase generated from the exact device under attack prior to analysis. An alternative approach is using analysis techniques from the black-box model together with SPA [87]. However, that would require an off-line phase using complex problem solvers.

Differential Power Analysis (DPA), which was introduced in 1999 [63], requires a set of power traces collected from the device using several different inputs. In the simplest attack scenario, the attacker finds an intermediate value that depends on only a small part of the secret key, referred to as the *sub-key*, and the input. Then, he guesses this sub-key and calculates the hypothetical values of the chosen intermediate value for each known input. The power traces are grouped depending on these hypothetical values. Next, the mean trace is calculated for each group by taking the average of the traces within the group. If the guessed sub-key is incorrect, the mean traces looks similar, i.e. differs only by the factor of the noise. In contrast, if the guessed sub-key is correct, the attacker can distinguish a difference between these mean traces at the time the intermediate value is executed on the device. This particular DPA method is called *difference of means* (DoM). It can be improved by observing a correlation between the instantaneous power consumption of a device and and (hypothetical) intermediate values. Moreover, an attacker can also examine the power traces using higher statistical moments such as the variance and the skewness. DPA is applied widely today due to the simplicity of its application on an unprotected cryptographic device.

DPA of the KEELOQ key-less remote entry system, which is used in many car and garage doors [48], is a famous example of a DPA on a commercial product. The attack had a big impact since the attacker not only reveals the secret key from the remote control but also the manufacturer key which allows creating any number of valid new remote controls in less than a day. Similarly, NXP decided to discontinue MIFARE DESFire MF3ICD40, which is used in several payment and public transportation systems including the Clippercard in San Fransisco, in 2011 after being informed about a successful attack [79, 81]. In 2012, Balasch et al. showed that it takes less than half an hour to recover the secret authentication key from an Atmel CryptoMemory device that is used even for military applications [6]. Once the authentication key is revealed, an adversary can read protected contents, clone devices, or manipulate the memory. None of these examples are platform dependent. Any implementation without a countermeasure against DPA is vulnerable to similar attacks.

1.2 Motivation

The efficacy of DPA brings the necessity to find countermeasures against it. These countermeasures can be applied from the highest system level with minimum assumptions on the specific implementation of the symmetric-key algorithm to the lowest cell/instruction level. Independent of the application level, all these countermeasures try to make one or more of the ingredients to

DPA, which are power traces of computations using the same key and several different inputs, information derived from these power traces and the correlation between this information and the intermediate values of the algorithm, invisible.

The first set of countermeasures, which is typically applied at the system level, focuses on limiting the number of iterations of an algorithm using the same key attempting to make DPA impossible. However, generating and synchronizing a new secret key is highly impractical. A technique called leakage resilience relocates this problem to the protocol level [47] by introducing an algorithm to generate these keys. This countermeasure is extended, such that several different keys (chunks) are used with the same input text, confusing an attacker [69]. Nevertheless, both of these approaches drastically decrease the performance of a system.

The second set of countermeasures, which focuses on decreasing the information gathered from a power trace, offers several approaches. Targeting a constant-power implementation is one approach which typically requires special cells. There exists exceptions, such as Wave Dynamic Differential Logic (WDDL) cells [95], which can be constructed using standard CMOS logic cells. This complementary technique will not be considered as it requires cell-level investigation.

There are several ad-hoc approaches that aim to increase the noise hence decrease the SNR for the attacker to make the information therefore the correlation less visible. Introducing external noise in the side-channel, shuffling the operations or inserting dummy operations until an attack is not feasible are typical examples. Ultimately, these countermeasures become insecure with increasing computation power and attack time [46, 97].

The third set of countermeasures aims to break the correlation between the power traces and the intermediate values of the computations. Unlike ad-hoc approaches, countermeasures in this set follow the *masking* method which provides provable security in a specified model even if a large number of traces are analyzed. We study this powerful method which achieves security by randomizing the intermediate values using secret sharing. A standard d^{th}-order masking is based on representing a sensitive variable by $d + 1$ randomized variables called *shares* such that an adversary who knows at most d of these shares cannot reproduce the sensitive information.

In the early works of masking, the circuit and variables are split into two shares in a randomized manner. This randomized splitting causes the average power consumption for calculations depending on the shares of a variable to be the same for all the values of the variable. Hence, a DPA using the means of the traces as described at the end of Section 1.1 gives no information on the

particular value of the secret since the averaged traces only differ by noise in this masked scenario. This analysis is called first-order DPA since it uses first-order moments (means) of information gathered from the power traces. Increasing the information order to d produces a dth-*order* DPA. Attacking the same two-share masked implementation using the variances of the traces generated for each intermediate value, hence the second-order moment information, reveals the secret if operations on the shares are performed at the same time. This derivation of second-order moment information is equivalent to combining information from the two shares in a nonlinear manner. This reveals information from both of the shares eliminating the effect of first-order masking. It is possible to avoid second-order attacks by splitting the variable into three shares. On one side this would increase the implementation cost even more. On the other side an attacker performing a third-order DPA can still reveal the key. However, such an attack would require much more traces decreasing the feasibility of the attack.

To generalize, given the above discussion, an adversary using $(d + 1)^{\text{st}}$-order DPA can successfully derive the secret information from a dth-order masking since he uses a nonlinear combination of information gathered from all $d + 1$ shares and reveals the secret. This randomized d^{th}-order masking hides the correlation between the sensitive variable and the power consumption for a dth-order adversary. We note that DPA attacks using mutual information [8], which exploit information from all possible orders together, can reveal the sensitive information from a masked implementation. Even though in theory such attacks are always successful given enough traces, in practice it becomes impractical to collect the required number of traces with increasing orders of the countermeasure. Therefore, they are out of the scope of this thesis.

If shared operations are performed at different times, combining information from those particular times nonlinearly also reveals the secret information. Combining information from t different times would produce a t-*variate* attack. The attack order in a t-variate attack still depends on the number of shares combined nonlinearly. An analysis where information from two shares are gathered from different times is referred to as bivariate second-order DPA in this thesis. This categorization allows us to further classify the DPA adversaries by their variant and order. Note that in practice it is hard to pinpoint the exact times when operations depending on each share are performed, which increases the complexity of such an attack. The DPA adversary described in this thesis is limited to univariate since shared operations are performed at the same time in all of the mentioned implementations.

Our ultimate goal is to provide a countermeasure that resists all known (and possibly unknown) attacks with minimum increase in resource requirements. However, achieving this goal is very hard due to the attack diversity. Typically,

a cryptographer takes the path to design a countermeasure against specific type of attacks with some pre-defined assumptions on the capabilities of the attacker or on the behavior of the device. Therefore this countermeasure might be insecure when the device behavior or the attack is out of the presumed model. The cryptographer usually takes a gradual approach to advance the countermeasure in order to provide security against a stronger attacker scenario or a wider range of devices.

In standard masking, both on the cell/instruction level [56, 96] and on the algorithmic level [32, 53, 71], the assumption is that there is no occurrence of unintended transition of a signal, the so called *glitch* which can reveal information from more than the expected amount of shares. The glitch-freeness, imposed by the masking model, limits the applicability of masking to different platforms. For instance, standard masking is insecure in the most common hardware circuitry CMOS (Complementary Metal Oxide Semiconductor) using standard CMOS cells, since glitches are unavoidable in CMOS circuits. Unfortunately, glitches can deteriorate the secret sharing by causing unwanted leakage depending on all shares, hence the shared sensitive variable. There exist only two masking schemes that are proven secure even in the presence of glitches so far, namely by Nikova et al. from ICICS'06 [75] and by Prouff et al. from CHES'11 [86]. The former, named *Threshold Implementation* (TI), can be implemented with a significantly smaller *gate count*[1] and requires much less randomness compared to the latter.

We choose TI which also splits the sensitive variable into several shares from this wide range of countermeasures mainly for three reasons. Firstly, unlike the ad-hoc approaches TI provides provable security hence is secure even with a large number of traces. Secondly, its provable security covers many platforms including the ones using CMOS-like cells that are problematic for some countermeasures. This is achieved by using $s \geq d + 1$ shares against a dth-order DPA such that no more than $s - 1$ of these shares are leaked to a dth-order adversary even in the presence of glitches. It can be applied using standard tools; furthermore, circuit-level investigation is unnecessary. And finally, the increase in resource requirements is low compared to other equivalent countermeasures. Even though TI is a very young countermeasure, before the beginning of this research, it has already been applied to standardized symmetric-key algorithms, namely PRESENT [83] and AES (Advanced Encryption Standard) [73] algorithms and a part of the NOEKEON [76] algorithm. The PRESENT and AES TIs showed that the timing overhead is negligible and area overhead is manageable.

---

[1]Even though the *gate count* of a circuit is not necessarily equal to its *area*, these words are used synonymously in cryptography, which we inherit throughout the thesis.

1.3   Research Questions

By the time this research has started, provable security of TI was shown against an adversary performing only first-order DPA. Existing TIs mentioned at the end of Section 1.2 use three shares such that no more than two of these shares are leaked to the adversary hence keeping the secret non-constructible. They have already been tested against such an adversary and shown to be secure as the theory suggests. As mentioned in Section 1.2, the ultimate goal is that the countermeasure resists a wide range of attacks and has minimum overhead. Our research questions are aligned with this statement. First of all, we would like to improve TI such that it provides provable security against a stronger adversary model than the suggested one. Following the step by step approach, we seek an answer to the following research question.

Question 1. *How can TI be improved to provide provable security against higher-order DPA which exploits higher-order statistical moments (variance, skewness, etc.)?*

Our goal, is not only to provide secure implementations against attack scenarios that are feasible with today's knowledge, but also to progress for future-proof implementations decreasing the reproduction cost.

Previous TIs show that if the building blocks of the symmetric-key algorithm is complex, a re-randomization of the shares might be necessary which requires random values. This re-randomization can be avoided if more shares are used which increases the area. Moreover, using more shares might increase the security of the system. This observation brings the following research question.

Question 2. *How does the decision of number of shares affect the area-randomness-security trade-off of TIs?*

Suggesting trade-offs between area, randomness and security adds flexibility to TI increasing the usability and the application range. We acknowledge that the area dimension can also include the randomness dimension if we consider the additional circuit required to generate the random numbers. However, a given device might already have a random number generator with a predefined throughput. In order to differentiate the area required by the countermeasure and the random number generator, we observe them in different dimensions.

Modifying the unprotected implementation to counteract DPA typically brings extra requirements especially on area. Therefore, a protected implementation is lower bounded by its unprotected version in terms of resource requirements. Combined with the previous motivational statement on minimizing the overhead this perception instigates the following question.

Question 3. *How close can the resource requirements of the TI get to the resource requirements of the unprotected implementation of the same algorithm?*

TI, enhanced by the answers to the above questions, is expected to become one of the main countermeasures to consider against DPA. Thus, an increasing number of secure embedded devices will more likely use this lightweight construction. That is why we also choose to exemplify our findings on standardized algorithms such as AES and SHA-3.

1.4   Thesis Overview

This dissertation studies theoretical and practical aspects of threshold implementations.   Chapter 2 starts with the notation presentation used throughout this thesis. Before introducing our contributions, we provide basic information on symmetric-key cryptography and its building blocks, especially the nonlinear *substitution boxes* (S-boxes) in the same chapter.   Moreover, we detail the preliminaries of DPA together with the most standard Boolean masking countermeasure. We examine the behavior of a circuit with glitches and explain why masking fails to provide security in such circuits.

We assemble most of the theoretical aspects of threshold implementations in Chapter 3. This theory was developed incrementally throughout this Ph.D. procedure and published in separate papers. The main contribution of this chapter is the answer to Question 1. A TI needs to satisfy four main properties to counteract higher-order DPA. These properties and the consequences of failing to satisfy these properties are discussed in this chapter.

In the following four chapters, we mainly analyze the practical aspects of threshold implementations. We especially focus on hardware implementations since TI differs from other masking schemes by its security in the presence of glitches. We always try to minimize the extra resource requirements of our threshold implementations with Question 3 in mind. In Chapter 4, we provide TI of KATAN cryptographic algorithm, which leans on a very simple (mathematically less complex) building block. We present resource requirements of TI together with experiments which confirm our theory. This work is published in [17].

Starting from Chapter 5, we analyze TI of more complex building blocks while considering an attacker performing a first-order DPA. Our findings on S-boxes up to size eight are are given in Chapter 5 and published in [22] and [23]. Moreover we used this knowledge during the design of FIDES [15] and PRIMATEs [4]

algorithms. We provide tools in [20] and [21] regarding this research for future references.

In Chapter 6 and 7, we implement several versions of KECCAK and AES algorithms such that we provide an answer to Question 2, namely area-randomness-security trade-offs of TI. These chapters contain several methods to reduce the area and randomness needs of a TI of a symmetric-key algorithm. The contents of these chapters are published in [16], [18] and [19]. [18] is a follow-up work of [73] improving it significantly. [19] is an extended version of [18] which analyses more trade-offs.

We described the proposed designs in Verilog and verified their functionality with ModelSim. Then we used a standard tool chain to synthesize them using Synopsys Design Vision D-201-.03-SP4. The NAND-gate equivalence (GE) of the circuit is taken as the area comparison metric. Unfortunately, we observed that there is no standard library used by all researchers which makes the comparison with previous works harder. Therefore we used several different libraries in order to provide a fair comparison with the prior works. The exact libraries used are provided in the beginning of the corresponding chapters. The use (GE)

We conclude this dissertation by listing open questions for future works in Chapter 8.

In the beginning of each chapter, we summarize in more detail which sections are published in which papers and what is our contribution.

2

Preliminaries

We start this chapter by introducing the general notations used in this thesis. The additional TI specific notations will be introduced in Section 3.1. We continue by providing preliminary information about symmetric-key cryptography in Section 2.2 which we use to exemplify our TI techniques. We mainly focus on Substitution Permutation Networks (SPNs). Hence, we detail their fundamental properties and building blocks. An S-box, which is usually a permutation defined in a finite field, is the only nonlinear building block of an SPN. In Section 2.2.1, we provide several properties of permutations since we primarily work on them. We describe a classification, which significantly reduces our work in the following chapters, based on affine equivalence of permutations. In addition, we categorize these S-boxes according to their sizes and examine size-specific properties in the rest of the section.

In order to confirm the security of our TIs of cryptographic algorithms, we play the role of an attacker. We use several DPA techniques which target different parts of the implementations in order to find a weakness. In the second half of this chapter, we describe these techniques. We focus on using a first-order DPA scenario during these descriptions in Sections 2.3.1, 2.3.5 and 2.3.6. We explain higher-order DPA using the probing model in Section 2.3.3. Additionally, we provide a discussion on why standard Boolean masking described in Section 2.3.2 becomes insecure on standard CMOS-circuits independent of their security order 2.3.4.

The pieces of information provided in this chapter are well known in the field.

These statements, which occur in the introductory sections of our papers [17, 18, 23] are partially written by co-authors.

2.1 Notation

We refer to a finite field $\mathcal{F}$ with characteristic c as $\mathcal{F}_c$. We mostly use fields with characteristic 2, namely $\mathcal{F}_{2^n}$ which are equivalent to $\mathbb{GF}(2^n)$. If it is clear from the context, we denote $\mathcal{F}_2$ by $\mathcal{F}$ for convenience. One *bit* refers to an element in $\mathcal{F}_2$. A *nibble* and a *byte* are 4- and 8-bit elements respectively. The size of a field is $|\mathcal{F}|$.

Lower-case characters refer to elements of a finite field $\mathcal{F}$, while upper-case characters are used for stochastic variables. The probability that X takes the value x is $Pr(X = x)$.

The number of ones in the binary description of the value x is called as its *Hamming weight* (HW). The HW of the difference between x and y is referred to as their *Hamming distance* (HD). We denote these notions with $HW(x)$ and $HD(x, y)$ respectively.

The bitwise addition and multiplication, which are referred to as the XOR and the AND operations are denoted by $\oplus$ and $\odot$ respectively. The operations $+$ and $\times$ stand for the addition and the multiplication in a given field. For convenience, the multiplication of two values $x \odot y$ is sometimes described as xy.

A function f is defined from $\mathcal{F}^n$ to $\mathcal{F}^m$ where n and m are natural numbers. If f is a bijection and $m = n$, the function is a *permutation*, hence is invertible. Any function $f(X)$ can be considered as an m-tuple of Boolean functions $(f^1(X), \ldots, f^m(X))$, where $X \in \mathcal{F}^n$, which are called the *coordinate functions* of $f(X)$. In Equation (2.1), we provide a 3-bit permutation f defined from $\mathcal{F}^3$ to $\mathcal{F}^3$ with the input $X = (W, Y, Z)$ where $w, y, z \in \mathcal{F}$, the output $(A, B, C) \in \mathcal{F}^3$ and the coordinate functions f^1, f^2 and f^3.

$$A = f^1(W, Y, Z) = W$$
$$B = f^2(W, Y, Z) = 1 \oplus Y \qquad (2.1)$$
$$C = f^3(W, Y, Z) = WY \oplus Z$$

The degree of a function is the maximum algebraic degree of these coordinate functions [26]. If the degree is one with zero or non-zero constants, the function

is called a *linear* (f^1) or an *affine* (f^2) function respectively. Otherwise, it is called a *nonlinear* (f^3) function. Equation (2.1) is quadratic since the maximum degree is two (f^3).

Finally, we express a vector of elements, variables or functions with bold characters. The dot product of x and y on $\mathcal{F}^n$ is denoted by $\langle x, y \rangle$.

2.2 Symmetric-key Cryptography

There are several types of symmetric-key algorithms, some of which are block ciphers, hash functions and authenticated encryption algorithms. A *block cipher* takes a secret key value $key \in \mathcal{K}$ and a block of *plaintext* $pt \in \mathcal{P}$, where $\mathcal{K}$ and $\mathcal{P}$ define the key and the plaintext space. These spaces are equal to $\mathcal{F}^k$ and $\mathcal{F}^p$ where k is the size of key and p is the size of pt respectively. The block cipher produces an output *ciphertext* $ct \in \mathcal{P}$ by using an *encryption* operation $E(key, pt)$. This operation, which provides confidentiality, is a nonlinear permutation when the key is fixed implying invertibility. The inverse *decryption* operation reproduces the plaintext by using the same key and the corresponding ciphertext. A *hash function*, on the other hand, is a keyless operation which inputs a plaintext block $H(pt)$ and outputs a hash value $hash \in \mathcal{H}$ where $|\mathcal{H}| < |\mathcal{P}|$ hence, it is not invertible. The hash value resembles a digital fingerprint which can be used to provide message integrity with the help of public key cryptography. If the hash function is modified such that it also takes a key as input, then it provides both authentication and integrity. An *authenticated encryption* algorithm is a combination of all these in the sense that it provides confidentiality, integrity and authentication. It inputs a key and blocks of plaintext, outputs blocks of ciphertext together with a *tag*. The tag resembles to a hash output.

These algorithms must provide a good confusion and diffusion of the key and the plaintext to provide the required mathematical security and resist cryptanalysis [92]. *Confusion*, which makes the relationship between the key and the ciphertext as complex as possible, is achieved with nonlinear operations. Linear operations assure that one bit change in the state spreads over the whole state quickly hence, provide *diffusion*.

A popular way to generate a symmetric-key algorithm is to use the output of a round, which is composed of linear and nonlinear operations, as the input to the next round, consisting of the same operations, in a cascaded manner. These rounds are typically formed as a layer of round-key XOR with the round input, a layer of nonlinear substitution blocks (*S-boxes*) and a layer of linear permutations. This round structure is called a *Substitution Permutation Network*

(SPN) where the substitution and the permutation layers provide confusion and diffusion respectively. The block cipher standards DES (Data Encryption Standard) [43] and AES [80], the hash function standard SHA-3 which is a subset of the KECCAK family [13] and the lightweight block cipher standard PRESENT [25] are examples of SPN.

2.2.1 Permutations and Affine Equivalence Relations

In Chapter 3, we will show that TI of any affine function, hence the key XOR and the permutation layer of an SPN, are trivial. On the other hand, TI of a nonlinear operation such as the S-box of the substitution layer can be challenging, determining our main focus. Most of these S-boxes are permutations defined over a small field (e.g. $\mathcal{F}_{2^4}$ or $\mathcal{F}_{2^8}$). Only a few exceptional cryptographic algorithms use S-boxes from $\mathcal{F}_{2^n}$ to $\mathcal{F}_{2^m}$, i.e. with n input and m output bits referred to as an $n \times m$ S-box. In this section, we investigate the properties of permutations.

All permutations from a set $\mathcal{D}$ to itself form the *symmetric group* on $\mathcal{D}$ denoted by $\mathcal{S}_{\mathcal{D}}$. A transposition is a permutation which exchanges two elements and keeps all others fixed. A classical theorem states that every permutation can be represented as a product of transpositions [89], and although this representation is not unique, the number of transpositions needed is either always even or always odd. The set of all even permutations form a normal subgroup of $\mathcal{S}_{\mathcal{D}}$, which is called the *alternating group* on $\mathcal{D}$ and denoted by $\mathcal{A}_{\mathcal{D}}$. The alternating group contains half of the elements of $\mathcal{S}_{\mathcal{D}}$. Instead of $\mathcal{A}_{\mathcal{D}}$ and $\mathcal{S}_{\mathcal{D}}$, we will write here $\mathcal{A}_m$ and $\mathcal{S}_m$, where m is the size of the set $\mathcal{D}$.

Lemma 1 ([99])**.** *For all $n \geq 3$, the n-bit affine permutations are in the alternating group.*

We classify permutations according to their affine equivalence as defined below to reduce the working space.

Definition 1 ([40])**.** *Two permutations $f(X)$ and $\tilde{f}(X)$ are* affine/linear equivalent *if there exists a pair of affine/linear permutations $l_r(X)$ and $l_l(X)$, such that $\tilde{f} = l_l \circ f \circ l_r$.*

Every affine permutation $l(X)$ can be written as $\mathrm{L} \cdot X + c$ with c an n-bit constant and L an $n \times n$ matrix which is invertible over $\mathcal{F}_2$. It follows that there are

$$2^n \times \prod_{i=0}^{n-1}(2^n - 2^i) \tag{2.2}$$

different affine permutations.

The relation "being affine equivalent" can be used to define equivalence classes. In Section 2.2.2, we provide lists of affine equivalence classes for 3- and 4-bit permutations. The classes are enumerated by the lexicographical order of their representatives' truth tables.

Note that the algebraic degree is invariant under affine equivalence, hence all permutations in a class have the same algebraic degree. Moreover, the maximal algebraic degree of an n-bit permutation is $n - 1$ [30, 67].

In order to increase readability, we introduce the following notation $\mathcal{A}_i^n$, $\mathcal{Q}_j^n$, $\mathcal{C}_k^n$ to denote the Affine class number i, Quadratic class number j and Cubic class number k of permutations of $\mathcal{F}_2^n$. Moreover, if a permutation is represented with an even (resp. odd) number of transpositions, all of its affine equivalent permutations are also represented with an even (resp. odd) number of transpositions.

2.2.2 2-, 3- and 4-bit Permutations

It is well known that all 2-bit permutations are affine, hence there is only one class. The set of 3-bit permutations contains 4 equivalence classes [40]: 3 classes containing quadratic functions, and 1 class containing the affine functions. The Inversion in $\mathcal{F}_{2^3}$ and the S-boxes of the PRINTcipher [61], the Threeway [38] and the Baseking [39] algorithms, which are the only cryptographically significant 3×3 S-boxes, belong to the quadratic class Q_3^3. The notations for all the 3-bit permutation classes together with their representatives are provided in the first two columns of Table A.1 in Appendix A.

De Cannière [24, 40] uses an algorithm to search for the affine equivalent classes which guesses the effect of the affine permutation l_r for as few input points as possible, and then uses the linearity of l_r and l_l (as given in Definition 1) to follow the implications of these guesses as far as possible. This search is accelerated by applying the next observation, which follows from linear algebra arguments (change of basis):

Lemma 2 ([66]). *Let f be an n-bit permutation. Then f is affine equivalent to another permutation $\tilde{f}$ with $\tilde{f}(X) = X$, for $X \in \{0, 1, 2, 4, 8, \ldots, 2^{n-1}\}$.*

In the case $n = 4$, this observation reduces the search space from $16! \approx 2^{44}$ to $11! \approx 2^{25}$.

De Cannière lists the 302 equivalence classes for the 4-bit permutations [40]: the class of affine functions, 6 classes containing quadratic functions and the

remaining 295 classes containing cubic functions. The classes are listed in the first two columns of Tables A.2–A.6 in Appendix A.

There are many cryptographically significant 4-bit permutations. First Leander and Poschmann [66] and later Saarinen et al. [90] classify all 4×4 invertible S-boxes up to affine equivalence and provide 16 "golden" S-box classes that provide optimal differential and linear properties which is helpful to design a secure and efficient algorithm. Tables A.10–A.12 in Appendix A list some of the S-boxes used in the design of cryptographic algorithms together with golden S-boxes (depicted as Optimal G_i) and the classes to which they belong.

Note that f^{-1}, the inverse permutation, is not necessarily affine equivalent to f and in this case may not have the same algebraic degree. We know however, that the inverse of an affine permutation is always an affine permutation. In the case of 3-bit permutations it follows that the inverse of a quadratic permutation is again a quadratic permutation. Moreover, it can be shown that the 3 quadratic classes in $\mathcal{S}_8$ are self-inverse, i.e. f^{-1} belongs to the same class as f. In the case $n = 4$, we can apply the following lemma.

Lemma 3 ([26]). *Let f be a permutation of $\mathbb{GF}(2^n)$, then $deg(f^{-1}) = n - 1$ if and only if $deg(f) = n - 1$.*

Since the inverse of an affine permutation is affine, and, when $n = 4$, the inverse of a cubic permutation is cubic, it follows that in this case the inverse of a quadratic permutation is quadratic. The KECCAK S-box ($n = 5$) [13], which is a permutation, is an example where the algebraic degree of the inverse S-box ($deg(f^{-1}) = 3$) is different from the algebraic degree of the S-box itself ($deg(f) = 2$).

We have observed that there are 172 self-inverse classes in the symmetric group $\mathcal{S}_{16}$. The remaining 130 classes form 65 pairs, i.e., any permutation f of the first class has an inverse permutation f^{-1} in the second class (and vice versa). Table 2.1 gives the list of the pairs of inverse classes.

2.2.3 5- and 6-bit Permutations

The number of classes increase exponentially when bigger permutations are considered. There exist roughly 2^{61} and 2^{215} different affine equivalent classes for 5-bit and 6-bit permutations respectively [40]. They have been used in cryptographic primitives. An important example is the 5-bit quadratic function of KECCAK [13] as mentioned in Section 2.2.2. 5-bit almost bent permutations and 6-bit almost perfect nonlinear permutations are also well studied since they have a particular importance in cryptography.

Table 2.1: 65 pairs of inverse classes that are not self-inverse; the remaining 172 classes are self-inverse

$(\mathcal{C}^4_{29},\mathcal{C}^4_{30}),(\mathcal{C}^4_{33},\mathcal{C}^4_{34}),(\mathcal{C}^4_{39},\mathcal{C}^4_{40}),(\mathcal{C}^4_{43},\mathcal{C}^4_{44}),(\mathcal{C}^4_{47},\mathcal{C}^4_{48}),(\mathcal{C}^4_{49},\mathcal{C}^4_{50}),(\mathcal{C}^4_{52},\mathcal{C}^4_{53}),(\mathcal{C}^4_{58},\mathcal{C}^4_{59}),$
$(\mathcal{C}^4_{60},\mathcal{C}^4_{61}),(\mathcal{C}^4_{63},\mathcal{C}^4_{64}),(\mathcal{C}^4_{66},\mathcal{C}^4_{67}),(\mathcal{C}^4_{68},\mathcal{C}^4_{69}),(\mathcal{C}^4_{70},\mathcal{C}^4_{71}),(\mathcal{C}^4_{73},\mathcal{C}^4_{74}),(\mathcal{C}^4_{79},\mathcal{C}^4_{80}),(\mathcal{C}^4_{85},\mathcal{C}^4_{86}),$
$(\mathcal{C}^4_{87},\mathcal{C}^4_{88}),(\mathcal{C}^4_{90},\mathcal{C}^4_{91}),(\mathcal{C}^4_{93},\mathcal{C}^4_{94}),(\mathcal{C}^4_{95},\mathcal{C}^4_{96}),(\mathcal{C}^4_{97},\mathcal{C}^4_{98}),(\mathcal{C}^4_{103},\mathcal{C}^4_{104}),(\mathcal{C}^4_{105},\mathcal{C}^4_{106}),$
$(\mathcal{C}^4_{108},\mathcal{C}^4_{109}),(\mathcal{C}^4_{110},\mathcal{C}^4_{111}),(\mathcal{C}^4_{112},\mathcal{C}^4_{113}),(\mathcal{C}^4_{114},\mathcal{C}^4_{115}),(\mathcal{C}^4_{116},\mathcal{C}^4_{117}),\ (\mathcal{C}^4_{120},\mathcal{C}^4_{121}),$
$(\mathcal{C}^4_{123},\mathcal{C}^4_{124}),(\mathcal{C}^4_{126},\mathcal{C}^4_{127}),(\mathcal{C}^4_{128},\mathcal{C}^4_{129}),(\mathcal{C}^4_{130},\mathcal{C}^4_{131}),(\mathcal{C}^4_{132},\mathcal{C}^4_{133}),(\mathcal{C}^4_{143},\mathcal{C}^4_{144}),$
$(\mathcal{C}^4_{147},\mathcal{C}^4_{148}),(\mathcal{C}^4_{150},\mathcal{C}^4_{151}),(\mathcal{C}^4_{152},\mathcal{C}^4_{153}),(\mathcal{C}^4_{154},\mathcal{C}^4_{155}),(\mathcal{C}^4_{156},\mathcal{C}^4_{157}),(\mathcal{C}^4_{158},\mathcal{C}^4_{159}),$
$(\mathcal{C}^4_{161},\mathcal{C}^4_{162}),(\mathcal{C}^4_{164},\mathcal{C}^4_{165}),(\mathcal{C}^4_{166},\mathcal{C}^4_{167}),(\mathcal{C}^4_{169},\mathcal{C}^4_{170}),(\mathcal{C}^4_{171},\mathcal{C}^4_{172}),(\mathcal{C}^4_{181},\mathcal{C}^4_{182}),$
$(\mathcal{C}^4_{183},\mathcal{C}^4_{184}),(\mathcal{C}^4_{185},\mathcal{C}^4_{186}),(\mathcal{C}^4_{190},\mathcal{C}^4_{191}),(\mathcal{C}^4_{199},\mathcal{C}^4_{200}),(\mathcal{C}^4_{201},\mathcal{C}^4_{202}),(\mathcal{C}^4_{203},\mathcal{C}^4_{204}),$
$(\mathcal{C}^4_{206},\mathcal{C}^4_{207}),(\mathcal{C}^4_{209},\mathcal{C}^4_{210}),(\mathcal{C}^4_{211},\mathcal{C}^4_{212}),(\mathcal{C}^4_{214},\mathcal{C}^4_{215}),(\mathcal{C}^4_{226},\mathcal{C}^4_{227}),(\mathcal{C}^4_{229},\mathcal{C}^4_{230}),$
$(\mathcal{C}^4_{233},\mathcal{C}^4_{234}),(\mathcal{C}^4_{241},\mathcal{C}^4_{242}),(\mathcal{C}^4_{243},\mathcal{C}^4_{244}),(\mathcal{C}^4_{256},\mathcal{C}^4_{257}),(\mathcal{C}^4_{259},\mathcal{C}^4_{260}),(\mathcal{C}^4_{296},\mathcal{C}^4_{297}).$

Definition 2 ([31]). *The permutation f is said to be* almost perfect nonlinear (APN) *if all the equations*

$$f(X) \oplus S(X \oplus A) = B,\ A, B \in \mathbb{GF}(2^n), A \neq 0,$$

have either 0 or 2 solutions.

Definition 3 ([31]). *The permutation f is said to be* almost bent (AB) *if the Walsh transform*

$$\mu_{f(A,B)} = \sum_{X \in \mathbb{GF}(2^n)} (-1)^{\langle B, f(X) \rangle \oplus \langle A, X \rangle},$$

is equal to either 0 or $\pm 2^{\frac{n+1}{2}}$ when $A, B \in \mathbb{GF}(2^n)$ and $(A, B) \neq (0, 0)$.

It is known that all AB permutations are also APN. An APN permutation provides optimum resistance only against differential cryptanalysis whereas an AB permutation provides optimum resistance against both differential and linear cryptanalysis [31]. Unfortunately, AB permutations exist only when n is odd [31].

Up to affine equivalence there are only four AB permutations of dimension five, all of which can be represented as a power function [28]. A representative of each class is provided in Table 2.2. We note that $AB4$ and $AB3$ are the inverse of $AB1$ and $AB2$, respectively.

Table 2.2: Representatives of AB permutations in $\mathbb{GF}(2^5)$ [28]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$AB1$	0	1	2	4	3	8	16	28	5	10	25	17	18	23	31	29	6
$AB2$	0	1	2	4	3	8	16	28	5	10	26	18	17	20	31	29	6
$AB3$	0	1	2	4	3	8	13	16	5	17	28	27	30	14	24	10	6
$AB4$	0	1	2	4	3	8	13	16	5	11	21	31	23	15	19	30	6
	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	deg.	pow.
$AB1$	20	13	24	19	11	9	22	27	7	14	21	26	12	30	15	2	x^3
$AB2$	21	24	12	22	15	25	7	14	19	13	23	9	30	27	11	2	x^5
$AB3$	19	11	20	31	29	12	21	18	26	15	25	7	22	23	9	3	x^7
$AB4$	28	29	9	24	27	14	18	10	17	12	26	7	25	20	22	3	x^{11}

There is only one known affine equivalence class of 6-bit APN permutations [44]. A representative of this APN permutation, which has degree 4, is provided in Table 5.5 in Section 5.2 for convenience.

2.2.4 8-bit Permutations

The full list of all affine equivalent classes of 8-bit permutations is not generated yet since no efficient algorithm to produce such a list is known. However, Rijndael [37] and its standardized version AES [80], both of which use a substitution layer composed of 8-bit permutations with strong cryptographic properties to resist cryptanalysis, inspired many symmetric-key algorithms. Similar to AES, most of these algorithms use an S-box based on a multiplicative inversion in $\mathcal{F}_{2^8}$ followed by an affine transformation. Namely, the S-box can be represented as $f(X) = L \cdot X^{-1} \oplus c$ where the specific values for the matrix L and the constant c change from design to design.

There exist other systematic ways to generate 8-bit permutations, e.g. combining several smaller permutations [54], or using genetic algorithms [33]. The former type of S-boxes can be examined by the properties of the permutations they are composed of. The latter method is not yet widely adopted by cryptographic algorithms, therefore they are out of the scope of this thesis. Here, we will mainly focus on the AES S-box of which the details can be found in [80].

A polynomial-based implementation and table look-up of the AES S-box are not preferred for lightweight applications since they are big in area. Moreover, the algebraic degree of the S-box is seven and produces very complex coordinate
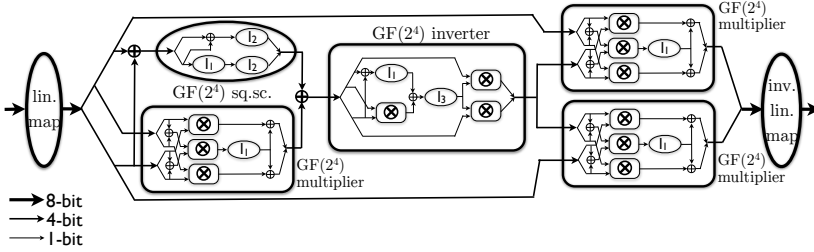
Figure 2.1: Schematic of AES S-box using tower field approach

functions that are not preferable on hardware. Instead the *tower field approach* is used [29] to achieve optimal area on hardware. With this approach, the inversion in $\mathcal{F}_{2^8}$ is implemented as inversion, multiplication and some linear operations in $\mathcal{F}_{2^4}$. Similarly, the multiplication and inversion in $\mathcal{F}_{2^4}$ can be implemented by using building blocks from $\mathcal{F}_{2^2}$. The diagram for a small unprotected tower field description reported by Canright is described in Figure 2.1. l_1, l_2 and l_3 correspond to the linear operations square scaling, squaring and inversion in $\mathcal{F}_{2^2}$.

2.3   Differential Power Analysis and Masking

Differential power analysis (DPA) uses multiple power traces collected from iterating an (encryption) algorithm with different plaintexts and the same key. It is assumed that the instantaneous power consumption is a linear combination of the outputs of *noisy leakage functions* $\mathcal{L}(.)$, each of which takes a subset of intermediate operations/variables happening at the same time as inputs and produces linear translations of them with additional Gaussian independent noise. Hence, the leakage from each encryption differs depending on the intermediate values generated during an encryption.

To clarify, consider a 1-bit intermediate variable X and assume that $\mathcal{L}(0) \neq \mathcal{L}(1)$; e.g. the device under test leaks the HW of X ($\mathcal{L}(X) = \mathrm{HW}(X)$). Given enough traces, this difference reveals the value x. If the intermediate variable depends on the sensitive variable, it helps the attacker to recover the sensitive value.

Early works on DPA, such as DoM described in Section 1.1, consider the leakage from the input or the output of a combinational operation. In this thesis, we use a more sophisticated version of this attack called correlation power analysis (CPA) [27] which is described in Section 2.3.1.

2.3.1 Correlation Power Analysis

CPA exploits the correlation between power traces and key dependent intermediate values. There are two aspects of a successful CPA. The first one is to identify the leakage behavior of the device under test as accurately as possible. The HW leakage of an intermediate value is typically considered to be a good estimation for DPA. For operations such as register and memory transitions, this is translated into the HD between two intermediate states that use the same resources one after another. We note that, statistically, the HD leakage is equivalent to the HW leakage if and only if one of the two intermediate states is fixed to a constant value or completely random. Otherwise, it might cause a decrease in the hypothesized security [5]. During CPA, the target resource is typically taken as a memory element in which intermediate values of a round, such as the key XOR, substitution layer or one round output lie. Even though it is also possible to consider the combinational operations, such a CPA might provide inadequate results, since projecting combinational operations to a leakage model is difficult even with a deep knowledge on the device and the particular implementation. If the intermediate states used for HD calculation depend on different key chunks, the number of bits of the key that needs to be guessed during the attack, and hence the complexity of the attack increases.

The second aspect of a successful CPA is choosing the target state such that the information becomes easier to exploit. In theory, any key dependent intermediate state can be the target since power consumption depends on every single operation in the device. On the other hand, the information from the output of a cryptographically strong S-box is more profitable than the output of an affine operation, such as the key addition itself since a wrong key guess is more distinguishable from the correct one in the former case [84].

During the attack phase, the attacker generates a $p \times t$ matrix T which corresponds to p power traces of length t representing encryptions with different plaintexts under the same key. The attacker also computes a $p \times k$ matrix L where each column corresponds to hypothetical leakages of the targeted state during the encryption of all plaintexts with a different key hypothesis. The Pearson correlation of the sample sets (the jth column of) T and (the lth column of) L using the following formula in which $\bar{T}$ and $\bar{L}$ correspond to the mean values, reveals the correct key hypothesis.

$$r_{j,l} = \frac{\sum_{i=1}^{p}(T_{i,j} - \bar{T}_j)(L_{i,l} - \bar{L}_l)}{\sqrt{\sum_{i=1}^{p}(T_{i,j} - \bar{T}_j)^2}\sqrt{\sum_{i=1}^{p}(L_{i,l} - \bar{L}_l)^2}} \tag{2.3}$$

More specifically, the attacker constructs a matrix R from the $r_{j,l}$ values where

each column corresponds to the correlation coefficients of a key hypothesis on all the time samples. The index(es) corresponding to the absolute maximum of the elements of R reveals the specific time sample(s) where the hypothetical leakage takes place with a particular key hypothesis indicating a correct guess.

CPA-like attacks that use the first-order moment of information triggered the development of different countermeasures (Section 1.2) some of which can be gathered under the set of masking as described below.

2.3.2 Masking

A conventional first-order Boolean masking scheme splits the intermediate variable X in two randomized variables X_1 and X_2 such that $X_1 \oplus X_2 = X$. In Table 2.3, we demonstrate the leakage $\mathcal{L}(X) = \mathrm{HW}(X_1, X_2)$ depending on these variables. It can be seen that this masking does not reveal any information on the value of x when a first-order analysis is performed since the mean of the leakages given at the fourth column is constant for different x. However, a second-order analysis can reveal the difference of variances for $x = 0$ and $x = 1$ provided in the fifth column given enough traces.

Table 2.3: Leakage behaviour of 1-bit split into two shares

x	x_1	x_2	$\mathcal{L}(x)$	$\mathrm{Mean}(\mathcal{L}(x))$	$\mathrm{Var}(\mathcal{L}(x))$
0	0	0	0	1	1
	1	1	2		
1	0	1	1	1	0
	1	0	1		

Higher-order Masking

To generalize, a dth-order masking countermeasure aims randomizing an intermediate sensitive variable X by splitting it into $d+1$ uniformly distributed variables $X_1, \ldots, X_d, X_{d+1}$, such that the following equation holds under the group operation $\perp$.

$$X = X_1 \perp X_2 \perp \ldots \perp X_d \perp X_{d+1} \tag{2.4}$$

Aforementioned $\perp$ operation can be multiplication or addition generating a multiplicative and additive masking respectively. The special form of additive masking where the group operation is bitwise XOR is called Boolean masking. Hereon, the latter scheme is the main consideration and the terms Boolean masking and masking are used interchangeably. Each variable X_i is referred to as a secret share and the secret sharing is typically done by generating (without loss of generality) d uniformly distributed random shares $X_1, \ldots, X_d$ and calculating X_{d+1} such that it satisfies Equation (2.4).

Given an input sharing, all the operations within an algorithm, such as the linear transformations and the S-box calculations, must also be carried out using these shares. Performing an affine transformation l on shared variables is straightforward since Equation 2.5 holds implying that we can apply the affine transformation to each share individually using $d + 1$ iterations/instantiations of l.

$$l(X) = l(X_1 \oplus \cdots \oplus X_{d+1}) = l(X_1) \oplus \cdots \oplus l(X_{d+1}) \qquad (2.5)$$

Nonlinear operations for which the above equation is incorrect, are more challenging. Equation 2.6 provides one sharing of $f(X, Y) = Z \oplus XY$ with two shares (for each input and output) that is secure against first-order DPA attacks if the circuit is glitch-free.

$$f_1(X_1, Y_1) = Z_1 \oplus X_1 Y_1$$

$$f_2(X_1, X_2, Y_1, Y_2) = ((Z_2 \oplus X_1 Y_2) \oplus X_2 Y_1) \oplus X_2 Y_2 \qquad (2.6)$$

The order of operations specified with the parentheses is important not to unmask a sensitive variable which is hard to achieve even with time consuming cell level investigation of a circuit in hardware. As a trivial example of breaking the order of operations, assume that $(X_2 Y_1 \oplus X_2 Y_2)$ is calculated. Equality of this XOR to $X_2 Y$ reveals the unmasked information Y.

Subsequently, we first analyze the security of a higher-order masking scheme under the ideal glitch-free leakage assumption. We then discuss how these idealized assumptions fail when standard CMOS gates are used in the circuit.

2.3.3 Higher-order DPA

As mentioned in Section 2.3.1, the leakage function $\mathcal{L}(.)$ is ideally assumed to depend on the input or the output of a shared function since it is difficult to

provide a more accurate leakage function without considering each intermediate transition in the circuit which would require a deep cell level investigation. In this section, we assume that the attacker is limited to such an idealized leakage function for analysis.

A DPA using a nonlinear combination of d intermediate variables is called a dth-order DPA. A CPA-like analysis exploiting the information difference in a dth-order moment can be used to observe a nonlinear combination of d shares of an intermediate variable if all its shares are updated simultaneously. We will apply this adaptation by centering and taking the dth power of the traces for each time sample before performing CPA described in Section 2.3.1.

Note that any masking scheme using $d + 1$ shares to represent a variable is vulnerable against $(d + 1)^{\text{st}}$-order DPA since an information depending on all the shares nonlinearly is leaked. An example is given in the first-order masking and second-order analysis case at the beginning of Section 2.3.2.

In a related passive-invasive adversary model, the attacker can observe the values of up to d intermediate wires of the circuit per bit during the computation within a certain time. The correspondence between this model which is called the *d-probing model* and the dth-order DPA attack model with the noisy leakage function is shown [36, 49, 88]. Moreover, this *d*-probing model is used [45] to prove security against d^{th}-order DPA. We make use of the following result.

Lemma 4. *The attack order in a higher-order DPA corresponds to the number of wires that are probed in the circuit (per unmasked bit).*

This lemma implies that if a circuit is perfectly secure against d probes, then combining d power consumption points nonlinearly as in a dth-order DPA will reveal no information. If the operations that correspond to the probed wires are in parallel, this is equivalent to security against DPA exploiting the dth-order statistical moment.

We note that the d-probing model is a stronger notion compared to higher-order DPA. Namely, if a system is secure against d-probing adversary, it is also secure against a dth-order DPA if the noisy leakage function $\mathcal{L}(.)$ behaves as described in the first paragraph of Section 2.3 [56]. But the inverse argument is not necessarily correct. Hereon, we discuss the security of the masking scheme against a d-probing adversary using Lemma 4.

2.3.4 Security on a Glitchy Circuit

In an ideal circuit implemented using standard CMOS logic, each gate exhibits at most one transition per clock cycle. However, physical repercussions, such as

propagation delays can cause a node to transition more than once if the node allows. An unintended double transition is called a *glitch* and is very common in unbalanced CMOS circuits [58]. Below we discuss the negative effect of glitches on securing the circuit performing the shared operation in Equation (2.6) of which the second-share circuit is described in Figure 2.2.
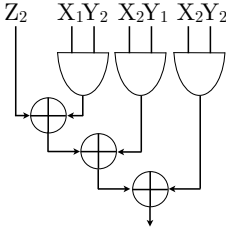
Figure 2.2: One share of the Boolean masked AND/XOR gate

y	y_1	y_2	x_2	$z_2 \oplus x_1 y_2$	AND	XOR
0	0	0	0	0	0+0	0+0
0	1	1	0	0	0+0	0+0
0	0	0	1	0	0+0	0+0
0	1	1	1	0	0+2	0+1
0	0	0	0	1	0+0	2+0
0	1	1	0	1	0+0	2+0
0	0	0	1	1	0+0	2+0
0	1	1	1	1	0+2	2+1
1	0	1	0	0	0+0	0+0
1	1	0	0	0	0+0	0+0
1	0	1	1	0	0+1	0+1
1	1	0	1	0	0+1	0+2
1	0	1	0	1	0+0	2+0
1	1	0	0	1	0+0	2+0
1	0	1	1	1	0+1	2+1
1	1	0	1	1	0+1	2+2

Table 2.4: The number of AND/XOR transitions on a glitchy circuit caused by the delay of X_2

Assume that the input values of the given circuit change from the constant zero initial state to a random value and that the input X_2 is slightly delayed due to a prior propagation delay. The amount of AND and XOR gate transitions resulting from this operation is given in the last two columns of Table 2.4 respectively. The values Z_2 and X_1 are ignored during this investigation since they do not have a joint effect on transitions when X_2 arrives late and the value $Z_2 \oplus X_1 Y_2$ is uniformly distributed. The plus sign is used to separate the transitions before and after the late arrival of the value x_2. Note that these transitions reflect the power consumption of the device directly[1]. Moreover, some of these transitions combined cause glitches, e.g. the case where $y_1 = x_2 = z_2 \oplus x_1 y_2 = 1$ and $y_2 = 0$.

Observe that the average XOR gate transitions vary for different unmasked values of Y. Therefore, the average power consumption when $y = 0$ and $y = 1$ are different revealing unintended secret information on Y.

Note that the counting in Table 2.4 refers to only one possible transition

---

[1]Even though the power consumption of a transition from 1 to 0 and from 0 to 1 is usually different in reality, we treat them as equal to simplify the example. A more sophisticated analysis where these consumptions are threated differently gives a similar result.

assuming that two inputs of the last (rightmost) XOR gate arrive at the same time. It is also possible that they arrive in different times causing that gate to glitch when $(Y_1, Y_2) = (1, 1)$. In that scenario, the late arrival of X_2 is considered safe since it does not reveal any information on the unmasked values. However, such an examination on the circuit level for every possible gate is time consuming and expensive. Note also that a misbehavior in one share of an element does not reveal information on that element but reveals information on another element that is nonlinearly combined with it.

Hereon, we modify the the probing model introduced in Section 2.3.3 [56] slightly in order to include the effect of glitches as follows. By probing a wire calculating a function, the attacker gets information on each input of the function (Y_1, Y_2, etc.), each intermediate value of the function (X_2Y_1, X_2Y_2, etc.) and the output of the function. This relation also shows that the probing adversary is stronger than that of DPA.

Predicting the exact timing of a device, and hence its leakage under these circumstances is difficult which makes the usage of CPA to recover the key from the combinational behavior highly challenging. On the other hand, the following analysis technique can be used to reveal information on the sub-key without any estimation on the leakage of the device.

2.3.5   Correlation-Enhanced Power Analysis Collision Attack

Correlation-enhanced power analysis collision attack (CEPACA) [72] inherits "divide and conquer" attack strategy similar to CPA. It is especially preferable if the exact leakage model is unknown to the attacker or hard to estimate (e.g. leakage from the combinational logic). Below we describe the attack for the first-order DPA model. It can be modified to perform higher-order analysis by using d^{th}-order moments of traces, such as the variances for the second-order case, instead of the mean traces described below.

The attack focuses on exposing the occurrences of collisions in two different intermediate values, typically in the S-box inputs or outputs using the same circuit. Specifically, consider two S-box inputs $pt_i \oplus key_i$ and $pt_j \oplus key_j$, where $i \neq j$. If these S-box inputs are equal, the S-box circuit behaves similarly during the calculations depending on these values. The equality of these intermediate values implies that $pt_i \oplus pt_j = key_i \oplus key_j$. Hence, the input difference $\Delta_{i,j}$ between the plaintext chunks is equal to the difference between the sub-keys. The attack is performed on several i, j pairs revealing different collisions hence relations between different sub-keys. Note that CEPACA significantly reduces the key space without detecting the exact sub-key values.

The attacker works on average power traces M_i^α and M_j^α generated by grouping the collected power traces for different α values of pt_i and pt_j respectively then taking their mean. The correlations between the mean traces corresponding to M_i^α and $M_j^{\alpha \oplus \Delta_{i,j}}$ are calculated for all possible $\Delta_{i,j}$ values. The correlation converges to zero for a wrong $\Delta_{i,j}$ value whereas the right $\Delta_{i,j}$ reveals itself as a peak when these correlation traces are plotted.

Even though CPA and CEPACA like attacks are valuable in order to reveal the sub-key, resistance against such attacks does not imply security against all possible attacks. The failure of an attack can result from wrong leakage function assumptions, poor intermediate value choices or the lack of the strength of the attack. However, by using the t-test based leakage detection which is described in the next section, we can determine the existence of any d^{th}-order leakage. The existence of such a leakage does not necessarily lead to key recovery since it might be caused by an intermediate value that is independent of the key. On the other hand, the absence of leakage given enough traces strongly supports the claimed security. Similar to the CPA and CEPACA, we describe the technique only for first-order DPA.

2.3.6   T-test Based Leakage Detection

Contrary to previous DPA techniques t-test based leakage detection [35] analyses differences of leakages without focusing on key recovery. The attacker collects traces from chosen inputs to generate two sets of measurements for which intermediate values in the implementation have a certain difference. A safe choice is to keep the intermediate values fixed for one set of measurements, while they take random values for the second set without making an assumption about how the implementation leaks. The test is *specific*, if particular intermediate values or transitions in the implementation are targeted (e.g. S-box input, S-box output, HD in a round register, etc.). This type of testing requires knowledge of the device key and carefully chosen inputs. On the other hand, the test is *non-specific* if *all* intermediate values and transitions are targeted at the same time. This type of testing only requires to keep all inputs to the implementation fixed for one set of measurements, and to choose them randomly for the second set. Obviously, the non-specific test is extremely powerful.

The attacker computes t-test statistics to determine if the two sets of measurements $\mathcal{S}_0$ and $\mathcal{S}_1$ are significantly different. We compute Welch's (two-tailed) t-test

$$t = \frac{\mu(\mathcal{S}_0) - \mu(\mathcal{S}_1)}{\sqrt{\frac{\sigma^2(\mathcal{S}_0)}{|\mathcal{S}_0|} + \frac{\sigma^2(\mathcal{S}_1)}{|\mathcal{S}_1|}}}$$

(where $\mu()$ is the sample mean, $\sigma^2()$ is the sample variance and $|\cdot|$ denotes the sample size) to determine if the samples in both sets were drawn from populations with the same mean (or from the same population). The null hypothesis is that the samples in both sets were drawn from populations with the same mean. The alternative hypothesis is that the samples in both sets were drawn from populations with different means.

In the final phase, the attacker computes the p value to determine if there is sufficient leakage to reject the null hypothesis at a particular significance level $(1 - \alpha)$. At each point in time, the test statistic t together with the degrees of freedom ν, computed with the Welch-Satterthwaite equation

$$\nu = \frac{(\sigma^2(\mathcal{S}_0)/|\mathcal{S}_0| + \sigma^2(\mathcal{S}_1)/|\mathcal{S}_1|)^2}{(\sigma^2(\mathcal{S}_0)/|\mathcal{S}_0|)^2/(|\mathcal{S}_0| - 1) + (\sigma^2(\mathcal{S}_1)/|\mathcal{S}_1|)^2/(|\mathcal{S}_1| - 1)} \, ,$$

allow to compute a p. The p value expresses the probability of observing the measured (or a greater) difference by chance if the null hypothesis was true. In other words, small p values give evidence to reject the null hypothesis.

2.4 Conclusion

The symmetric-key algorithms which we work with are SPNs formed from a layer of key XOR, a layer of substitution boxes and a layer of permutations. The S-boxes within the substitution layer are typically nonlinear permutations preferably on the small side for lightweight considerations. In this chapter, we provided foundations for nonlinear permutations up to size 8. We listed all the affine equivalence classes of 3- and 4-bit permutations to be able to analyze them systematically in the following chapters. We discussed AB and APN permutations together with the KECCAK S-box which are 5- or 6-bit permutations. We concluded the discussion on permutations by describing AES S-box of size 8 and its tower-field implementation for lightweight constructions.

In the second part of the preliminaries, we explained the basics of DPA. We described that a device leaks information from storing intermediate values to memory elements, from combinational operations to calculate these intermediate values and from the control logic. We mentioned that we use three types of DPA, namely CPA, CEPACA and t-test based leakage detection and argued that each of them has its advantages in different scenarios. We use CPA when we know the exact leakage model especially on memory elements, CEPACA when the leakage model is hard to predict such as gate-level combinational operations that can produce glitches on hardware and t-test based leakage detection to detect any leakage without the consideration of revealing the key.

It is important to state now that as the order of DPA increases, the required number of traces increases exponentially due to noise [68].

Finally we showed that the security against d^{th}-order DPA can be proven using the dth-order probing model. Namely, to be able to prove d^{th}-order security, we must show that an attacker observing d intermediate values never gets the sensitive information. We explained that standard masking fails to provide security in a circuit with glitches due to shared functions which possibly take all the shares as input.

"Experience is what you get when you didn't get what you wanted."

— Randy Pausch

3

Threshold Implementations

In this chapter, we theorize generating a threshold implementation (TI) of any function in order to use it as a dth-order DPA countermeasure on a device that reveals a linear combination of the intermediate values' noisy leakages. Hence, we answer the theoretical aspects of Question 1 in this chapter. We use the modified d-probing model adversary described in Section 2.3.4 in our proofs.

A TI, which is based on multi-party computation and secret sharing, satisfies four properties in order to achieve the mentioned security. In Section 3.1, we provide two of these properties which are common in all the masking schemes, namely *correctness* and *uniform masking*. Moreover, we introduce specific notations in that section. The threshold implementation technique, which inherits its name from threshold cryptography, is based on the observation that if an attacker probing d wires can not get information from all the shares then he can not reveal the secret information. The third property *non-completeness* describing this feature is given in Section 3.2 together with a discussion on how many shares are necessary to satisfy it. Early works of TI considers only one wire probing and proves that at least $t + 1$ shares are necessary where t is the algebraic degree of the function under consideration. We present the last property, that is *uniform sharing of a function*, following a discussion on the consequences of uniformity failure in Section 3.3. Even though the uniform sharing of a function was given as an important requirement in the early works, neither the consequences of abandoning this property, nor suggestions on how

to achieve uniformity was discussed by the time we started this research causing possible insecure implementations. We conclude that section by proposing several methods to fix such a uniformity failure. Finally, we explain the TI behavior under affine equivalence relations.

This chapter forms the core of the theoretical work done during this Ph.D. We make use of this theory in the following practical chapters. We note that the information provided in this chapter can be used to implement many other algorithms that are not considered in this thesis. The material treated here accumulated in collaboration with several co-authors [17, 18]. The discussion on TI of nonlinear functions using $d + 1$ input shares is a part of an ongoing work in collaboration with O. Reparaz, B. Gierlichs, S. Nikova and V. Rijmen.

3.1 Notation

In order to implement a function $f(X) = A$ from $\mathcal{F}^n$ to $\mathcal{F}^m$ with TI, we first split each variable x into s_x shares x_i, where $i \in \{1, 2, \ldots, s_x\}$, by means of Boolean masking. In order to do that, without loss of generality, we randomly choose the shares $x_1, \ldots, x_{s_x-1}$ from a uniform distribution then calculate x_{s_x} so that the XOR sum of these shares is equal to the variable itself ($x = \bigoplus_i x_i$). We refer to a valid share vector $\mathbf{x} = (x_1, \ldots, x_{s_x})$ and this splitting operation into shares as a *sharing* or *masking of the unshared value x*. Moreover we use the term s_x-sharing of x to emphasize the number of shares.

For all values x with $\Pr(X = x) > 0$, let $\mathrm{Sh}(x)$ denote the set of valid share vectors $\mathbf{x}$ for x:

$$\mathrm{Sh}(x) = \left\{ \mathbf{x} \in \mathcal{F}^{ns_x} \mid x_1 \oplus x_2 \oplus \cdots \oplus x_{s_x} = x \right\} .$$

$\Pr(\mathbf{X} = \mathbf{x} | X = x)$ denotes the probability that $\mathbf{X} = \mathbf{x}$ when the unshared input value of the masking equals x, taken over all auxiliary inputs of the masking. We use the same notation for the output $a \in \mathcal{F}^m$, and corresponding $s_a, \mathbf{a}, \mathrm{Sh}(a)$.

The masking $\mathbf{X}$ also satisfies the following.

Property 1 (Uniform masking). *A masking $\mathbf{X}$ is* uniform *if and only if there exists a constant p such that for all x we have:*

$$if \ \mathbf{x} \in \mathrm{Sh}(x) \ then \ \Pr(\mathbf{X} = \mathbf{x} | X = x) = p, \ else \ \Pr(\mathbf{X} = \mathbf{x} | X = x) = 0$$

and

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

In words, we call a masking uniform if for each value x, the corresponding vectors with masked values occur with the same probability.

We start with a lemma proving that uniformity of a masking implies the independence of the combination of any $s_x - 1$ shares from the unmasked value hence, satisfying an (s_x, s_x) *secret sharing* scheme. A (t, n) secret sharing [91] is defined as distributing parts of a secret x among n players such that the information from at least t players are required to calculate the secret. Let $\mathbf{X}_{\bar{i}}$ denote the vector obtained by removing X_i from $\mathbf{X}$.

Lemma 5. *If the masking $\mathbf{X}$ of X is uniform, then $\mathbf{x}_{\bar{i}}$ and x are independent (for any choice of i).*

Proof. Two stochastic functions are independent if and only if their joint distribution equals the product of their marginal distributions. Hence, we have to show for all i that

$$\forall \mathbf{x}_{\bar{i}}, x : \Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) = \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) \Pr(X = x).$$

Since $\Pr(A, B) = \Pr(B)\Pr(A|B)$, it suffices to show that $\forall \mathbf{x}_{\bar{i}}, x : \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}|X = x) = \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})$. We start from

$$\Pr(\mathbf{X} = \mathbf{x}|X = x) = \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}, X_i = x_i|X = x)$$

$$= \frac{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}, X_i = x_i)}{\Pr(X = x)}$$

$$= \frac{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}, X_i = x_i)}{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})} \frac{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})}{\Pr(X = x)}$$

$$= \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}|X = x) \Pr(X_i = x_i|X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}).$$

We know that the last factor equals 1 when $\mathbf{x} \in Sh(x)$ and zero otherwise. Hence, we obtain

$$\forall x : \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}|X = x) = p. \tag{3.1}$$

Now we can write (Bayes' Theorem):

$$\Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) = \sum_x \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}|X = x) \Pr(X = x)$$

$$= p \sum_x \Pr(X = x) = p. \tag{3.2}$$

The equality of Equations (3.1) and (3.2) proves the claim. $\square$

It follows that $p = |\mathcal{F}|^{n(1-s_x)}$.

Hence, the knowledge of up to $s_x - 1$ shares of a masking defined in Property 1 does not reveal any information on x.

In a TI, f is implemented as a vector of functions $\mathbf{f} = f_1, \ldots, f_{s_f}$ that takes $\mathbf{x}$ as input and outputs $\mathbf{a}$. Each function in this vector is called a *component function* and represented by f_i where $i \in \{1, \ldots, s_f\}$. From now on, we use the term (s_f)-*sharing of the function* to describe $\mathbf{f}$. $\mathbf{f}$ must satisfy the following property for a correct implementation.

Property 2 (Correctness)**.** *For all* $a \in \mathcal{F}_2^m$, $\mathbf{A} = \mathbf{f}(\mathbf{X})$ *implies that* $a = \sum_i a_i = \sum_i f_i(\mathbf{x})$ *for all* $\mathbf{x}$ *satisfying* $\sum_i x_i = x$ *and* $x \in \mathcal{F}_2^n$.

It is clear that the number of output shares s_a is equal to the number of component functions s_f from this property. From now on, we refer to s_x as the number of input shares s_{in} and s_f as the number of output shares s_{out}. If $s_{\text{in}} = s_{\text{out}}$, they are denoted by s.

Correctness of a sharing and the uniform masking of the input to this sharing are standard properties for all masking schemes. To achieve higher-order DPA security on hardware where glitches occur, one needs to follow two other properties given in the following sections. Failing to achieve any one of these properties can result in leakage of sensitive information. The non-completeness property given in Section 3.2 is the main difference compared to standard masking schemes and provides protection against DPA in a circuit with glitches. In Section 3.2.1, we discuss how many input and output shares are necessary to achieve the non-completeness property. To build TI on a big circuit that has cascaded or parallel operations, e.g. the whole block cipher, one needs the uniformity property which is defined in Section 3.3. After discussing several methods to achieve that, we conclude with some remarks and extensions.

3.2 Non-completeness

As mentioned in Section 2.3.2, most of the masking schemes use all s_{in} input shares jointly in at least one of their component functions. Therefore, an attacker probing the corresponding wire can observe all the information required to solve the $(s_{\text{in}}, s_{\text{in}})$ secret sharing. The threshold implementation technique aims that an attacker probing d wires can only observe information from at most $s_{\text{in}} - 1$ shares, which is independent of the sensitive information making a dth-order DPA infeasible by Lemma 4. Hence, if the input is a uniform masking and the

following property is satisfied, the shared function $\mathbf{f}$ achieves security against d^{th}-order DPA.

Property 3 (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.*

Property 3, which we shortly refer to as the non-completeness property in the rest of the thesis, is more general than the non-completeness property defined in the early works of TI [76] for first-order DPA resistance when $d = 1$. We define a sharing that satisfies correctness and non-completeness properties as d^{th}-order TI and prove its security with the following theorem.

Theorem 1. *If the input masking $\mathbf{X}$ of the shared function $\mathbf{f}$ is a uniform masking and $\mathbf{f}$ is a dth-order TI then the dth-order analysis on the power consumption of a circuit implementing $\mathbf{f}$ does not reveal the unmasked input value x even if the inputs are delayed or glitches occur in the circuit.*

Proof. By Lemma 4, it is sufficient to prove that an adversary who can probe d wires does not get any information about x. By construction, if $\mathbf{X}$ is a uniform masking and $\mathbf{f}$ satisfies correctness and non-completeness properties, an adversary who probes d or less wires will get information from all but at least one input share, which is independent of the input by Lemma 5. $\square$

Note that the only required assumption on the physical behavior of the hardware or software implementation of $\mathbf{f}$ is that the component functions can be implemented such that their leakages are independent of each other hence satisfying Property 3. In other words, the cross-talk between implementations of different components should be negligible. Note that this is a common and realistic assumption required for all masking schemes.

3.2.1 Number of Shares

An affine function $f(X) = A$ can be implemented with $s \geq d + 1$ component functions to thwart d^{th}-order DPA. Without loss of generality, one can generate $\mathbf{f}$ by defining the first component function to be $f_1(X_1) = A_1 = f(X_1)$ and the rest of the component functions to be $f_i(X_i) = A_i$ where f_i is equal to f without constant terms and $2 \leq i \leq s$. To give an example $f(X) = 1 \oplus X$ can be implemented with the following component functions:

$$f_1(X_1) = 1 \oplus X_1 \text{ and } f_i(X_i) = X_i, \text{ where } i \in \{2, \ldots, s\}.$$

Since every component function depends on only one share, it is clear that the non-completeness property is satisfied with $s_{\text{in}} = d + 1$.

However, the minimum number of shares required increases together with the degree of the function and the number of variables effecting this degree. Let's take the quadratic function $A = f(X, Y, Z) = 1 \oplus X \oplus XZ \oplus YZ$ where $X, Y, Z \in \mathcal{F}_2$. The Equation (3.3) with $s_{\text{in}} = 2$ and $s_{\text{out}} = 4$ is a first-order TI if the inputs X, Y and Z are independent of each other since probing any wire gives information from at most one share of each input.

$$A_1 = 1 \oplus X_1 \oplus X_1 Z_1 \oplus Y_1 Z_1$$

$$A_2 = X_1 Z_2 \oplus Y_1 Z_2$$

$$A_3 = X_2 \oplus X_2 Z_1 \oplus Y_2 Z_1 \tag{3.3}$$

$$A_4 = X_2 Z_2 \oplus Y_2 Z_2$$

On the other hand, if we look at the more complex function $A = f(X, Y, Z) = 1 \oplus X \oplus XY \oplus XZ \oplus YZ$, we observe that we can not achieve the non-completeness property with only four component functions. Note that the difference between these two unmasked equations is the term XY of which the sharings should be added to Equation (3.3) for a correct implementation. Even if we place the additional terms $X_1 Y_1$ and $X_2 Y_2$ from the sharing of XY to the first and the last statement in Equation (3.3) respectively, the rest of the terms $X_1 Y_2$ and $X_2 Y_1$ can not be placed in these four component functions without breaking the non-completeness property. Hence, we need to increase the number of shares. One option to obtain non-completeness is increasing the number of output shares as shown in the equation below.

$$A_1 = 1 \oplus X_1 \oplus X_1 Y_1 \oplus X_1 Z_1 \oplus Y_1 Z_1$$

$$A_2 = X_1 Z_2 \oplus Y_1 Z_2$$

$$A_3 = X_2 \oplus X_2 Z_1 \oplus Y_2 Z_1$$

$$A_4 = X_2 Y_2 \oplus X_2 Z_2 \oplus Y_2 Z_2 \tag{3.4}$$

$$A_5 = X_1 Y_2$$

$$A_6 = X_2 Y_1$$

Applying this option to permutations with more output bits causes an undesired increase in the number of output shares. Another option, on the other hand,

is to increase the number of input shares to provide a three-share alternative given in Equation (3.5) which is presented in the early works of TI [76].

$$A_1 = 1 \oplus X_2 \oplus (X_2 Y_2 \oplus X_2 Y_3 \oplus X_3 Y_2) \oplus (X_2 Z_2 \oplus X_2 Z_3 \oplus X_3 Z_2)$$

$$\oplus (Y_2 Z_2 \oplus Y_2 Z_3 \oplus Y_3 Z_2)$$

$$A_2 = X_3 \oplus (X_3 Y_3 \oplus X_3 Y_1 \oplus X_1 Y_3) \oplus (X_3 Z_3 \oplus X_3 Z_1 \oplus X_1 Z_3)$$

$$\oplus (Y_3 Z_3 \oplus Y_3 Z_1 \oplus Y_1 Z_3) \tag{3.5}$$

$$A_3 = X_1 \oplus (X_1 Y_1 \oplus X_1 Y_2 \oplus X_2 Y_1) \oplus (X_1 Z_1 \oplus X_1 Z_2 \oplus X_2 Z_1)$$

$$\oplus (Y_1 Z_1 \oplus Y_1 Z_2 \oplus Y_2 Z_1)$$

Observe that with the latter approach, the number of input and output shares for a first-order TI is the same which can be advantageous in some applications.

The first-order TI sharing of a quadratic function where the linear terms with indices i, the quadratic terms with indices both i and $i+1$ and the quadratic terms with only the indices i appear in the same component function f_{i-1} in a cyclic manner is called a *first-order direct sharing with three shares* (Equation (3.5)).

It is possible to extend the first-order sharing to higher degree functions by increasing the number of shares. Similar to the previous argument on sharing a quadratic function with two input shares, one way to achieve this is to increase the number of output shares as in Equation (3.6) which is given for the function $f(X, Y, Z) = 1 \oplus X \oplus XY \oplus XYZ$.

$$
\begin{aligned}
A_1 &= 1 \oplus X_1 \oplus X_1 Y_1 \oplus X_1 Y_1 Z_1 & A_2 &= X_1 Y_1 Z_1 \\
A_3 &= X_1 Y_2 \oplus X_1 Y_2 Z_1 & A_4 &= X_1 Y_2 Z_2 \\
A_5 &= X_2 \oplus X_2 Y_1 \oplus X_2 Y_1 Z_1 & A_6 &= X_2 Y_1 Z_2 \\
A_7 &= X_2 Y_2 \oplus X_2 Y_2 Z_1 & A_8 &= X_2 Y_2 Z_2
\end{aligned}
\tag{3.6}
$$

A similar limitation observed for Equation (3.3) also exists for this sharing if the number of distinct high-degree terms in the unshared function increases. In this case the alternative sharing given in Equation (3.7) can be used for the same function. This sharing, comparable to the one in Equation (3.5), has the same number of input and output shares, i.e. $s_{\text{in}} = s_{\text{out}} = 4$. TI of functions of up to degree three of which all the shared linear, quadratic and cubic terms are

distributed as in Equation (3.7) is referred to as *first-order direct sharing with four shares.*

$$
\begin{aligned}
A_1 =\ & 1 \oplus X_2 \oplus (X_2Y_2 \oplus X_2Y_3 \oplus X_2Y_4 \oplus X_4Y_3) \oplus (X_2Y_2Z_2 \oplus X_2Y_3Z_2 \oplus X_2Y_2Z_3 \\
& \oplus X_2Y_3Z_4 \oplus X_2Y_4Z_3 \oplus X_2Y_2Z_4 \oplus X_2Y_4Z_2 \oplus X_2Y_4Z_4 \oplus X_2Y_3Z_3 \oplus X_4Y_3Z_2 \\
& \oplus X_3Y_4Z_2 \oplus X_4Y_2Z_3 \oplus X_3Y_2Z_4 \oplus X_4Y_3Z_3 \oplus X_4Y_4Z_3 \oplus X_4Y_3Z_4) \\[4pt]
A_2 =\ & 1 \oplus X_3 \oplus (X_3Y_3 \oplus X_3Y_4 \oplus X_3Y_1 \oplus X_1Y_4) \oplus (X_3Y_3Z_3 \oplus X_3Y_4Z_3 \oplus X_3Y_3Z_4 \\
& \oplus X_3Y_4Z_1 \oplus X_3Y_1Z_4 \oplus X_3Y_3Z_1 \oplus X_3Y_1Z_3 \oplus X_3Y_1Z_1 \oplus X_3Y_4Z_4 \oplus X_1Y_4Z_3 \\
& \oplus X_4Y_1Z_3 \oplus X_1Y_3Z_4 \oplus X_4Y_3Z_1 \oplus X_1Y_4Z_4 \oplus X_1Y_1Z_4 \oplus X_1Y_4Z_1) \\[4pt]
A_3 =\ & 1 \oplus X_4 \oplus (X_4Y_4 \oplus X_4Y_1 \oplus X_4Y_2 \oplus X_2Y_1) \oplus (X_4Y_4Z_4 \oplus X_4Y_1Z_4 \oplus X_4Y_4Z_1 \\
& \oplus X_4Y_1Z_2 \oplus X_4Y_2Z_1 \oplus X_4Y_4Z_2 \oplus X_4Y_2Z_4 \oplus X_4Y_2Z_2 \oplus X_4Y_1Z_1 \oplus X_2Y_1Z_4 \\
& \oplus X_1Y_2Z_4 \oplus X_2Y_4Z_1 \oplus X_1Y_4Z_2 \oplus X_2Y_1Z_1 \oplus X_2Y_2Z_1 \oplus X_2Y_1Z_2) \\[4pt]
A_4 =\ & 1 \oplus X_1 \oplus (X_1Y_1 \oplus X_1Y_2 \oplus X_1Y_3 \oplus X_3Y_2) \oplus (X_1Y_1Z_1 \oplus X_1Y_2Z_1 \oplus X_1Y_1Z_2 \\
& \oplus X_1Y_2Z_3 \oplus X_1Y_3Z_2 \oplus X_1Y_1Z_3 \oplus X_1Y_3Z_1 \oplus X_1Y_3Z_3 \oplus X_1Y_2Z_2 \oplus X_3Y_2Z_1 \\
& \oplus X_2Y_3Z_1 \oplus X_3Y_1Z_2 \oplus X_2Y_1Z_3 \oplus X_3Y_2Z_2 \oplus X_3Y_3Z_2 \oplus X_3Y_2Z_3)
\end{aligned}
\tag{3.7}
$$

These ideas can also be carried to higher-order TIs. In Equation (3.8), we provide a second-order example for the function $A = f(X, Y, Z) = 1 \oplus X \oplus XZ \oplus YZ$ with $s_{\text{in}} = 3$ and $s_{\text{out}} = 9$ shares.

$$
\begin{aligned}
A_1 &= 1 \oplus X_1 \oplus X_1Y_1 \oplus Y_1Z_1 & A_2 &= X_1Z_2 \oplus Y_1Z_2 \\
A_3 &= X_1Z_3 \oplus Y_1Z_3 & A_4 &= X_2Z_1 \oplus Y_2Z_1 \\
A_5 &= X_2 \oplus X_2Z_2 \oplus Y_2Z_2 & A_6 &= X_2Z_3 \oplus Y_2Z_3 \\
A_7 &= X_3Z_1 \oplus Y_3Z_1 & A_8 &= X_3Z_2 \oplus Y_3Z_2 \\
A_9 &= X_3 \oplus X_3Z_3 \oplus Y_3Z_3 &&
\end{aligned}
\tag{3.8}
$$

Similar to the first-order TI case, this equation can not be extended to the function $A = f(X, Y, Z) = 1 \oplus X \oplus XY \oplus XZ \oplus YZ$ with $s_{\text{in}} = 3$ shares without breaking the non-completeness property. If this is the case, the following equation with $s_{\text{in}} = 5$ and $s_{\text{out}} = 10$ provided for the function $A = f(X, Y, Z) = 1 \oplus X \oplus YZ$ can be used. We refer to a sharing of a quadratic function of which

all the shared linear and quadratic terms are distributed as in Equation (3.9) as *second-order direct sharing with five input shares.*

$$
\begin{aligned}
A_1 &= 1 \oplus X_2 \oplus Y_2 Z_2 \oplus Y_1 Z_2 \oplus Y_2 Z_1 & A_2 &= X_3 \oplus Y_3 Z_3 \oplus Y_1 Z_3 \oplus Y_3 Z_1 \\
A_3 &= X_4 \oplus Y_4 Z_4 \oplus Y_1 Z_4 \oplus Y_4 Z_1 & A_4 &= X_1 \oplus Y_1 Z_1 \oplus Y_1 Z_5 \oplus Y_5 Z_1 \\
A_5 &= Y_2 Z_3 \oplus Y_3 Z_2 & A_6 &= Y_2 Z_4 \oplus Y_4 Z_2 \\
A_7 &= X_5 \oplus Y_5 Z_5 \oplus Y_2 Z_5 \oplus Y_5 Z_2 & A_8 &= Y_3 Z_4 \oplus Y_4 Z_3 \\
A_9 &= Y_3 Z_5 \oplus Y_5 Z_3 & A_{10} &= Y_4 Z_5 \oplus Y_5 Z_4
\end{aligned}
\tag{3.9}
$$

These examples show how the degree and the function representation of an S-box change the required number of shares. Moreover, the number of input and output shares have an important role on the cost of the TI. Namely, the number of ANDs and XORs hence, the area of the combinational logic increase together with the number of input shares and the number of registers increase together with the number of input/output shares. Therefore, we will provide the minimum number of required input shares for a TI.

Lemma 6. *The minimum number of input shares required for implementing a dth-order TI of a function with independent inputs is*

$$
s_{in} \geq d + 1.
$$

Proof. Consider the product(s) $X^{j_1} X^{j_2} X^{j_3} \dots X^{j_t}$ of t variables where $\mathcal{F}^n \ni X = (X^1, X^2, \dots, X^n)$ and $X^j \in \mathcal{F}$. We represent the sharing of each variable X^j as X_i^j where $i \in \{1, \dots, s_{\text{in}}\}$. Then,

$$
\begin{aligned}
X^{j_1} X^{j_2} X^{j_3} \dots X^{j_t} &= (X_1^{j_1} + X_2^{j_1} + \dots + X_{s_{in}}^{j_1}) \dots (X_1^{j_t} + X_2^{j_t} + \dots + X_{s_{in}}^{j_t}) \\
&= (X_1^{j_1} X_1^{j_2} \dots X_1^{j_t}) + (X_1^{j_1} X_1^{j_2} \dots X_2^{j_t}) + \dots + (X_{s_{in}}^{j_1} X_{s_{in}}^{j_2} \dots X_{s_{in}}^{j_t}).
\end{aligned}
$$

Consider a correct sharing where each term in the above sum belongs to a different component function. Given the independence of the inputs, each component function carries information from at most one share of each input variable. Hence, any combination of up to d component functions carries information from at most d shares of an input. To achieve the non-completeness property, $s_{\text{in}} > d$ which implies the equation $s_{\text{in}} \geq d + 1$ for the number of input shares. $\qquad\square$

However, choosing $s_{\text{in}} = d + 1$ causes a huge increase of the number of output shares as the number of terms in the function representation of a function

increases since each component function uses only one share of each variable. Moreover, the independence of each input variable can be a limiting factor. Namely, without the independence limitation, the second output share A_2 in Equation (3.3) can reveal information from all the shares of its inputs. On the other hand, we show in the following theorem that there is always a structured way to generate the component functions using more than one input share hence possibly decreasing the number of output shares. This sharing additionally eliminates the independence of the unshared inputs limitation.

Theorem 2. *There always exist a dth-order TI of a function of degree t that requires $s_{in} \geq t \times d + 1$ input and $s_{out} \geq \binom{s_{in}}{t}$ output shares.*

Proof. Consider the product(s) $X^{j_1} X^{j_2} X^{j_3} \ldots X^{j_t}$ of t variables where $\mathcal{F}^n \ni X = (X^1, X^2, \ldots, X^n)$ and $X^j \in \mathcal{F}$. We represent the sharing of each variable X^j as X_i^j where $i \in \{1, \ldots, s_{in}\}$. Then,

$$X^{j_1} X^{j_2} X^{j_3} \ldots X^{j_t} = (X_1^{j_1} + X_2^{j_1} + \cdots + X_{s_{in}}^{j_1}) \ldots (X_1^{j_t} + X_2^{j_t} + \cdots + X_{s_{in}}^{j_t})$$

$$= (X_1^{j_1} X_1^{j_2} \ldots X_1^{j_t}) + (X_1^{j_1} X_1^{j_2} \ldots X_2^{j_t}) + \ldots + (X_{s_{in}}^{j_1} X_{s_{in}}^{j_2} \ldots X_{s_{in}}^{j_t}).$$

To satisfy the correctness each term in the above sum should belong to at least one component function. This can be done in the following way. Let each component function use only t different shares (indices) such that any t combination of s_{in} shares is used by only one component function. Hence, any combination of up to d component functions carries information from at most $t \times d$ shares. To achieve the non-completeness property, $s_{in} > t \times d$ which implies the equation $s_{in} \geq t \times d + 1$ for the number of input shares. With the given sharing, there exist $\binom{s_{in}}{t}$ different ways of choosing t combination of s_{in} shares and placing them in component functions. Hence, this sharing needs $s_{out} \geq \binom{s_{in}}{t}$ component functions. $\square$

Remark 1. *Each component function of a dth-order TI that is defined as in the proof of Theorem 2 uses at most t input shares where t is the degree of the function. Hence, each component function is independent of $s_{in} - t$ input shares.*

The required number of input and output shares given in Theorem 2 correspond to the number of shares provided in the earlier works [76] for $d = 1$. Hence, the property of each component function being independent of at least one input share defined in [76] is also satisfied.

The sharings provided in Equations (3.5) and (3.9) are examples of first- and second-order TI applied to a quadratic Boolean function using the technique

described in Theorem 2. We will use the same structure to generate component functions unless stated otherwise. We point out that the theorem does not imply that the number of input, output shares or their sum are minimized. A second-order TI with six input and seven output shares (hence the sum of input output shares is 13 instead of 15) is given for $A = 1 \oplus X \oplus YZ$ in Equation (3.10) as a counter example. It is still an open question to find a lower bound for $s_{\text{in}} + s_{\text{out}}$. We leave the analysis of TIs with $t \times d + 1 > s_{\text{in}} \geq d + 1$ as a future work.

$$A_1 = 1 + X_2 + Y_2 Z_2 + Y_1 Z_2 + Y_2 Z_1 + Y_1 Z_3 + Y_3 Z_1 + Y_2 Z_3 + Y_3 Z_2$$

$$A_2 = X_3 + Y_3 Z_3 + Y_3 Z_4 + Y_4 Z_3 + Y_3 Z_5 + Y_5 Z_3$$

$$A_3 = X_4 + Y_4 Z_4 + Y_2 Z_4 + Y_4 Z_2 + Y_2 Z_6 + Y_6 Z_2$$

$$A_4 = X_5 + Y_5 Z_5 + Y_1 Z_4 + Y_4 Z_1 + Y_1 Z_5 + Y_5 Z_1 \tag{3.10}$$

$$A_5 = Y_2 Z_5 + Y_5 Z_2 + Y_4 Z_5 + Y_5 Z_4$$

$$A_6 = X_6 + Y_6 Z_6 + Y_3 Z_6 + Y_6 Z_3 + Y_4 Z_6 + Y_6 Z_4$$

$$A_7 = X_1 + Y_1 Z_1 + Y_1 Z_6 + Y_6 Z_1 + Y_5 Z_6 + Y_6 Z_5$$

3.3 Uniformity

So far, we have shown that a function f can be implemented in a way that it is secure against d^{th}-order DPA with the requirement that uniform masking, correctness and non-completeness properties are satisfied. In the following we will discuss the consequences of having a non-uniform masking in the input.

3.3.1 Analyzing the Lack of Uniformity

Let $(X, Y) \in \mathcal{F}_2^2$ and $\mathcal{F}_2 \ni A = f(X, Y) = XY$. Define $\mathbf{f}$ corresponding to first-order TI of f as follows:

$$A_1 = f_1(X_2, X_3, Y_2, Y_3) = X_2 Y_2 \oplus X_2 Y_3 \oplus X_3 Y_2$$

$$A_2 = f_2(X_1, X_3, Y_1, Y_3) = X_3 Y_3 \oplus X_1 Y_3 \oplus X_3 Y_1 \tag{3.11}$$

$$A_3 = f_3(X_1, X_2, Y_1, Y_2) = X_1 Y_1 \oplus X_1 Y_2 \oplus X_2 Y_1.$$

If the masking of the input (X, Y) is uniform, then the masking of A is distributed as shown in Table 3.1. In order to satisfy the uniformity of masking definition for the output $\mathbf{A}$, we would need that the 16 non-zero values in the table were equal (specifically to $2^{2(3-1)-1(3-1)} = 4$ as will be defined in Property 4).

Table 3.1: Number of times that a masking a_1, a_2, a_3 occurs for a given input (x, y)

	a_1, a_2, a_3							
(x, y)	000	011	101	110	001	010	100	111
$(0,0)$	7	3	3	3	0	0	0	0
$(0,1)$	7	3	3	3	0	0	0	0
$(1,0)$	7	3	3	3	0	0	0	0
$(1,1)$	0	0	0	0	5	5	5	1

Theorem 1 implies that there is no leakage of information in *this* circuit. However, if $\mathbf{A}$ is used as input of a second circuit, then Theorem 1 does not apply anymore to the second circuit (because its inputs are not uniform) and potentially the second circuit might leak information.

Let $B = g(Z, A) = ZA$ and let this multiplication be implemented by similar formulas as above. For example, Equation (3.11) becomes:

$$B_1 = g_1(Z_2, Z_3, A_2, A_3) = Z_2 A_2 + Z_2 A_3 + Z_3 A_2 . \tag{3.12}$$

Assume that the masking of Z is uniform but the masking of A has the distribution given in Table 3.1. Then the masking of B will be distributed as shown in Table 3.2. Remember from Section 2.3.3 that a first-order DPA exploits information from first-order statistical moment, namely the deviations in the mean values of leakages corresponding to different inputs. Without loss of generality[1], we can assume a HW leakage model as described in Section 2.3. The average HW of B_1, B_2, B_3 in the seventh row $((x, y, z) = (1, 1, 0))$ equals $33/32$, whereas it equals $27/32$ in the first six rows. This implies that some hardware implementations might show a different average power consumption when $(x, y, z) = (1, 1, 0)$. Observe also that $\text{cor}(B_i, B) = 0.125$. Hence, in the part of the circuit implementing Equation (3.12), the average of the leakage $\mathcal{L}$ can be correlated to B, since both B_2 and B_3 are correlated to B. Similar arguments would follow for a nonuniform higher-order TI if higher-order statistical moments

—————————————————————

[1]We assume that each component function has its own circuit. If the component functions are the same and one single circuit is used for all shared calculations, transition based leakages such as HD model can cause a decrease in the security order [5].

Table 3.2: Number of times that a masking b_1, b_2, b_3 occurs for a given input (x, y, z)

	b_1, b_2, b_3							
(x, y, z)	000	011	101	110	001	010	100	111
$(0, 0, 0)$	37	9	9	9	0	0	0	0
$(0, 0, 1)$	37	9	9	9	0	0	0	0
$(0, 1, 0)$	37	9	9	9	0	0	0	0
$(0, 1, 1)$	37	9	9	9	0	0	0	0
$(1, 0, 0)$	37	9	9	9	0	0	0	0
$(1, 0, 1)$	37	9	9	9	0	0	0	0
$(1, 1, 0)$	31	11	11	11	0	0	0	0
$(1, 1, 1)$	0	0	0	0	21	21	21	1

such as the variance distribution of a leakage is considered leading to a possibly successful second-order DPA.

Notice that if the function g following the nonlinear operation f was linear and it is shared as defined in the beginning of Section 3.2.1, the output distribution of f would be carried to the output of g and would still be secure. Hence, we face this problem only when the input sharing of a *nonlinear* function is not uniform. Therefore we need to make sure that the input of a sharing $\mathbf{g}$ of a nonlinear function g which follows $\mathbf{f}$ is also a uniform masking. This is equivalent to saying that $\mathbf{f}$ should be a uniform sharing of the function f as defined by the following property.

Property 4 (Uniform sharing of a function). *The dth-order sharing $\mathbf{f}$ is uniform if and only if*

$$\forall x \in \mathcal{F}^n, \forall a \in \mathcal{F}^m \text{ with } f(x) = a, \forall \mathbf{a} \in Sh(a) \text{ and } s_{out} \geq d + 1 :$$

$$|\{\mathbf{x} \in Sh(x) | \mathbf{f}(\mathbf{x}) = \mathbf{a}\}| = \frac{|\mathcal{F}|^{n(s_{in}-1)}}{|\mathcal{F}|^{m(s_{out}-1)}} .$$

If $s_{in} = s_{out}$ and $n = m$, this simplifies to:

$$\forall x, a \in \mathcal{F}^n \text{ with } f(x) = a, \forall \mathbf{a} \in Sh(a) : |\{\mathbf{x} \in Sh(x) | \mathbf{f}(\mathbf{x}) = \mathbf{a}\}| = 1 .$$

It follows that a uniform circuit $\mathbf{f}$ with $s_{in} = s_{out}$ is invertible if and only if f is invertible. Moreover if $n < m$, then uniformity can be achieved only if $s_{in} > s_{out}$. We now prove that the uniform circuit condition is sufficient to achieve a uniform distribution at the output.

Theorem 3. *If the masking $\mathbf{X}$ is uniform and the circuit $\mathbf{f}$ is uniform, then the masking $\mathbf{A}$ of $A = f(X)$, defined by $\mathbf{A} = \mathbf{f}(\mathbf{X})$ is uniform.*

Proof. In order to prove that $\mathbf{A}$ is uniform, we need to show that $\Pr(\mathbf{A} = \mathbf{a}|A = a)$ is equal to a constant p if $\mathbf{a} \in \mathrm{Sh}(a)$ and 0 otherwise by Property 1. Considering $\mathbf{a} = \mathbf{f}(\mathbf{x})$ and $a = f(x)$, we obtain:

$$\Pr(\mathbf{A} = \mathbf{a}|A = a)$$

$$= \sum_{\substack{\mathbf{x} \in \mathrm{Sh}(x), \\ x, f(x) = a}} \Pr(\mathbf{A} = \mathbf{f}(\mathbf{x})|A = f(x)) \Pr(\mathbf{X} = \mathbf{x}, X = x).$$

Using the equality

$$\Pr(\mathbf{X} = \mathbf{x}, X = x) = \Pr(\mathbf{X} = \mathbf{x}|X = x) \Pr(X = x)$$

and Property 1, the second factor becomes $p' \Pr(X = x)$. The proof of Lemma 5 implied that $p' = |\mathcal{F}|^{-n(s_{\mathrm{in}}-1)}$. Property 4 implies that the first factor equals $|\mathcal{F}|^{n(s_{\mathrm{in}}-1)-m(s_{\mathrm{out}}-1)}$ for all $\mathbf{a} \in Sh(a)$. We obtain

$$\Pr(\mathbf{A} = \mathbf{a}|A = a) = \sum_{\substack{\mathbf{x} \in \mathrm{Sh}(x), \\ x, f(x) = y}} |\mathcal{F}|^{-m(s_{\mathrm{out}}-1)} \Pr(X = x) = |\mathcal{F}|^{-m(s_{\mathrm{out}}-1)}.$$

Hence, $\Pr(\mathbf{A} = \mathbf{a}|A = a)$ is equal to a constant $p = |\mathcal{F}|^{-m(s_{\mathrm{out}}-1)}$ if $\mathbf{a} \in \mathrm{Sh}(a)$ and 0 otherwise satisfying a uniform masking. $\square$

We call a dth-order TI that is a uniform sharing a uniform d^{th}-order TI.

Assume that the sharing $\mathbf{B} = \mathbf{g}(\mathbf{A})$ is taking the output of the uniform sharing $\mathbf{A} = \mathbf{f}(\mathbf{X})$. We still need to be careful to satisfy the non-completeness (Property 3) in the cascaded function $\mathbf{h} = \mathbf{g} \circ \mathbf{f}$. As an example we can assume that $\mathbf{f}$ and $\mathbf{g}$ are uniform first-order 3-sharings where $A_1 = F_1(X_1, X_2)$ and $A_2 = f_2(X_2, X_3)$. The function $g_1(A_1, A_2)$ can also be written as $g_1(X_1, X_2, X_3)$. In that case, a glitch in that function can produce a leakage that depends on all the shares of the value X. We can avoid this by dividing these two nonlinear operations with a register which disallows the propagation of a glitch that effects all the shares of an unmasked value. Hence, a leakage will still be independent of the unmasked value.

Remark 2. *If the output of a nonlinear function is used as an input to a nonlinear function, it is important to separate these shared functions by means of a register even if the functions have uniform TIs.*

Achieving Property 4 is important not only for the cascaded functions, but also for functions that are acting in parallel on (partially) the same input. If no special care is taken, then "local uniformity" of the distributions of the outputs of the individual functions will not lead to "global uniformity", i.e. for the joint distributions of the outputs of all blocks. For example, let $\mathbf{f}, \mathbf{g}$ be two functions acting on the same uniform input $\mathbf{X}$. Then, even if $\mathbf{f}, \mathbf{g}$ are uniformly shared functions, producing uniform $\mathbf{A} = \mathbf{f}(\mathbf{X})$ and $\mathbf{A}' = \mathbf{g}(\mathbf{X})$, this does not imply that $(\mathbf{A}, \mathbf{A}')$ is uniform. Like with cascaded functions, if each of the parallel blocks satisfies Properties 1, 2 and 3, there will be no leakage of signals within the parallel blocks, but the lack of uniformity in the joint distribution of the masking of the outputs can lead to information leakage if the outputs are combined as inputs to a next function.

Unfortunately, we do not know a straightforward way to generate the component functions so that Property 3 and 4 hold jointly (unlike the other properties individually) for any Boolean function. Hence, each dth-order non-complete sharing, should be checked in order to assure the uniformity property. Moreover, practice shows that adding the uniformity requirement to a sharing tends to blow up the mathematical complexity of the sharing, as well as the cost of implementation. We can take different types of actions to remedy these problems which we will discuss in the following section.

3.3.2   Achieving Uniformity of a Shared Function

As discussed in the previous subsection, uniformity has a particular importance for TI of the entire algorithm which is possibly composed of several layers of linear and nonlinear functions. It is known that a linear function would carry the input distribution to its output not requiring a further investigation. Unfortunately, satisfying uniformity of the sharing is not trivial when nonlinear functions are considered. In the following, we suggest several options to treat this problem.

Re-masking

Re-masking is a technique to make a nonuniform sharing of a value uniform by introducing extra fresh randomness in the circuit. It is initially used by Moradi et al. [73] in the first-order TI context on a three-share implementation as given below.

If the sharing $\mathbf{A} = (A_1, A_2, A_3)$ that is the output of the nonuniform sharing $\mathbf{f}$ is followed by the re-masking operation $\mathbf{r}$ provided in Equation (3.13) that uses two random masks M_1, M_2, then the output sharing $\mathbf{B}$ is uniform.

$$B_1 = r_1(A_1, M_1) = A_1 \oplus M_1$$

$$B_2 = r_2(A_2, M_2) = A_2 \oplus M_2 \tag{3.13}$$

$$B_3 = r_3(A_3, M_1, M_2) = A_3 \oplus M_1 \oplus M_2.$$

This re-masking can be extended to any number of shares to generate uniform sharings.

Definition 4 (Re-masking). *The shared function $\mathbf{r}$ that takes a sharing $\mathbf{A}$ with s_{out} shares, which can be a result of a non-uniform TI, and $s_{out} - 1$ randomly generated numbers (masks) $(M_1, M_2, \ldots, M_{s_{out}-1})$ as input and is defined as follows without loss of generality produces a uniform sharing $\mathbf{B}$.*

$$B_1 = r_1(A_1, M_1) = A_1 \oplus M_1$$

$$B_2 = r_2(A_2, M_2) = A_2 \oplus M_2$$

$$\vdots$$

$$B_{s_{out}-1} = r_{s_{out}-1}(A_{s_{out}-1}, M_{s_{out}-1}) = A_{s_{out}-1} \oplus M_{s_{out}-1}$$

$$B_{s_{out}} = r_{s_{out}}(A_{s_{out}}, M_1, M_2, \ldots M_{s_{out}-1}) = A_{s_{out}} \oplus \bigoplus M_i$$

This function and this operation is called re-masking.

Even though re-masking is a technique that can be used when finding uniform sharings is unattainable otherwise, it should not be considered as a straightforward approach since generation of good masks can be a burden.

The following theorem allows to reduce the amount of random bits used by re-masking steps of threshold implementations: under certain circumstances, only a fraction of the shares needs to be re-masked.

Theorem 4. *Let $(X_1, X_2, \ldots, X_s)$ be a sharing of a (stochastic) variable $X \in \mathcal{F}^n$, where $\Pr(X_1 = x_1, \ldots, X_t = x_t) = |\mathcal{F}|^{-tn}, \forall(x_1, \ldots, x_t)$ for some t with $1 \leq t \leq s$. Then the sharing $(A_1, \ldots, A_s)$, defined by $A_i = X_i$ for $1 \leq i \leq t$ and*

$A_i = X_i \oplus M_i$ for $t < i \leq s$, is a uniform *sharing for X provided that the masks M_i, $i = t+1, \ldots, s-1$ are independently and uniformly distributed random variables and that $M_s = M_{t+1} \oplus \cdots \oplus M_{s-1}$.*

Proof. Showing

$$\Pr(A_1 = a_1, A_2 = a_2, \ldots, A_s = a_s | X = a_1 \oplus a_2 \oplus \cdots a_s) = |\mathcal{F}|^{n(1-s)}$$

proves that the described re-masking produces a uniform sharing. We have:

$$\Pr(A_1 = a_1, \ldots, A_s = a_s | X = a_1 \oplus a_2 \oplus \cdots \oplus a_s)$$

$$= \quad \Pr(A_1 = a_1, \ldots, A_t = a_t | X = a_1 \oplus a_2 \oplus \cdots \oplus a_s) \tag{3.14}$$

$$\cdot \Pr(A_{t+1} = a_{t+1}, \ldots, A_s = a_s | X = a_1 \oplus a_2 \oplus \cdots \oplus a_s, A_1 = a_1, \ldots, A_t = a_t).$$

Since $A_i = X_i$ for $1 \leq i \leq t$, the first factor equals $|\mathcal{F}|^{-tn}$. For the second factor we recall the definition of A_{t+1} to obtain that:

$$\Pr(A_{t+1} = a_{t+1}) = \sum_{x_{t+1}} \Pr(X_{t+1} = x_{t+1}) \underbrace{\Pr(M_{t+1} = a_{t+1} \oplus x_{t+1})}_{|\mathcal{F}|^{-n}} \quad .$$

The same holds for $A_{t+2}, \ldots, A_{s-1}$ and since the M_i have independent distributions, we can equate the second factor of (3.14) to:

$$|\mathcal{F}|^{(1-s-t)n} \sum_{x_{t+1}, \ldots, x_{s-1}} \Pr(X_{t+1} = x_{t+1}, \ldots, X_{s-1} = x_{s-1}, A_s = a_s | X = a_1 \oplus \cdots \oplus a_s, X_1 = x_1, \ldots, X_t = x_t).$$

Recalling the definition of A_s completes the proof. □

Clearly, the extra randomness required by the re-masking approach may be a worse problem than the blow-up in gate count caused by the uniform sharing approach in some cases. However, we want to stress the following.

Observation 1. *An implementation that uses re-masking, does not need uniform sharings in order to resist DPA attacks.*

Theorem 1 can be proven using only Properties 1, 2 and 3. For example, Property 4 is needed if several circuits are cascaded (*pipelined*), and even then it can be ignored if re-masking is used. In other words, there is no need to demand uniformity of a circuit that is followed by a re-masking step anyway.

By relinquishing the uniformity, it is often possible to reduce the number of shares and the size of the circuit.

Note also that this re-masking operation can be used to increase the number of shares if more than the required amount of masks are used. Specifically, each extra mask creates a new share while the XOR of these extra masks are added to one of the shares of a uniform sharing existed before the re-masking.

Increasing the Number of Input Shares

In both Lemma 6 and Theorem 2, we provide a lower bound on the number of required input shares implying that it is possible to increase the number of input shares without breaking the non-completeness requirement. The entropy introduced in the system increases together with the number of input shares. The increase of entropy can be beneficial during the search for a uniform TI. The first example of this argument is given in [75] on the function $A = f(X, Y) = XY$. Remember from Section 3.3.1 that the direct three-share TI of this function is not uniform. It has been shown in [75] that Equation (3.15) with four input and output shares satisfies all TI properties.

$$A_1 = (X_3 \oplus X_4)(Y_2 \oplus Y_3) \oplus Y_2 \oplus Y_3 \oplus Y_4 \oplus X_2 \oplus X_3 \oplus X_4$$

$$A_2 = (X_1 \oplus X_3)(Y_1 \oplus Y_4) \oplus Y_1 \oplus Y_3 \oplus Y_4 \oplus X_1 \oplus X_3 \oplus X_4 \qquad (3.15)$$

$$A_3 = (X_2 \oplus X_4)(Y_1 \oplus Y_4) \oplus Y_2 \oplus X_2$$

$$A_4 = (X_1 \oplus X_2)(Y_2 \oplus Y_3) \oplus Y_1 \oplus X_1$$

In addition, first-order direct sharing with four shares also provides a uniform TI for the mentioned function. We note that the number of output shares in Equation (3.15) is less than the suggested amount in Theorem 2. We remind again with this example that the construction in the theorem is not necessarily the optimal one.

Decreasing the Number of Output Shares

With the constructions described in Section 3.2.1, we see that the number of output shares can become greater than the number of input shares. It can be useful to decrease the number of shares to achieve Property 4. Moreover, we also avoid further increase in shares and hence in area by decreasing the number

of shares after nonlinear operations. This decrease can be done by combining different shares with an affine function as described in the following theorem.

Theorem 5. *Given $s_{in} \geq d + r$ input shares, where $r \geq 1$, that are not necessarily uniform but secure against $(d + r - 1)^{st}$-order DPA, any sharing* **g** *that combines any r of the input shares linearly in one component function and keeps the rest of the input shares unchanged, is secure against d^{th}-order DPA.*

Proof. We represent the shared variable A which is not necessarily a uniform masking using $s_{\text{in}} \geq d + r$ shares for a given d. Assume that this initial masking of A is secure against $(d + r - 1)^{\text{st}}$-order DPA. That implies that combining any $d + r - 1$ shares does not reveal the unmasked value A. Consider $s_{\text{in}} - 1$ component functions: the first component function combines, without loss of generality, the first two input shares linearly; each of the other component functions takes one share as input and outputs it unchanged, i.e. $B_1 = g_1(A_1, A_2) = A_1 \oplus A_2$ and $B_{i-1} = g_{i-1}(A_i) = A_i$ for $3 \leq i \leq s_{\text{in}}$. This construction satisfies both Property 2 and Property 3 for $(d + r - 2)^{\text{nd}}$-order security and one needs $s_{\text{in}} - 1 \geq d + r - 1$ shares to reveal the unmasked variable. Moreover, the component function g_1 only uses a balanced gate. Namely, a 2×1 XOR gate whose output changes with probability 1 for any input bit change, independent of the input value. Hence, even though the input is not uniform, this circuit **g** will not leak information. A mere $r - 1$ iterative repetition of this procedure gives a sharing C with $d + 1$ shares that satisfies Property 2 and Property 3 and that is hence d^{th}-order DPA secure. Moreover, since there are only balanced gates involved, one can combine this repetitive construction in one step. $\square$

Remark 3. *To satisfy Property 3, the nonlinear operation generating the sharing (C mentioned in the proof of Theorem 5) and the operation to decrease the number of shares should be separated by registers.*

Note that the statement being secure against $(d + r - 1)^{\text{st}}$-order DPA in Theorem 5 is very important. Consider the following sharing with two input and four output shares for $A = XY \oplus Z$.

$$A_1 = X_1 Y_1 \oplus Z_1$$

$$A_2 = X_1 Y_2$$

$$A_3 = X_2 Y_1 \oplus Z_2$$

$$A_4 = X_2 Y_2$$

Even though the value A is a 4-sharing, it is secure against first-order DPA only. Hence, the shares of A can not be combined using Theorem 5. Therefore any reduction of number of shares should be done with extreme care not to unmask a sensitive value and reveal information. If the reduction function output $\mathbf{B}$ is defined as $B_1 = A_1 \oplus A_3$ and $B_2 = A_2 \oplus A_4$, the sharing $\mathbf{B}$ would leak first-order information since $B_2 = XY_2$ reveals the unmasked value X. On the other hand if the reduction function is defined as $B_1 = A_1 \oplus A_2$ and $B_2 = A_3 \oplus A_4$, the sharing $\mathbf{B}$ would be first-order secure.

Given the above discussion, for Equation (3.9) which represents a second-order TI of a quadratic function, one possible way to decrease the shares such that $\mathbf{X}$ and $\mathbf{B}$ are represented with the same number of shares is given below.

$$B_i = A_i, \text{ where } i < 5 \text{ and } B_5 = A_5 + A_6 + A_7 + A_8 + A_9 + A_{10}. \quad (3.16)$$

With both of these TIs, it is important to make sure that Remark 3 is applied by using registers after the nonlinear operation $\mathbf{f}$.

As mentioned before, there are many ways to achieve uniformity of the output of a shared function. So far we have discussed the options where we change the number of shares to achieve this property or we used extra randomness. We can also modify the component functions that we have provided in Section 3.2.1.

Using Correction Terms

The use of *correction terms* (CT) as defined below has initially been proposed in [75].

Definition 5 (Correction terms). *Terms that can be added in pairs to more than one share, such that they satisfy the non-completeness property are called correction terms. Since the terms in a pair cancel each other, the sharing still satisfies Property 2.*

Let's take the example provided in Equation (3.5) for the first-order TI of the quadratic function $A = f(X, Y, Z) = 1 \oplus X \oplus XY \oplus XZ \oplus YZ$ where X, Y, Z and $A \in \mathcal{F}_2$ that is not a uniform sharing. If we apply the CT to the previous equation as highlighted in Equation (3.17), we get a sharing that is uniform.

By varying the CT one can obtain all possible sharings of a given function. Consider a Boolean quadratic function with n variables $(X, Y, \ldots, Z)$ (1 output

bit), which we want to share with three shares similar to the Equation (3.5) against first-order DPA. Note that the only terms which can be used as CT without increasing the degree of the component functions are degree 1 terms (e.g. X_i) or degree 2 terms (e.g. $X_i Y_i$) for $i = 1, 2, 3$. Indeed terms like $X_i Y_j$ for $i \neq j$ cannot be used in the ith and j^{th} share of the function because of the non-completeness property, i.e. such a term can be used in only one out of three shares, hence it cannot be used as a CT. We can also use higher degree terms such as $X_i Y_i Z_i$ as CT if we do not limit the CT to the function degree.

$$
\begin{aligned}
A_1 &= 1 \oplus X_2 \oplus (X_2 Y_2 \oplus X_2 Y_3 \oplus X_3 Y_2) \oplus (X_2 Z_2 \oplus X_2 Z_3 \oplus X_3 Z_2) \\
&\quad \oplus (Y_2 Z_2 \oplus Y_2 Z_3 \oplus Y_3 Z_2) \oplus \boldsymbol{X_2} \oplus \boldsymbol{X_3} \\
&= 1 \oplus \boldsymbol{X_3} \oplus (X_2 Y_2 \oplus X_2 Y_3 \oplus X_3 Y_2) \oplus (X_2 Z_2 \oplus X_2 Z_3 \oplus X_3 Z_2) \\
&\quad \oplus (Y_2 Z_2 \oplus Y_2 Z_3 \oplus Y_3 Z_2) \\
A_2 &= X_3 \oplus (X_3 Y_3 \oplus X_3 Y_1 \oplus X_1 Y_3) \oplus (X_3 Z_3 \oplus X_3 Z_1 \oplus X_1 Z_3) \\
&\quad \oplus (Y_3 Z_3 \oplus Y_3 Z_1 \oplus Y_1 Z_3) \oplus \boldsymbol{X_3} \oplus \boldsymbol{X_1} \\
&= \boldsymbol{X_1} \oplus (X_3 Y_3 \oplus X_3 Y_1 \oplus X_1 Y_3) \oplus (X_3 Z_3 \oplus X_3 Z_1 \oplus X_1 Z_3) \\
&\quad \oplus (Y_3 Z_3 \oplus Y_3 Z_1 \oplus Y_1 Z_3) \\
A_3 &= X_1 \oplus (X_1 Y_1 \oplus X_1 Y_2 \oplus X_2 Y_1) \oplus (X_1 Z_1 \oplus X_1 Z_2 \oplus X_2 Z_1) \\
&\quad \oplus (Y_1 Z_1 \oplus Y_1 Z_2 \oplus Y_2 Z_1) \oplus \boldsymbol{X_1} \oplus \boldsymbol{X_2} \\
&= \boldsymbol{X_2} \oplus (X_1 Y_1 \oplus X_1 Y_2 \oplus X_2 Y_1) \oplus (X_1 Z_1 \oplus X_1 Z_2 \oplus X_2 Z_1) \\
&\quad \oplus (Y_1 Z_1 \oplus Y_1 Z_2 \oplus Y_2 Z_1)
\end{aligned}
\tag{3.17}
$$

Counting the linear, quadratic and cubic CT and ignoring the constant terms, which will not influence the uniformity, we obtain

$$
3\left(n + \binom{n}{2} + \binom{n}{3}\right)
$$

CT. Taking into account all possible positions for the CT we get

$$
2^{3\left(n + \binom{n}{2} + \binom{n}{3}\right)}
$$

different sharings. For example, for a quadratic function of 3 variables, which we want to have a first-order three-sharing starting from the direct sharing, there are 2^{21} possible CT.

If we start from a sharing which is not a direct sharing as in Equation (3.4), counting the amount of CT and using them becomes more demanding than direct sharing due to the limitations derived from the Boolean function structure. Therefore throughout this thesis, we will not consider CT for those sharings.

Virtual Variable and Virtual Shares

So far, we always assumed that if the domain of f is $\mathcal{F}^n$, then the domain of its s-sharing $\mathbf{f}$ would be $\mathcal{F}^{ns}$. In the following, we provide an example in which the number of input variables of the function $\mathbf{f}$ is increased in order to find a uniform sharing.

Consider the first-order direct sharing of multiplication of two variables $(A = f(X, Y) = XY)$ with three shares which does not satisfy Property 4. We call the extra variable Z which is introduced to $\mathbf{f}$ without influencing the output of the function a *virtual variable*. We refer to the shares of this variable as *virtual shares*. Together with the uniform sharing of Z, we re-define the sharing of f as follows:

$$A_1 = X_2Y_2 \oplus X_2Y_3 \oplus X_3Y_2 \oplus X_2Z_2 \oplus X_3Z_3 \oplus Y_2Z_2 \oplus Y_3Z_3$$

$$A_2 = X_3Y_3 \oplus X_1Y_3 \oplus X_3Y_1 \oplus X_3Z_3 \oplus X_1Z_1 \oplus Y_3Z_3 \oplus Y_1Z_1 \quad (3.18)$$

$$A_3 = X_1Y_1 \oplus X_1Y_2 \oplus X_2Y_1 \oplus X_1Z_1 \oplus X_2Z_2 \oplus Y_1Z_1 \oplus Y_2Z_2.$$

The sharing defined uses $3 \times 3 = 9$ elements including the virtual shares for one multiplication. On the other hand, the same function f has a uniform first-order TI with four shares as shown in Equation (3.15), hence with $2 \times 4 = 8$ elements. Therefore using the virtual variable as suggested above does not necessarily provide a sharing that uses less elements even though it can be advantageous depending on the implementation.

We can also introduce less than three virtual shares since they are unrelated to the *real* input of the function and do not need to be taken into account during the non-completeness check of the sharing. The previous multiplication can be shared using only one virtual share as:

$$A_1 = X_2Y_2 \oplus X_2Y_3 \oplus X_3Y_2 \oplus Z$$

$$A_2 = X_3Y_3 \oplus X_1Y_3 \oplus X_3Y_1 \oplus X_1Z \oplus Y_1Z \quad (3.19)$$

$$A_3 = X_1Y_1 \oplus X_1Y_2 \oplus X_2Y_1 \oplus X_1Z \oplus Y_1Z \oplus Z.$$

It is important to ensure that this virtual variable (or its shares) is unpredictable to an attacker similar to the input shares, i.e. virtual variable (and shares) has to be random.

This approach can be seen as introducing additional randomness to the shared function $\mathbf{f}$. Notice that if we consider the term Z and the term $X_1 Z \oplus Y_1 Z$ in Equation (3.19) as two different masks, this equation becomes equivalent to re-masking as in Equation (3.13) which uses 8 elements. Therefore, using virtual variables can be viewed as a clever way of re-masking that requires only 7 elements in this case. This improvement in the number of elements is very small when only one multiplication is considered. On the other hand, a cryptographic algorithm typically uses much more than one multiplication.

Varying the Number of Shares

Until now we only considered $s_{\text{in}} \leq s_{\text{out}}$. Here we will shortly illustrate in a first-order scenario that it is possible to have a uniform TI with $s_{\text{in}} > s_{\text{out}}$. Take the product $A = XY$, such that $s_{\text{in}} = 4$ and $s_{\text{out}} = 3$. The following sharing of the function satisfies all TI properties.

$$A_1 = (X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3) \oplus Y_4$$

$$A_2 = (X_1 \oplus X_3)(Y_1 \oplus Y_4) \oplus X_1 Y_3 \oplus X_4$$

$$A_3 = (X_2 \oplus X_4)(Y_1 \oplus Y_4) \oplus X_1 Y_2 \oplus X_4 \oplus Y_4.$$

Even though this approach does not decrease the number of elements used, it can be considered as a clever way of reducing the number of shares without losing a clock cycle that is inherited from the need of using registers before decreasing the number of shares as mention in Remark 3. Hence, this approach gives flexibility when several blocks are combined with each other.

Decomposition

Last but not least, we can also decompose nonlinear equations into lower degree nonlinear equations to achieve Property 4, to be able to use less shares or to be able to have more freedom in sharing. We will discuss several examples of this approach in Sections 5.1.1, 5.2.1 and 7.2.2.

3.4 TI and Affine Equivalence

In the previous sections, we discussed several methods to generate shared functions such that the sharing satisfies correctness and non-completeness properties. In Theorem 2, we provided a systematic way to generate the component functions of a sharing with which these properties are satisfied automatically. We also showed that generating a dth-order TI that also satisfies the uniformity property is not trivial however, there are several options to remedy this problem. This wide range of possibilities makes the search space too big. Hence, trying all the options for a given function is sometimes infeasible. On the other hand, we can use the knowledge of the affine equivalent classes (Definition 1) to relatively reduce the work since we mostly work with permutations in this thesis.

If a uniform TI for a permutation f is known, any of its affine equivalents $\tilde{f} = l_l \circ f \circ l_r$ can trivially be implemented uniformly in a cascaded manner. This observation reveals the interesting question of generating a uniform TI of $\tilde{f}$, given both $\tilde{f} = l_l \circ f \circ l_r$ and the uniform TI of f, without the need of cascaded operations. We show that the answer is positive when sharings are defined as in the proof of Theorem 2 with the following theorem. Note that we mostly restrict ourselves to this kind of sharings and deviate from it only when we can not go further with this approach or that there is a trivial alternative solution. Therefore the following theorem has a particular importance.

Theorem 6. *If we have a uniform d^{th}-order TI as described in the proof of Theorem 2 for a representative of an affine equivalence class (as in Definition 1), then we can derive a uniform d^{th}-order TI for all permutations from the same class.*

Proof. Let f be an n-bit permutation which has a uniform, non-complete and correct sharing $\mathbf{f}$ with s_{out} shares f_j. Denote the input vector of $\mathbf{f}$ by $\mathbf{X}$, and its s_{in} shares by X_i. Each f_j contains n shared coordinate functions depending on at most t of the input shares X_i, where t is the maximum degree of these functions, such that the non-completeness property is satisfied (Remark 1). Without loss of generality, we denote by $\bar{X}_j$ the vector $(X_{\alpha_1}, X_{\alpha_2}, \ldots, X_{\alpha_t})$ where $1 \le \alpha_i \le s_{\text{in}}$ which contains the t inputs of f_j.

We now construct a uniform, non-complete and correct sharing for any permutation $\tilde{f}$ which is affine equivalent to f. By Definition 1, there exist two n-bit affine permutations l_r and l_l s.t. $\tilde{f} = l_l \circ f \circ l_r$. In order to lighten the notation, we give the proof for the case that l_r and l_l are linear permutations. We define $\mathbf{l}_r, \mathbf{l}_l$ as the $ns_{\text{in}} \times ns_{\text{in}}$ and $ns_{\text{out}} \times ns_{\text{out}}$ permutations that apply l_r

and l_l respectively, to each of the shares separately:

$$\mathbf{l}_r(X_1, X_2, \ldots, X_{s_{\text{in}}}) = (l_r(X_1), l_r(X_2), \ldots, l_r(X_{s_{\text{in}}})),$$

$$\mathbf{l}_l(X_1, X_2, \ldots, X_{s_{\text{out}}}) = (l_l(X_1), l_l(X_2), \ldots, l_l(X_{s_{\text{out}}})).$$

Denote $A_i = l_r(X_i), 1 \leq i \leq s_{\text{in}}$ and without loss of generality define $\bar{A}_j$ as the vector $A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_t}$ containing only t shares where $1 \leq \alpha_i \leq s_{\text{in}}$ which we need to compute f_j. Consider $\mathbf{f}(\mathbf{l}_r(X_1, X_2, \ldots, X_{s_{\text{in}}})) = (f_1(\bar{A}_{\alpha_1}), f_2(\bar{A}_{\alpha_2}), \ldots, f_{s_{\text{out}}}(\bar{A}_{\alpha_{s_{\text{out}}}}))$. By slight abuse of notation we can write $\bar{A}_j = \mathbf{l}_r(\bar{X}_j)$ and see that the non-completeness of the f_j is preserved in $\mathbf{f} \circ \mathbf{l}_r$. Since $\mathbf{l}_r$ is a permutation, it preserves the uniformity of the input and since $\mathbf{f}$ is uniform so will be the composition $\mathbf{f} \circ \mathbf{l}_r$. The correctness follows from the fact that $\mathbf{f}$ is a correct sharing and that

$$A_1 \oplus A_2 \oplus \cdots \oplus A_{s_{\text{in}}} = l_r(X_1) \oplus l_r(X_2) \oplus \cdots \oplus l_r(X_{s_{\text{in}}}) = l_r(X_1 \oplus X_2 \oplus \ldots X_{s_{\text{in}}}) = l_r(X).$$

Consider now $\mathbf{l}_l(\mathbf{f}(\mathbf{l}_r(X))) = (l_l(f_1(\bar{A}_1)), l_l(f_2(\bar{A}_2)), \ldots, l_l(f_{s_{\text{out}}}(\bar{A}_{s_{\text{out}}})))$. Since $\mathbf{l}_l$ is a permutation, it preserves uniformity of the output and since $\mathbf{f}$ is uniform, the composition $\mathbf{l}_l \circ \mathbf{f}$ is uniform. The composition is non-complete since the f_j are non-complete and $\mathbf{l}_l$ does not combine different shares. Correctness follows from the fact that $\mathbf{f}$ is a correct sharing and hence

$$l_l(f_1(\bar{A}_1)) \oplus l_l(f_2(\bar{A}_2)) \oplus \cdots \oplus l_l(f_{s_{\text{out}}}(\bar{A}_{s_{\text{out}}}))$$
$$= l_l(f_1(\bar{A}_1) \oplus f_2(\bar{A}_2) \oplus \cdots \oplus f_{s_{\text{out}}}(\bar{A}_{s_{\text{out}}})) = l_l(f(l_r(X))).$$

$\square$

3.5   Conclusion

The aim of this chapter was to discuss all the theoretical aspects of TI. We started with the generic properties, correctness and uniformly distributed input shares, which any masking scheme needs to follow.

Afterwards we moved to the non-completeness property which distinguishes TI from other masking schemes. We showed that a dth-order TI can be generated with $d + 1$ input shares under certain conditions. However, it is not preferable due to the increase of number of output shares when complex functions are

considered and its limitation of the input behavior. Moreover, such a sharing requires attention in many stages during an implementation such as reducing the number of shares which is not preferable in a TI. The goal of TI is to produce a dth-order secure sharing that can be applied to any function with minimum effort and resource requirements. With that motivation we provided a systematic way to generate a dth-order TI of a degree t function with $s_{\text{in}} \geq t \times d + 1$ input shares and $s_{\text{out}} \geq \binom{s_{\text{in}}}{t}$ output shares. We proved its security and discussed reducing the number of shares back to s_{in} following such a sharing.

We argued that when parallel or cascaded operations are considered such as several layers and rounds following each other in a symmetric-key algorithm, the uniformity of a shared function becomes important since its output will be the input of the following function. Even if the first three properties of a TI are satisfied, the uniformity of the shared function is not directly assured thus, each sharing should be individually checked. For the situations where the test reveals that the shared function is nonuniform, we suggested several options to remedy this problem providing flexibility during the implementation process. We believe each option has its advantages and disadvantages. Hence, we avoid to argue one is better than the other. We note that the area-randomness trade-off must be examined before making a decision.

Finally, we showed that if we have a TI of a function, we can generate a TI for all of its affine equivalents which we will especially use in Chapter 5.

4

Threshold Implementations of KATAN-32

In the previous chapter, we discussed the theoretical aspects of applying TI to an arbitrary function. Hereon, we examine TI's behavior in practice, especially on hardware since TI claims security even in the presence of glitches. The goal of this work is twofolds. Firstly, we examine the effects of TI without any optimization on area and timing. We choose the block cipher KATAN [42] due to the simple structure of its nonlinear block with only a few AND gates as described in Section 4.1. We implement an unprotected version in addition to first-, second- and third-order TIs which are provided in Section 4.2. We provide the area requirements of these implementations using the Faraday Standard Cell Library FSA0A_C_Generic_Core which is based on UMC 0.18μm GenericII Logic Process with 1.8V voltage.

The second goal is to analyze the claimed security against a strong adversary. We provide an analysis of our second-order TI in Section 4.3. The analysis is performed on a Field-Programmable Gate Array (FPGA) using the leakage detection test as explained in Section 2.3.6. Our implementations with different orders follow the same strategy hence, showing the security on second-order implementation implies first- and third-order TI's security.

With this chapter, we complete the answer to Question 1, namely extension

of TI to counteract higher-order DPA. This work is published in [17]. We provide the analysis section which is performed and written by B. Gierlichs for completeness. We start this chapter with a brief description of the algorithm.

4.1   Introduction to KATAN

KATAN is a family of block ciphers that is designed to be efficient in hardware. The family has three variants with 32-, 48- or 64-bit state size. All these variants use an 80-bit key, hence provide the same security level. A plaintext block, of the same size as the state, is loaded into the state to start an encryption. After 254 rounds, the content of the state is taken as the ciphertext. The round operation is very similar for all variants and has only a few AND and XOR gates. We implement the smallest variant of KATAN with 32-bit state size and focus on encryption.
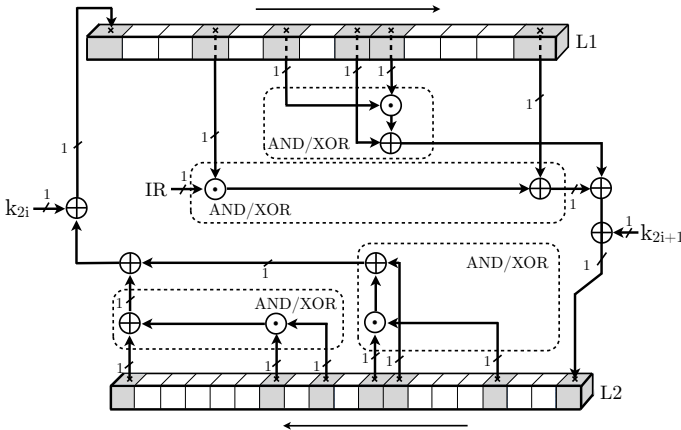
Figure 4.1: Schematic of one round of KATAN-32

The schematic of one round KATAN-32 is provided in Figure 4.1 where each block represents one bit. 32-bit plaintext is divided into two chunks of 13 and 19 bits and written to the registers $L1$ and $L2$ respectively. In every round, several bits are used to update the first bits of the registers together with a one bit shift to the right for $L1$ and to the left for $L2$. The bit depicted by IR is the last bit of a round counter that decides irregularly if the fourth bit of $L1$ is used for the round update or not. k_{2i} and k_{2i+1} are the $2i^{\text{th}}$ and $(2i+1)^{\text{st}}$ bits of the 80-bit key for rounds $i \leq 40$. For the rest of the rounds they are generated from the original key by an LFSR. For more details, we refer to [42]

4.2   Implementations

The threshold implementations are based on the unprotected implementation, thus we first explain this plain version.

4.2.1   Unprotected Implementation

The module responsible from the KATAN encryption takes 32-bit plaintext and 80-bit key as input. We use 32-bit register divided into $L1$ and $L2$ in order to store the intermediate state as defined in 4.1. We output the 32-bit content of this register after 254 clock cycles. We update two bits of this state per cycle without any unrolling. We define the nonlinear function as four instantiations of the same one AND and one XOR gate block each of which can be described with the function $A = f(X, Y, Z) = X \oplus YZ$. The particular choice for this nonlinear block is given in Section 4.2.2.

The leading area cost is the key-schedule LFSR with 444 GE which is mainly caused by the 80-bit register. The state register and the nonlinear function cost 170 GE and 54 GE respectively. The implementation is extremely lightweight with only 1002 GE.

4.2.2   Threshold Implementations

In all the TIs, we assume that the plaintext shares, which are generated from the unshared plaintext by a uniform masking, are provided from an outside source. For simplicity, we use an unshared key and key schedule. The key XOR is performed only on the first shares of the state. We focus on the sharing of the state and its nonlinear round function. Even though there exists DPA techniques specifically targeting the unmasked key schedule, we do not consider such analysis since the main goal of this work is to show the security of the TI construction. The key schedule, being linear, can be implemented easily using several shares and the reflected area cost can be calculated trivially.

We start by finding first-, second- and third-order TIs for the nonlinear block of KATAN since it is the most challenging part. We then decide the number of state shares.

First-Order Threshold Implementation

It has been described in the early works of first-order TI [75] that a uniform first-order three-share TI of one AND gate does not exist even with CTs. On the other hand, there exists a uniform TI of the function $A = f(X, Y, Z) = X \oplus YZ$ with three shares. Therefore, we always group an AND gate with an XOR gate in our implementations for consistency. For all the AND/XOR blocks except the one that receives IR, we use the direct sharing provided in Equation (4.1). This sharing for the quadratic function follows Theorem 2 with $s_{\text{in}} = t \times d + 1 = 3$ input and $s_{\text{out}} = \binom{s_{\text{in}}}{t} = 3$ output shares.

$$
\begin{aligned}
A_1 &= X_2 \oplus (Y_2 Z_2 \oplus Y_2 Z_3 \oplus Y_3 Z_2) \\
A_2 &= X_3 \oplus (Y_3 Z_3 \oplus Y_3 Z_1 \oplus Y_1 Z_3) \\
A_3 &= X_1 \oplus (Y_1 Z_1 \oplus Y_1 Z_2 \oplus Y_2 Z_1)
\end{aligned}
\tag{4.1}
$$

For the AND/XOR block that receives IR we use the sharing

$$
A_i = X_i + IR \times Y_i \text{ where } i \leq s_{\text{in}}
\tag{4.2}
$$

because we do not share the round counter (and hence IR).

The number of state shares is chosen to be three following the sharing of the nonlinear function. Hence, the state is defined as three 32-bit registers. Our first-order KATAN TI module takes three shares simultaneously that are uniformly distributed following Property 1. We write the output of the nonlinear function to the shared state right after this operation fulfilling all the TI requirements.

The timing of this implementation is the same as the unprotected implementation. The main differences between the area requirements of these two implementations are the costs of the state register and the nonlinear blocks. The overall cost of this three-share implementation is 1720 GE.

Second-Order Threshold Implementation

Similar to the previous version, the TIs of the nonlinear AND/XOR gates are chosen to be the direct sharing with $s_{\text{in}} = 5$ and $s_{\text{out}} = 10$ shares following Theorem 2, which is provided in Equation (3.9). After the nonlinear operation, the number of output shares are decreased back to five as described

in Equation (3.16) in order to satisfy all four TI properties. We use s_{in} shares for most of the state and use Equation (4.2) if IR is an input to the AND/XOR block. The shared output XOR of Equation (4.2) to Equation (3.9) is performed only on the first five shares of the latter equation.
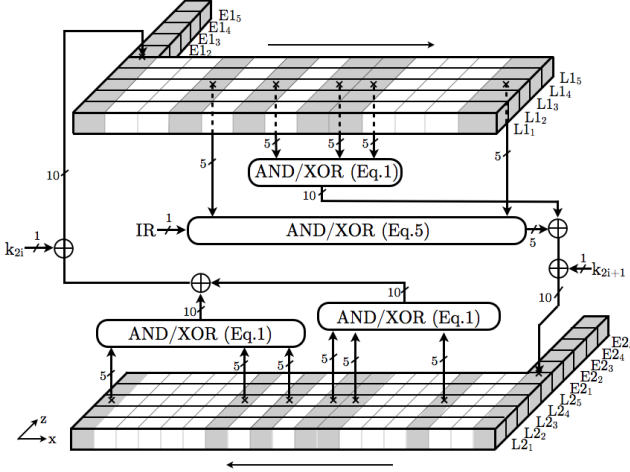
Figure 4.2: Schematic of second-order TI of one round of KATAN-32

One round of this implementation is depicted in Figure 4.2 where the z coordinate refers to s_{in} different shares of the state. Note that each shared register corresponding to the updated bits uses an additional 5-bit register to store the extra $s_{\text{out}} - s_{\text{in}}$ bits. In the clock cycle following a nonlinear operation, the s_{out} shares in the first bits of the $L1$ and $L2$ registers are first reduced to s_{in} shares, then written as the second bits. This preserves the non-completeness property as pointed out in Remark 3. This implementation which also has the same timing as the previous versions costs 2556 GE.

Third-Order Threshold Implementation

This implementation which uses a nonlinear block with $s_{\text{in}} = 7$ input and $s_{\text{out}} = 21$ output shares given in Equation (4.3) resembles to the second-order TI. The state is composed of seven 32-bit registers together with two additional 14-bit registers to store the extra bits (shares) of the nonlinear operations. The seven output shares of Equation (4.2) are XORed only to the the first seven

output shares of Equation (4.3).

$$A_1 = X_2 + Y_2 Z_2 + Y_1 Z_2 + Y_2 Z_1 \quad A_8 = Y_2 Z_4 + Y_4 Z_2 \qquad A_{15} = Y_3 Z_7 + Y_7 Z_3$$

$$A_2 = X_3 + Y_3 Z_3 + Y_1 Z_3 + Y_3 Z_1 \quad A_9 = Y_2 Z_5 + Y_5 Z_2 \qquad A_{16} = Y_4 Z_5 + Y_5 Z_4$$

$$A_3 = X_4 + Y_4 Z_4 + Y_1 Z_4 + Y_4 Z_1 \quad A_{10} = Y_2 Z_6 + Y_6 Z_2 \qquad A_{17} = Y_4 Z_6 + Y_6 Z_4$$

$$A_4 = X_5 + Y_5 Z_5 + Y_1 Z_5 + Y_5 Z_1 \quad A_{11} = X_7 + Y_7 Z_7 + Y_2 Z_7 + Y_7 Z_2 \quad A_{18} = Y_4 Z_7 + Y_7 Z_4 \quad (4.3)$$

$$A_5 = X_6 + Y_6 Z_6 + Y_1 Z_6 + Y_6 Z_1 \quad A_{12} = Y_3 Z_4 + Y_4 Z_3 \qquad A_{19} = Y_5 Z_6 + Y_6 Z_5$$

$$A_6 = X_1 + Y_1 Z_1 + Y_1 Z_7 + Y_7 Z_1 \quad A_{13} = Y_3 Z_5 + Y_5 Z_3 \qquad A_{20} = Y_5 Z_7 + Y_7 Z_5$$

$$A_7 = Y_2 Z_3 + Y_3 Z_2 \qquad\qquad A_{14} = Y_3 Z_6 + Y_6 Z_3 \qquad A_{21} = Y_6 Z_7 + Y_7 Z_6$$

In Table 4.1, we provide the area requirements of all four implementations. The results reflect a linear increase of area parallel to the linear increase of the number of input shares of the nonlinear function. This is due to the number of registers being the dominating cost. The key register is included in the gate count of the key schedule together with the LFSR update.

Table 4.1: Synthesis results for plain and TI of KATAN-32

Design	Unprotected	First-order	Second-order	Third-order
State Ar.	170	510	900	1330
Key Ar.	444	444	444	444
Round Func.	54	135	341	760
Control	64	64	64	64
Other	270	567	807	941
Total	1002	1720	2556	3539
Cycles	254	254	254	254

4.3   Power Analysis

We implement our second-order TI of KATAN-32 on a SASEBO-G board [3] using Xilinx ISE version 10.1 to evaluate its leakage characteristics in practice. The board features two Xilinx Virtex-II Pro FPGA devices: we implement the second-order TI of KATAN-32 in the crypto FPGA (xc2vp7) while the control FPGA (xc2vp30) handles I/O with the measurement computer and other equipment including the random number generation. We use the "keep hierarchy" constraint when we generate the bitstream for the crypto FPGA to

prevent the tools from optimizing over module boundaries. This is to prevent the tools from merging component functions and to reduce the chance for crosstalk. The key is hard-coded in the KATAN-32 implementation. The pseudo-random number generator (PRNG) on the control FPGA is implemented as AES-128 in CTR mode. To start an encryption, we share the plaintext in five shares using random numbers from the PRNG and send the shares to the KATAN-32 implementation. When the PRNG is turned off, it outputs zeros.

We measure the power consumption of the crypto FPGA during the first 12 rounds of KATAN-32 encryption as the voltage drop over a 1Ω resistor in the FPGA core GND line. The output of the passive probe is sampled with a Tektronix DPO 7254C digital oscilloscope at 1GS/s sampling rate and 1mV/div amplitude resolution. We provide the FPGA with a stable 3 MHz clock signal and use synchronized clocks to obtain high-quality measurements.

The main goal of our evaluation is not to demonstrate that the implementation resists state-of-the-art attacks that exploit the first or second statistical moment of the leakage distributions, but beyond that to demonstrate that there is no evidence of leakage in these moments of the leakage distributions, exploitable by state-of-the-art attacks or not. Obviously achieving this goal is much more demanding than resistance to known attacks, but it directly corresponds to our claims regarding provable security. We narrow the evaluation to univariate attacks because our implementation processes all component functions in parallel. For our purpose we use the *non-specific* t-test based fixed versus random leakage detection methodology of [35, 52], which is briefly introduced in Section 2.3.6.

For all tests we obtain two sets of measurements. For the first set, we fix the plaintext to some chosen value. We denote this set $\mathcal{S}_0$. For the second set, the plaintexts are uniformly distributed and random. We denote this set $\mathcal{S}_{random}$. We obtain the measurements for both sets interleaved and in a random order, i.e. before each measurement we flip a coin, to avoid any deterministic or time-dependent external and internal influences on the test result.

While this evaluation methodology relieves us from choosing certain parameters such as targeted intermediate value, power model and distinguisher, it does not resolve all such issues. As in any evaluation, the tests are limited to the number of measurements at hand and one has to choose a threshold to decide if an observed difference is statistically significant or not. Nevertheless, as we demonstrate below this type of evaluation is very data-efficient, i.e. a small number of measurements is required to provide evidence of leakage, and a decision threshold can be motivated with some basic experiments.

To calibrate our threshold value we apply the test methodology to two groups of 10 000 measurements each for which we know that the null hypothesis is true.

For the first group of measurements we switch off the PRNG and use the same fixed plaintext for both sets, i.e. all measurements in both sets are samples from the same population and the only cause of variance is noise. We compute the t statistic, record its greatest absolute value and repeat the experiment 100 times on a random split of the measurements in this group. The highest absolute t value we observed was 4.7944. For the second group we switch on the PRNG and use random plaintexts for both sets, i.e. the measurements in both sets are samples from distributions with the same mean and high variance. We repeat the analysis and the highest absolute t value we observed was 4.8608. Based on these results and the recommendation in [35] we select the significance threshold ± 4.5. For large sample sizes, observing a single t value greater/smaller than ± 4.5 roughly corresponds to a 99.999% probability of the null hypothesis being false.

To confirm that our setup works correctly and to get some reference values we first evaluate the implementation with the PRNG switched off. Figure 4.3 shows the t values of fixed versus random tests with two different fixed plaintexts (left and right) and for the first, second and third statistical moment of the distributions (for the higher-order moments we pre-process the traces to expose the desired standardized moment before we apply the t-test, e.g. for the second moment we center and then square the traces). Horizontal lines mark the ± 4.5 thresholds.

The plots clearly show that there is sufficient evidence of leakage in all cases, as there are multiple and systematic crossings of the thresholds. Comparing the plots on the left hand side with the plots on the right hand side, we see that the "shape" of the t curve depends on the fixed plaintext value. This is no longer true when we switch on the PRNG, because all shares of the input are random. We used 1 000 measurements (500 for fixed and 500 for random plaintext) to generate these plots, but less than 100 measurements are required to see evidence of leakage in the 1st statistical moment.

Now we switch on the PRNG and repeat the evaluation with a randomly chosen fixed plaintext using 300 million measurements (150M for fixed, 150M for random, done in a temperature controlled environment). Figure 4.4 (top left and right) shows plots of the t values for the first and second moment. As expected there is not sufficient evidence of leakage. But as mentioned earlier, one may always wonder if the number of measurements at hand is sufficient. For completeness, we also provide evaluation results of the third and fifth moment. The third moment is the smallest moment for which our implementation does not provide provable security in the combinational logic (Property 3) and the fifth moment is the smallest moment for which our implementation does not provide provable security in the memory elements (the state is shared in at least five shares). Therefore we may be able to detect leakage in these moments.
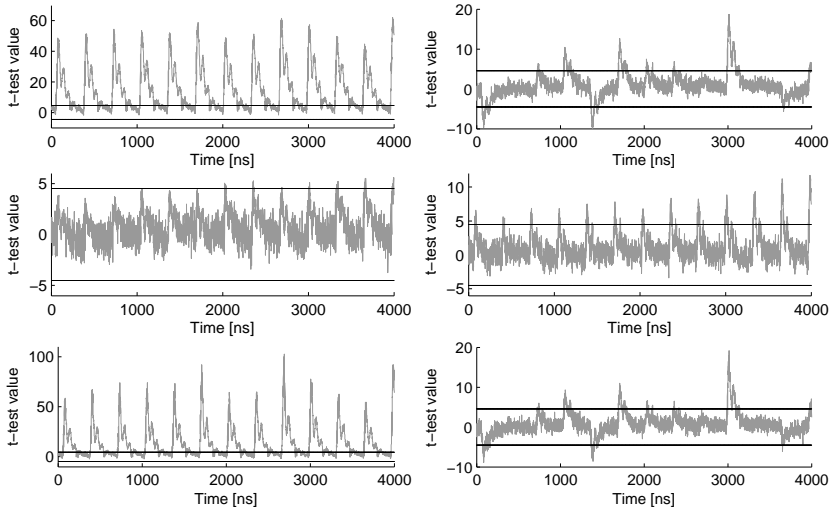
Figure 4.3: Fixed versus random t-test evaluation results with PRNG switched off; left: for fixed plaintext 0x00000000, right: for a randomly chosen fixed plaintext; from top to bottom: $1^{st}, 2^{nd}$ and 3rd-order statistical moment; 1 000 measurements

Figure 4.4 (bottom left and right) shows plots of the t values.
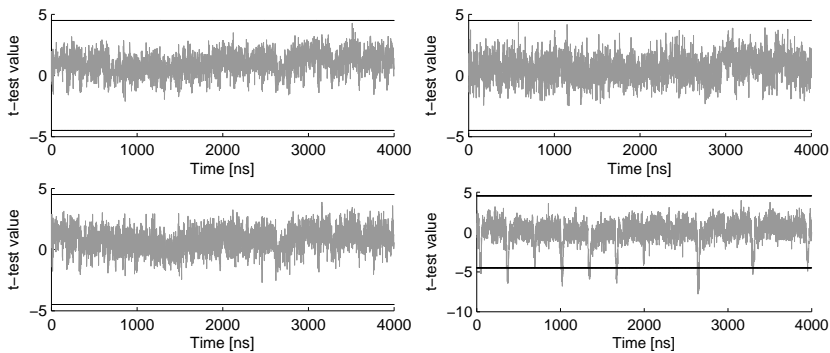
Figure 4.4: Fixed versus random t-test evaluation results with PRNG switched on for a randomly chosen fixed plaintext; from top left, top right, to bottom right: first, second, third and fifth statistical moment; 300 million measurements

While there is not sufficient evidence of leakage also in the third moment, we can see multiple and systematic crossings of the threshold in the fifth moment. This result suggests that we use enough measurements, and that we should be

able to detect leakage in the lower-order moments, if there was any. Together, the results support our claim regarding provable second-order DPA resistance.

One may wonder why we do not detect leakage in the third moment. Several explanations are possible. We leave the careful investigation as future work.

4.4   Conclusion

Due to the strength of HO-DPA attacks, it is desirable to provide a countermeasure that counteracts these attacks at any order on any device. Therefore, we worked on an answer to Question 1 of which the theoretical solution is provided in Chapter 3. Prior works in the area show that it is important to perform analysis to examine the claimed security on practice. Thus, we implemented the block cipher KATAN-32 with second-order TI which is then tested with first-, second-, third- and fifth-order DPA. Our examination using 300 million traces aligns with the claimed security since we observe no leakage with first- and second-order DPA. To observe the increase in number of shares which affects the area requirement, we additionally provided first- and third-order TI of KATAN. For a fair comparison, we also implemented an unprotected version then compared all these increasing order implementations. We observed that the area cost of the nonlinear block is low compared to the cost of the state register. Hence, the number of shares of the state decides the increase of area with higher-order implementations of KATAN. We acknowledge that the nonlinear block of KATAN is very simple. Hence, we will move to more complex S-boxes in the following chapters.

5

Threshold Implementations of Small S-boxes

We observed in the previous chapters that TI of the nonlinear blocks have a particular importance since the number of shares used in the system are chosen accordingly. S-boxes, as defined in Section 2.2, are nonlinear blocks defined on a finite field that bring confusion to symmetric key algorithms. Typically, these S-boxes are more complex than the the nonlinear AND/XOR gate block discussed in Chapter 4. The input and output size of these S-boxes can vary as in DES block cipher [43] which uses a 6×4 S-box. However, most of the S-boxes used in cryptographic algorithms are permutations of at least size three, i.e. $n \times n$ S-boxes (n-bit permutations with good cryptographic properties). We refer to S-boxes where $n < 8$ as small S-boxes. In this chapter, we aim to provide uniform TIs for a wide range of small S-boxes and explore their area requirements. We use the affine equivalence relations described in Section 2.2 for a systematic work.

Considering that all 2-bit permutations are affine and that TI of affine operations are trivial (Section 3.3), we do not discuss them any further. 4-bit permutations inherit some properties from 3-bit permutations. That is why they are investigated together in Section 5.1. 5- and 6-bit permutations are not used as widely as 4-bit permutations, however some of them provide good cryptographic properties which are discussed in Section 2.2.3. TIs of these S-boxes are examined in Section 5.2.

For both 3- and 4-bit permutations, and 5- and 6-bit permutations, we first discuss how to find uniform TIs in Sections 5.1.1 and 5.2.1 respectively. We mainly provide first-order TI of these permutations as defined in Theorem 2. Therefore the number of input shares are defined as $s_{\text{in}} \geq t + 1$ where t is the algebraic degree of the permutation. Moreover, we fix $s_{\text{out}} = s_{\text{in}} = s$ during this chapter.

In the second part of each section, we provide a fair comparison and prediction of the area cost using $45nm$ NanGate standard cell library [74]. We used *compile ultra* command in Synopsis to optimize each component function (resp. unshared function). Then we combined these to get the cost of TI (resp. permutation). The results are shown in Sections 5.1.2 and 5.2.2.

In Section 5.1.3, we provide a possible extension to higher-order TI of quadratic 3- and 4-bit permutations and leave the rest to the reader.

The work presented in this chapter is published in [22, 23]. Moreover, we used this information during the designs of FIDES [15] and PRIMATEs [4] authenticated encryption algorithms. Algorithm 1 is mainly the work of N. Tokareva and V.Vitkup. The toolbox [20] produced by S. Nikova and V. Nikov can be used to generate the work presented in Section 5.1.

5.1 3- and 4-bit Permutations

Before any further discussion on TI of these permutations, we hereby provide a relation between 3-bit permutations and some classes of 4-bit permutations.

Lemma 7. *There exists a transformation which expands $\mathcal{Q}_1^3$, $\mathcal{Q}_2^3$ and $\mathcal{Q}_3^3$ (in Table A.1) into $\mathcal{Q}_4^4$, $\mathcal{Q}_{12}^4$ and $\mathcal{Q}_{300}^4$ (in Table A.2-A.6) correspondingly.*

Proof. Starting from a 3-bit permutation f and adding a new variable we can obtain a 4-bit permutation $\tilde{f}$. Namely, the transformation is defined as follows: let $f(Y, Z, W) = (A, B, C)$ and define $\tilde{f}(X, Y, Z, W) = (A, B, C, X)$. It is easy to check that this transformation maps the former three classes into the latter three classes.

$\square$

The relation from Lemma 7 explains why having a uniform TI for a permutation in some class in $\mathcal{F}_2^3$ leads to having a uniform TI for the corresponding S-box in the corresponding class in $\mathcal{F}_2^4$ and vice versa using the same amount of correction terms. Therefore, if we cannot provide uniform TI for any permutation from

a given class then none of the permutations in the corresponding class can be implemented uniformly with TI using the same set of CT.

5.1.1 Finding Uniform Threshold Implementations

We can always find a first-order TI of a permutation with $s_{\mathrm{in}} \geq t + 1$ shares as defined in Theorem 2, i.e. a quadratic permutation needs at least three shares and a cubic permutation needs at least four shares. The main question is finding a *uniform* TI. In the following, we apply several ideas from Section 3.3.2 to find uniform TIs for all 3- and 4-bit affine equivalence classes.

Direct Sharing

Recall that the uniformity of a direct sharing produced as given in Section 3.2.1 is not guaranteed. It has to be verified separately. On the other hand, it is enough to find a uniform direct sharing for one permutation within the class to judge if permutations that belong to the same class have a TI by Theorem 6. Therefore, we run a search algorithm that goes through all permutations within a class to find a permutation that has a direct uniform sharing. With this search, we find three-share TIs of several permutations in $\mathcal{Q}_1^3$, but none in $\mathcal{Q}_2^3$ and $\mathcal{Q}_3^3$. We also find three-share TIs of many permutations in $\mathcal{Q}_4^4$, $\mathcal{Q}_{294}^4$ and $\mathcal{Q}_{299}^4$, but none in $\mathcal{Q}_{12}^4$, $\mathcal{Q}_{293}^4$ and $\mathcal{Q}_{300}^4$. So, unfortunately half of the quadratic permutations lack a uniform TI when shared directly with three shares. Note that direct sharing of a cubic S-box using three shares is unrealizable due to the $s \geq t + 1$ limitation requiring at least four shares.

We can increase the number of shares to find a uniform sharing for quadratic or cubic permutations. When we use four shares, we observe that all quadratic classes have at least one permutation that has a uniform TI with direct sharing. We also find uniform TIs for the permutations in cubic classes $\mathcal{C}_1^4$, $\mathcal{C}_3^4$, $\mathcal{C}_{13}^4$ and $\mathcal{C}_{301}^4$ from $\mathcal{S}_{16} \setminus \mathcal{A}_{16}$ using direct sharing with four shares.

On the other hand, a uniform TI of at least one permutation from all classes can be found if all the shared linear and quadratic terms of component functions are distributed as in Equation (5.1) which is given for the function $A = f(X, Y, Z) = 1 \oplus X \oplus YZ$. We call such a sharing as a *first-order direct sharing with five shares*. Being able to find uniform TIs for all classes is a big improvement compared to the situation with four shares.

$$A_1 = 1 \oplus X_2 \oplus Y_2 Z_2 \oplus Y_2 Z_3 \oplus Y_3 Z_2 \oplus Y_2 Z_4 \oplus Y_4 Z_2$$

$$A_2 = X_3 \oplus Y_3 Z_3 \oplus Y_3 Z_4 \oplus Y_4 Z_3 \oplus Y_3 Z_5 \oplus Y_5 Z_3$$

$$A_3 = X_4 \oplus Y_4 Z_4 \oplus Y_4 Z_5 \oplus Y_5 Z_4 \oplus Y_4 Z_1 \oplus Y_1 Z_4 \tag{5.1}$$

$$A_4 = X_5 \oplus Y_5 Z_5 \oplus Y_5 Z_1 \oplus Y_1 Z_5 \oplus Y_5 Z_2 \oplus Y_2 Z_5$$

$$A_5 = X_1 \oplus Y_1 Z_1 \oplus Y_1 Z_2 \oplus Y_2 Z_1 \oplus Y_1 Z_3 \oplus Y_3 Z_1.$$

Sharing Using Correction Terms

As mentioned in Section 3.3.2, the number of correction terms defined for 3-bit permutations is significantly large and when we consider 4-bit permutations, trying all possible CT exhaustively becomes infeasible. Here we describe an algorithm which can provide a negative result in case no uniform TI with CT exists, with a complexity less than the exhaustive search.

Consider an s-share TI $\mathbf{f}$ of an n-bit permutation f (if it exists). The uniformity property implies that the vectorial Boolean function $\mathbf{f} : \mathcal{F}_2^{ns} \to \mathcal{F}_2^{ns}, \mathbf{f} = (f_1, ..., f_s)$ is a balanced function. Recall some properties of the balanced vectorial functions [30].

Lemma 8. *Let $f = (f^1, \ldots, f^n)$ be a vectorial Boolean function from $\mathcal{F}_2^n$ to $\mathcal{F}_2^n$. f is a bijection if and only if for any k ($1 \leqslant k \leqslant n$) and for any tuple of indices $i_1, \ldots, i_k$ ($1 \leqslant i_1 \leqslant \ldots \leqslant i_k \leqslant n$) the vectorial Boolean function $(f^{i_1}, ..., f^{i_k})$ is balanced.*

Let $f = (f^1, \ldots, f^n)$ be a vectorial Boolean function such that $f : \mathcal{F}_2^n \to \mathcal{F}_2^n$ and $\mathbf{f} = (f_1^1, \ldots, f_1^n, \ldots, f_s^1, \ldots, f_s^n)$ be the direct sharing of f with s shares $f_i = (f_i^1, \ldots, f_i^n)$. We say that the function $\mathbf{c_f} = (c_1^1, \ldots, c_1^n, \ldots, c_s^1, \ldots, c_s^n)$, where $c_j^i : \mathcal{F}_2^{ns} \to \mathcal{F}_2$ are CT, is a *correction function* for $\mathbf{f}$, if the function $\mathbf{f} \oplus \mathbf{c_f}$ satisfies all the properties of a TI.

Let $k \in \{1, \ldots, ns\}$ and let $(i_1 j_1, ..., i_k j_k)$ be a k-tuple from the set $\{11, \ldots, n1, \ldots, 1s, \ldots, ns\}$. Denote the set

$$C_{i_1 j_1, \ldots, i_k j_k}^k = \{\mathbf{c_f} \mid (f_{j_1}^{i_1} \oplus c_{j_1}^{i_1}, \ldots, f_{j_k}^{i_k} \oplus c_{j_k}^{i_k}) \text{ is a balanced function from } \mathcal{F}_2^{ns} \text{ to } \mathcal{F}_2^k\}.$$

Then consider the set

$$\mathbf{C} = \bigcap_k \bigcap_{i_1 j_1, \ldots, i_k j_k} C_{i_1 j_1, \ldots, i_k j_k}^k.$$

Theorem 7. *The function $\mathbf{f} \oplus \mathbf{c_f}$ is a bijection if and only if $\mathbf{c_f} \in \mathbf{C}$.*

Proof. (Sufficient condition) By the definition of $\mathbf{C}$, for any function $\mathbf{c_f} \in \mathbf{C}$, the function $\mathbf{f} \oplus \mathbf{c_f}$ is such that for any $k \in \{1, \ldots, ns\}$ and for any tuple $(i_1 j_1, ..., i_k j_k)$, from $\{11, \ldots, n1, \ldots, 1s, \ldots, ns\}$ the vectorial sub-function $((f_{j_1}^{i_1} \oplus c_{j_1}^{i_1}, \ldots, f_{j_k}^{i_k} \oplus c_{j_k}^{i_k})$ is balanced. Hence, it follows from the lemma that the function $\mathbf{f} \oplus \mathbf{c_f} = (f_1^1 \oplus c_1^1, \ldots, f_1^n \oplus c_1^n, \ldots, f_s^1 \oplus c_s^1, \ldots, f_s^n \oplus c_s^n)$ is a bijection.

(Necessary condition) Let the function $\mathbf{f} \oplus \mathbf{c_f}$ be a bijection. Then for any $k \in \{1, \ldots, ns\}$, for any tuple $(i_1 j_1, ..., i_k j_k)$ from the set $\{11, \ldots, n1, \ldots, 1s, \ldots, ns\}$, the vectorial function $((f_{j_1}^{i_1} \oplus c_{j_1}^{i_1}, \ldots, f_{j_k}^{i_k} \oplus c_{j_k}^{i_k})$ is balanced. Hence, the function $\mathbf{c_f}$ belongs to the set $C_{i_1 j_1, \ldots, i_k j_k}^k$ by construction. Therefore, due to the arbitrariness of k and $(i_1 j_1, ..., i_k j_k)$, the function $\mathbf{c_f}$ belongs to the set $\mathbf{C} = \bigcap_k \bigcap_{i_1 j_1, \ldots, i_k j_k} C_{i_1 j_1, \ldots, i_k j_k}^k$.

$\square$

This theorem allows us to use Algorithm 1 to search for a TI for the permutation $f = (f^1, \ldots, f^n)$ s.t. $F : \mathcal{F}_2^n \to \mathcal{F}_2^n$.

Input: Direct sharing $\mathbf{f} = (f_1^1, \ldots, f_s^1, \ldots, f_1^n, \ldots, f_s^n)$ of f with s shares
$\quad\quad f_i = (f_i^1, \ldots, f_i^n)$ and the sets J_k containing all k-tuples $(i_1 j_1, ..., i_k j_k)$, from $\{11, \ldots, n1, \ldots, 1s, \ldots, ns\}$ for $1 \le k \le ns$.
Output: The set $\mathbf{C}$ s.t. for each function $\mathbf{c_f} \in \mathbf{C}$ the sharing
$\quad\quad f_1' = (f_1^1 \oplus c_1^1, \ldots, f_1^n \oplus c_1^n), \ldots, f_s' = (f_s^1 \oplus c_s^1, \ldots, f_s^n \oplus c_s^n)$ is uniform.
for $k = 1$ **to** ns **do**
$\quad$ **while** $J_k \ne \emptyset$ **do**
$\quad\quad$ Choose a tuple of indices $(i_1 j_1, ..., i_k j_k) \in J_k$.
$\quad\quad$ Assign $J_k := J_k \backslash (i_1 j_1, ..., i_k j_k)$.
$\quad\quad$ Construct the set
$\quad\quad C_{i_1 j_1, \ldots, i_k j_k}^k = \{\mathbf{c_f} \mid (f_{j_1}^{i_1} \oplus c_{j_1}^{i_1}, \ldots, f_{j_k}^{i_k} \oplus c_{j_k}^{i_k})$ is balanced function$\}$.
$\quad\quad \mathbf{C} := \bigcap_k \bigcap_{i_1 j_1, \ldots, i_k j_k} C_{i_1 j_1, \ldots, i_k j_k}^k$.
$\quad\quad$ **if** $\mathbf{C} \ne \emptyset$ **then**
$\quad\quad\quad \mid$ break;
$\quad\quad$ **end**
$\quad$ **end**
$\quad$ **if** $\mathbf{C} = \emptyset$ **then**
$\quad\quad \mid$ break;
$\quad$ **end**
end

Algorithm 1: Algorithm searching for a uniform TI

With this algorithm, we can conclude that there does not exist a uniform TI with correction terms if $\mathbf{C} = \emptyset$, which is likely to happen faster than the exhaustive search. Observe also that for any k, if the set $C^k_{i_1 j_1, \ldots, i_k j_k} = \emptyset$ then $\mathbf{C} = \emptyset$. Moreover, since the goal is to satisfy $\mathbf{C} \neq \emptyset$ for $k = ns$, we can start the *for* loop from any k s.t. $1 \leq k \leq ns$. If for any k, we have $\mathbf{C} \neq \emptyset$ at the end of the *while* loop, it implies that $\mathbf{C} \neq \emptyset$ for all $k' \leq k$.

We consider the permutation $f = (XY \oplus YZ \oplus XZ, X \oplus Y \oplus XY \oplus YZ, X \oplus Z \oplus YZ)$ from the class $\mathcal{Q}^3_3$ and apply the above algorithm with initialization of $k = 4$. We choose the tuple of indices as $(11, 12, 13, 21) \in J_k$ and construct the set $C_1 = C^4_{11,12,13,21}$ using linear and quadratic CTs. We obtain that the set C_1 is empty. Therefore, the set C from the theorem is empty, hence, there is no uniform sharing for the given permutation. The algorithm terminated after a computation with complexity 2^{35}. This proves the following:

Corollary 1. *There does not exist a quadratic uniform sharing with three shares for permutations from $\mathcal{Q}^3_3$.*

Recall that by Lemma 7, the class $\mathcal{Q}^3_3$ corresponds to the class $\mathcal{Q}^4_{300}$. Moreover, if there is no uniform sharing for 3-bit permutations from a class, then there does not exist a uniform sharing for 4-bit permutations from the corresponding class with the same amount of CT.

Corollary 2. *There does not exist a quadratic uniform sharing with three shares for permutations from $\mathcal{Q}^4_{300}$.*

Recall that, we could not find a uniform sharing for many classes with direct sharing. However, sharing linear and quadratic terms as in Equation (3.17) by using CT, we found uniform three-share TIs for permutations from the classes $\mathcal{Q}^3_2$, $\mathcal{Q}^4_{12}$, $\mathcal{Q}^4_{293}$.

So all 3- and 4-bit quadratic classes except $\mathcal{Q}^3_3$ and $\mathcal{Q}^4_{300}$ have a uniform TI with three shares.

Sharing Using Decomposition

We will show in Section 5.1.2 that the area requirements of an implementation of a cryptographic algorithm increase with the number of shares. This is only natural since more shares implies more AND and XOR gates even for small circuits. Therefore, it is desirable to keep the number of shares as low as possible. To this purpose, we can decompose a permutation into other, possible lower degree, permutations that have uniform TIs. Examples of such decompositions have been presented for NOEKEON [76, 77] and PRESENT [83] S-boxes in earlier

works. These three realizations decompose the S-box with algebraic degree three into two permutations with algebraic degree two. For the PRESENT S-box, decompositions $sbox(X) = f(g(X))$ with $g(0) = 0$ have been found [83] where $f(X)$ and $g(X)$ are quadratic permutations. By varying the constant term $g(0)$ the authors found all possible decompositions of $sbox(X) = f(g(X))$. Both permutations, $f(X)$ and $g(X)$, have been shared with direct sharing with three shares, (f_1, f_2, f_3) and (g_1, g_2, g_3), that are correct, non-complete and uniform. Moreover, the output of **g** is stored in the registers before it is taken as input to **f** to force the non-completeness property when the functions are cascaded as emphasized in Remark 2.

Now we consider all 4-bit permutations, and investigate when a cubic permutation from $\mathcal{S}_{16}$ can be decomposed as a *composition of quadratic permutations*. We refer to the minimum number of quadratic permutations in such a decomposition as *decomposition length*.

Lemma 9. *If a permutation p can be decomposed into a sequence of t quadratic permutations, then all permutations which are affine equivalent to p can be decomposed into a sequence of t quadratic permutations.*

Proof. Let p be a cubic permutation which can be decomposed as a composition of quadratic permutations $q_1 \circ q_2 \circ \ldots \circ q_{t-1} \circ q_t$ with length t. Let w be a permutation which is affine equivalent to p. By definition, there exist affine permutations l_r and l_l s.t. $w = l_l \circ p \circ l_r$, therefore $w = l_l \circ q_1 \circ q_2 \circ \ldots \circ q_{t-1} \circ q_t \circ l_r$. Now, by defining two quadratic permutations $q_1' = l_l \circ q_1$ and $q_t' = q_t \circ l_r$, we obtain that $w = q_1' \circ q_2 \circ \ldots \circ q_{t-1} \circ q_t'$ has a decomposition with quadratic permutations and that its length is t.

$\square$

Before investigating which permutations can be generated by combining the affine and the quadratic permutations, we prove the following two lemmas.

Lemma 10. *All 4-bit quadratic permutations belong to the alternating group $\mathcal{A}_{16}$.*

Proof. Since all affine permutations are in the alternating group (Lemma 1), two permutations which are affine equivalent, are either both even or both odd. We have taken one representative of each of the 6 quadratic classes $\mathcal{Q}_i^4$ for $i \in \{4, 12, 293, 294, 299, 300\}$ and have verified that their parities are even.

$\square$

Lemma 11. *Let q_i be one of the 6 arbitrarily selected representatives of the 6 quadratic classes $\mathcal{Q}_i^4$ (hence $i \in \{4, 12, 293, 294, 299, 300\}$). Then all cubic permutations p that have decomposition length 2, are affine equivalent to one of the cubic permutation that can be written as*

$$\tilde{p}_{i \times j} = q_i \circ l \circ q_j, \tag{5.2}$$

where l is an affine permutation and $i, j \in \{4, 12, 293, 294, 299, 300\}$.

Proof. Assume that $p = q_a \circ q_b$. Then we know that there are affine permutations $l_{ra}, l_{la}, l_{rb}, l_{lb}$ such that $p = (l_{la} \circ q_i \circ l_{ra}) \circ (l_{lb} \circ q_j \circ l_{rb})$, where q_i, q_j are two of the representatives defined above. We choose $l_r = l_{ra} \circ l_{lb}$ and $\tilde{p}_{i \times j} = l_{la}^{-1} \circ p \circ l_{rb}^{-1}$. $\square$

It follows that we can construct all cubic classes of decomposition length two by running through the 36 possibilities of $i \times j$ and the 322560 invertible affine transformations in Equation (5.2). This approach produces 30 cubic classes. Hereon, we denote the permutations $\tilde{p}_{i \times j}$ by $i \times j$ and refer to them as the *simple solutions*. Tables A.7-A.9 in Appendix A list the simple solutions for all 30 decompositions with length two. Note that if $q_i \circ l \circ q_j = p$, i.e. p can be decomposed as a product of $i \times j$, then $q_j^{-1} \circ l^{-1} \circ q_i^{-1} = p^{-1}$. Since for $n = 4$ all quadratics are affine equivalent to their inverse, it follows that p^{-1} is decomposed as a product of $j \times i$. Thus any self-inverse class has decomposition $i \times j$ and $j \times i$ as well. For the pairs of inverse classes we conclude that if $i \times j$ belongs to the first class, then $j \times i$ belongs to the second class.

To obtain all decompositions with length three we use a similar approach as for length two but the first permutation q_i is cubic (instead of quadratic) and belongs to the already found list of cubic classes decomposable with length two. It turns out that we can generate in this way the 114 remaining elements of $\mathcal{A}_{16}$.

Summarizing, we can prove the following theorem and lemma (stated without proof in [41]).

Theorem 8. *A 4-bit permutation can be decomposed using 4-bit quadratic permutations if and only if it belongs to the alternating group $\mathcal{A}_{16}$ (151 classes).*

Proof. ($\Rightarrow$) Let p be a permutation which can be decomposed with quadratic permutations, say $q_1 \circ q_2 \circ \ldots \circ q_t$. Since all $q_i \in \mathcal{A}_{16}$ (Lemma 10) and the alternating group is closed it follows that $p \in \mathcal{A}_{16}$.
($\Leftarrow$) Lemma 9, Lemma 11 and the discussion following it imply that we can generate all elements of the alternating group using quadratic permutations.

$\square$

The three left-hand-side columns of Table 5.1 list the number of classes with a quadratic decomposition from a given length for all 4-bit permutations. Theorem 8 implies that the classes which are not in the alternative group i.e. in $\mathcal{S}_{16} \setminus \mathcal{A}_{16}$, cannot be decomposed as a product of quadratic classes. Now we make the following simple observation:

Lemma 12. *Let $\tilde{p}$ be a fixed permutation in $\mathcal{S}_{16} \setminus \mathcal{A}_{16}$ then any cubic permutation from $\mathcal{S}_{16} \setminus \mathcal{A}_{16}$ can be presented as a product of $\tilde{p}$ and a permutation from $\mathcal{A}_{16}$.*

Proof. By definition, all permutations in $\mathcal{S}_{16} \setminus \mathcal{A}_{16}$ are odd permutations, and if $\tilde{p} \in \mathcal{S}_{16} \setminus \mathcal{A}_{16}$, then $\tilde{p}^{-1} \in \mathcal{S}_{16} \setminus \mathcal{A}_{16}$. Since the product of two odd permutations is even, we have: $\forall p \in \mathcal{S}_{16} \setminus \mathcal{A}_{16} : p \circ \tilde{p}^{-1} \in \mathcal{A}_{16}$. It follows that $\exists t \in \mathcal{A}_{16} : p \circ \tilde{p}^{-1} = t$, i.e. $p = t \circ \tilde{p}$.

$\square$

Now that we know the possible decompositions of 3- and 4-bit permutations, we attempt to find uniform TIs for quadratic permutations by using less than four shares, trading off with the decomposition length. However, this problem is more restrained than the basic problem, since we can use only the permutations for which we already have a uniform sharing. It turns out that the decompositions for $\mathcal{Q}_3^3$ are 1×2 and 2×1, i.e., we obtain a sharing with three shares for $\mathcal{Q}_3^3$ at the cost of decomposition length two (instead of length one). Similarly $\mathcal{Q}_{300}^4$ can be decomposed as 4×12, 4×293, 12×4, 12×294, 293×4, 293×294, 294×12 and 294×293. So, again we obtain a sharing with three shares with length two. With this result, we find TI for all 3- and 4-bit quadratic permutations with three shares.

Recall from Theorem 8 that one can find decompositions into quadratic permutations for cubic permutations in the alternating group. Therefore these permutations have a TI with three shares. However, the three-share TI of permutations outside the alternating group cannot be generated with this method. By using Lemma 12 and the sharings in Equations (3.17) and (5.3) for each linear, quadratic and cubic (when necessary) term, we obtain a TI with four shares for all 4-bit permutations.

$$f(X, Y, Z) = X \oplus YZ \oplus XYZ$$

gives:

$$f_1 = X_2 \oplus (Y_2 \oplus Y_3 \oplus Y_4)(Z_2 \oplus Z_3 \oplus Z_4)$$

$$\oplus (X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3 \oplus Y_4)(Z_2 \oplus Z_3 \oplus Z_4)$$

$$f_2 = X_3 \oplus Y_1(Z_3 \oplus Z_4) \oplus Z_1(Y_3 \oplus Y_4) \oplus Y_1 Z_1 \oplus X_1(Y_3 \oplus Y_4)(Z_3 \oplus Z_4)$$

$$\oplus Y_1(X_3 \oplus X_4)(Z_3 \oplus Z_4) \oplus Z_1(X_3 \oplus X_4)(Y_3 \oplus Y_4) \oplus X_1 Y_1(Z_3 \oplus Z_4)$$

$$\oplus X_1 Z_1(Y_3 \oplus Y_4) \oplus Y_1 Z_1(X_3 \oplus X_4) \oplus X_1 Y_1 Z_1 \tag{5.3}$$

$$f_3 = X_4 \oplus Y_1 Z_2 \oplus Y_2 Z_1 \oplus X_1 Y_1 Z_2 \oplus X_1 Y_2 Z_1 \oplus X_2 Y_1 Z_1 \oplus X_1 Y_2 Z_2$$

$$\oplus X_2 Y_1 Z_2 \oplus X_2 Y_1 Z_1 \oplus X_1 Y_2 Z_4 \oplus X_2 Y_1 Z_4 \oplus X_1 Y_4 Z_2 \oplus X_2 Y_4 Z_1$$

$$\oplus X_4 Y_1 Z_2 \oplus X_4 Y_2 Z_1$$

$$f_4 = X_1 \oplus X_1 Y_2 Z_3 \oplus X_1 Y_3 Z_2 \oplus X_2 Y_1 Z_3 \oplus X_2 Y_3 Z_1 \oplus X_3 Y_1 Z_2 \oplus X_3 Y_2 Z_1.$$

We provide all possible decompositions for all 4-bit permutations in [21]. The total length of the sharing depends on the cubic class we use ($\mathcal{C}_1^4$, $\mathcal{C}_3^4$, $\mathcal{C}_{13}^4$ and $\mathcal{C}_{301}^4$) and also on the class from the alternating group, which is used for the decomposition. For example, it can be seen in [21] that the class $\mathcal{C}_7^4$ can be decomposed using $\mathcal{C}_1^4$ with length four but with classes $\mathcal{C}_3^4$ and $\mathcal{C}_{13}^4$ it can be decomposed with length three. Note also that the number of solutions differs. We have found 10, 31 and 49 solutions when using $\mathcal{C}_1^4$, $\mathcal{C}_3^4$ and $\mathcal{C}_{13}^4$ classes, correspondingly. Table 5.1 summarizes these results.

Table 5.1: Overview of the numbers of classes of 4-bit permutations that can be decomposed and shared using 3 shares, 4 shares and 5 shares uniformly; the numbers are split up according to the decomposition length of the permutations (1, 2, 3, or 4), respectively their shares

unshared			3 shares				4 shares			5 shares	remark
1	2	3	1	2	3	4	1	2	3	1	
6			5	1			6			6	quadratics
	30			28	2			30		30	cubics in $\mathcal{A}_{16}$
		114			113	1			114	114	cubics in $\mathcal{A}_{16}$
	−				−		4	22	125	151	cubics in $\mathcal{S}_{16}\backslash\mathcal{A}_{16}$

In Tables A.1–A.6, the column *Sharing (Sh.)* describes the length of the found TI with three and with four shares, separated by a comma. Since all classes can be shared with five shares uniformly with length one, we omit this fact in these tables. Recall that for the permutations in $\mathcal{S}_{16} \setminus \mathcal{A}_{16}$ no three-share solution exists due to the degree of the permutations which is indicated in the table by a $-$.

We leave here a question for further research: Why is it the case that for all 4-bit permutations, the TIs with four shares do not improve the results significantly compared to three shares and suddenly with five shares we can share all classes uniformly with length one?

Recall that decompositions $sbox(X) = f(g(X))$ have been found in [83] for the PRESENT S-box which belongs to $\mathcal{C}_{266}^4 \in \mathcal{A}_{16}$ as shown in Table A.11. The authors made an observation that exactly $\frac{3}{7}$ sharings out of the decompositions automatically satisfy the uniformity condition (i.e. without any correction terms). Recall that with the direct sharing method without CT we (as well as the authors of [83]) were able to share only three quadratic classes uniformly: $\mathcal{Q}_4^4$, $\mathcal{Q}_{294}^4$ and $\mathcal{Q}_{299}^4$. The PRESENT S-box has 7 simple solutions (see Table A.8) but only 3 of them can be shared uniformly with direct sharing, namely 294×299, 299×294, 299×299, which explains the authors' observation.

Sharing Using Factorization

The work on decompositions presented in the previous subsection is extended by Kutzner et al. in [65]. The extension introduces *factorization* technique to enable three-share TI for the invertible 4-bit permutations not belonging to $\mathcal{A}_{16}$. In fact, factorization uses a combination of decomposition and XOR of quadratic functions which are not necessarily permutations, i.e. a permutation $p \in \mathcal{S}_{16} \setminus \mathcal{A}_{16}$ is represented as $p(X) = f(g(X)) \oplus h(X)$ where $g \in \mathcal{A}_{16}$ and h and f are defined from $\mathcal{F}_2^4$ to $\mathcal{F}_2^4$. This implementation is typically bigger in area when only a single permutation is considered. However, for the whole symmetric key algorithm, this brings a trade-off between the number of registers and the cascaded operations together with the cost of registers. Therefore, the overall cost should be examined depending on the case and the requirements.

5.1.2 Implementations

In the previous subsection, we detailed finding a uniform TI for 3- and 4-bit permutations. We observed that all 3-bit permutations have a three-share TI where the permutations from $\mathcal{Q}_3^3$ need to be decomposed. None of the

permutations needs decomposition for four- and five-share TI hence can be implemented in one clock cycle. Similar observations are made in Table 5.1 for 4-bit permutations where decomposition length can be considered as the number of cycles required for the implementation. Here, we investigate the area requirements of permutations within each class using different number of shares.

The area results listed consider a set of input or output registers. The cost of a permutation is the cost of the combinational logic required to calculate the (shared) permutation, the cost of the pipelining registers if it is required (i.e. if it is decomposed) as described in Section 5.1.1 and the cost of the n-bit (resp. $n \times s$-bit if shared) input or output register. The formulas for three shares are generated using Equations (3.5) and (3.17). For four shares, we use a variation of these equations for quadratic functions. For cubic permutations with four shares and for all permutations with five shares, we use a variation of Equation (5.3).

Table 5.2: Area comparison for randomly selected quadratic permutations in $\mathcal{S}_{16}$

3-bit Permutations Class #	Sharing Length (L)	Unshared		Shared		
		Original 1 reg	Decomp. L reg	3 shares L reg	4 shares 1 reg	5 shares 1 reg
$\mathcal{Q}_1^3$	1	23	-	120	189	176
$\mathcal{Q}_2^3$	1	24	-	129	193	184
$\mathcal{Q}_3^3$	2	24	67	243	196	190

For 3-bit permutations, we choose a permutation randomly from each class and observed the cost as shown in Table 5.2. We observe that for these small permutations, the sharing used has a big effect on area. That is why four-share implementations that use Equation (3.17) are bigger than the five-share implementations that use Equation (5.3) even though they require less registers.

Since the wide-range of S-boxes used in cryptography are from the set of 4-bit permutations, we deepen our research for these permutations. Moreover, since we use quadratic or cubic classes with length one for decomposing permutations, we concentrated our efforts on these classes and implemented 1000 permutations chosen randomly per each class. Area distributions in Figure 5.1 and the average areas in Table 5.3 show that it is advantageous to use permutations from $\mathcal{Q}_4^4$ and to avoid using permutations from $\mathcal{Q}_{299}^4$ if possible for a more area efficient implementation.

A similar argument can also be made for cubic permutations with decomposition
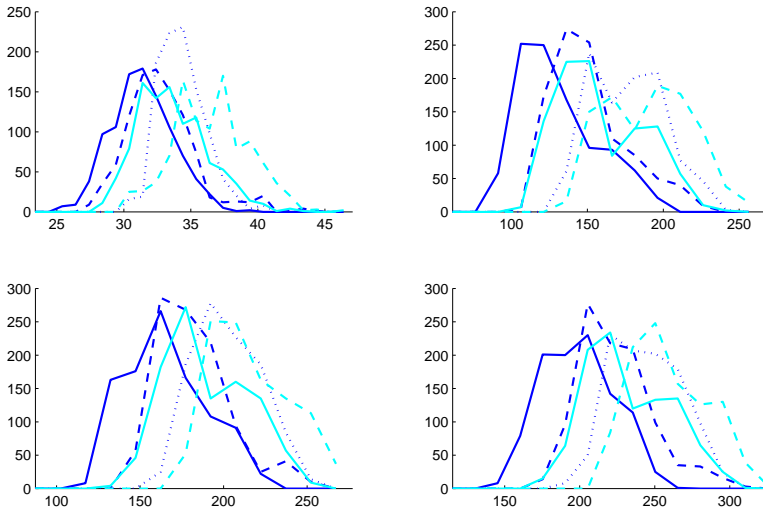
Figure 5.1: Area (x-axis) distribution of permutations from $\mathcal{Q}_4^4$ (——), $\mathcal{Q}_{12}^4$ (- -), $\mathcal{Q}_{293}^4$ (··), $\mathcal{Q}_{294}^4$ (——), $\mathcal{Q}_{299}^4$ (- -) with unshared (top, left), 3-share TI (top, right), 4-share TI (bottom, left) and 5-share TI (bottom, right) where y-axis refers to the number of permutations

Table 5.3: Average area comparison for quadratic permutations in $\mathcal{A}_{16}$ and cubic permutations in $\mathcal{S}_{16}\backslash\mathcal{A}_{16}$ which have decomposition length 1 with 3 and 4 shares respectively

4-bit Permutations Class #	Original S-box	Shared		
		3 shares	4 shares	5 shares
$\mathcal{Q}_4^4$	31	131	165	199
$\mathcal{Q}_{12}^4$	32	151	182	223
$\mathcal{Q}_{293}^4$	34	176	182	244
$\mathcal{Q}_{294}^4$	33	159	191	233
$\mathcal{Q}_{299}^4$	36	190	216	259
$\mathcal{C}_1^4$	31	-	254	322
$\mathcal{C}_3^4$	33	-	291	361
$\mathcal{C}_{13}^4$	34	-	285	349
$\mathcal{C}_{301}^4$	36	-	298	360
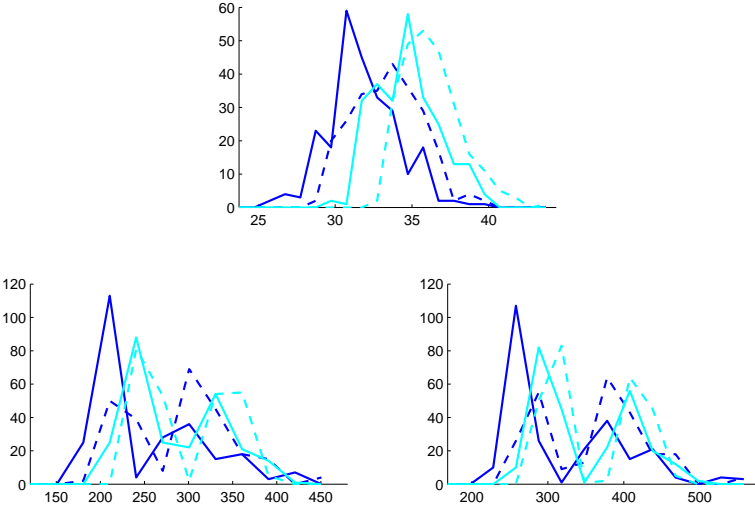
Figure 5.2: Area (x-axis) distribution of permutations from $\mathcal{C}_1^4$ (——), $\mathcal{C}_3^4$ (- -), $\mathcal{C}_{13}^4$ (——), $\mathcal{C}_{301}^4$ (- -) with unshared (top), 4-share TI (bottom, left) and 5-share TI (bottom, right) where y-axis refers to the number of permutations

length one with four shares. Namely, by Figure 5.2 and Table 5.3, we can conclude that it is more efficient to use $\mathcal{C}_1^4$ especially compared to $\mathcal{C}_{301}^4$ where the difference becomes more visible.

For classes with decomposition length more than one (Tables 5.2 and 5.4), we randomly select a class representative i.e. a permutation. For each class representative, we first list all its decompositions, analyze how many AND and XOR gates are required for these decompositions and estimate their area. Then we choose the smallest decomposition by comparing these estimations and implement it. This methodology allows us to find the decomposition with the minimum GE when a straight-forward implementation is considered. We acknowledge that the ordering and hence the smallest decomposition can change if special cells and synthesis options are used to minimize the GE. We saw that, classes $\mathcal{Q}_3^3$, $\mathcal{Q}_{300}^4$, $\mathcal{C}_{150}^4$, $\mathcal{C}_{151}^4$, $\mathcal{C}_{130}^4$, $\mathcal{C}_{131}^4$, $\mathcal{C}_{24}^4$, $\mathcal{C}_{204}^4$, $\mathcal{C}_{257}^4$ and $\mathcal{C}_{210}^4$ give relatively small results when decomposed as 2×1, 12×4, 12×293, 293×12, $12 \times 4 \times 299$, $299 \times 12 \times 4$, $299 \times 12 \times 4 \times 299$, 3×294, 3×12 and $3 \times 293 \times 12$ respectively. Observe that we use permutations from $\mathcal{Q}_{299}^4$ to decompose permutations from $\mathcal{C}_{130}^4$, $\mathcal{C}_{131}^4$ and $\mathcal{C}_{24}^4$ since there does not exist a decomposition with the same length that does not require a permutation from $\mathcal{Q}_{299}^4$. On the other hand,

Table 5.4: Area comparison for randomly selected quadratic and cubic permutations in $\mathcal{A}_{16}$ and cubic permutations in $\mathcal{S}_{16}\backslash\mathcal{A}_{16}$ which have decomposition more than one for 3 and 4 shares respectively

4-bit Permutations Class #	Sharing Length (L, L')	Unshared		Shared		
		Original 1 reg	Decomp. (L, L') reg	3 shares L reg	4 shares L' reg	5 shares 1 reg
$\mathcal{Q}^4_{300} \in \mathcal{S}_{16}$	2, 1	45	63, 41	301	266	314
$\mathcal{C}^4_{150} \in \mathcal{S}_{16}$	2, 2	38	66, 66	284	408	399
$\mathcal{C}^4_{151} \in \mathcal{S}_{16}$	2, 2	38	64, 64	267	396	490
$\mathcal{C}^4_{130} \in \mathcal{S}_{16}$	3, 2	40	88, 65	375	360	506
$\mathcal{C}^4_{131} \in \mathcal{S}_{16}$	3, 2	43	88, 62	370	404	517
$\mathcal{C}^4_{24} \in \mathcal{S}_{16}$	4, 3	41	126,102	627	678	524
$\mathcal{C}^4_{204} \in \mathcal{S}_{16}\backslash\mathcal{A}_{16}$	-, 2	39	-, 65	-	495	466
$\mathcal{C}^4_{257} \in \mathcal{S}_{16}\backslash\mathcal{A}_{16}$	-, 2	41	-, 67	-	498	492
$\mathcal{C}^4_{210} \in \mathcal{S}_{16}\backslash\mathcal{A}_{16}$	-, 3	38	-, 115	-	750	518

several possibilities for decomposing a permutation $\mathcal{Q}^4_{300}$ exist as described in Section 5.1.1 and the smallest decomposition we find also matches with the findings in Figure 5.1.

Table 5.4 also shows that depending on the decomposition length and the permutation class, a five-share TI can give smaller results than a three- or four-share TI if only the S-box is considered. The optimal invertible S-boxes used in cryptography have decomposition lengths (2,2), (3,3) or (-,3) for which an estimation can be deduced from the same table.

5.1.3   Extensions

We briefly introduce two extensions of this section. The first extension is using $s_{\mathrm{in}} = 2$ shares to achieve first-order DPA security of a quadratic permutation. The second extension, on the other hand, is increasing the security level by considering second-order TI.

First-Order TI of Quadratic Permutations with $s_{\mathsf{in}} = 2$ shares

Even though quadratic permutations are typically undesirable as an S-box, they can be used as a building block during decomposition as discussed in Section 5.1.1.

Based on this observation, we discuss here $s_{\text{in}} = 2$ share implementations of these permutation classes and leave the work on cubic classes without decomposition as future work.

In Section 3.2.1, we have already provided sharings with $s_{\text{in}} = 2$ as in Equations (3.3) and (3.4) for first-order TIs of quadratic functions. However, the mentioned sharings output $s_{\text{out}} \geq 2$ shares which causes a blow-up of output registers. Moreover, when only one S-box is considered the implementation becomes slower since the output shares must be written to the register before decreasing the number of shares back to two as suggested in Remark 3. On the other hand, the cost of initial masking, and hence generating randomness for it, becomes smaller especially when the full algorithm is studied. Our research shows that there exists at least one permutation that has a uniform first-order TI with $s_{\text{in}} = 2$ shares in all quadratic 3- and 4-bit classes. More specifically, permutations in $\mathcal{Q}_1^3$, $\mathcal{Q}_2^3$, $\mathcal{Q}_3^3$, $\mathcal{Q}_4^4$, $\mathcal{Q}_{12}^4$, $\mathcal{Q}_{293}^4$, $\mathcal{Q}_{294}^4$, $\mathcal{Q}_{299}^4$ and $\mathcal{Q}_{300}^4$ have a TI satisfying the first three properties when the sharing of the linear and quadratic terms are generated as in Equation (3.3). In Equation (5.5) we provide an example for the representative of $\mathcal{Q}_{294}^4$ (Table A.6) which has the algebraic form in Equation (5.4).

$$A = f^1(W, X, Y, Z) = W$$
$$B = f^2(W, X, Y, Z) = X \qquad\qquad (5.4)$$
$$C = f^3(W, X, Y, Z) = WX \oplus Y$$
$$D = f^4(W, X, Y, Z) = WY \oplus Z$$

$$
\begin{array}{lll}
A_1 = W_1 & C_1 = W_1 X_1 \oplus Y_1 & D_1 = W_1 Y_1 \oplus Z_1 \\[4pt]
A_2 = W_2 & C_2 = W_1 X_2 & D_2 = W_1 Y_2 \\[4pt]
B_1 = X_1 & C_3 = W_2 X_1 & D_3 = W_2 Y_1 \\[4pt]
B_2 = X_2 & C_4 = W_2 X_2 \oplus Y_2 & D_4 = W_2 Y_2 \oplus Z_2
\end{array}
\qquad (5.5)
$$

Notice that the first two output bits have $s_{\text{out}} = 2$ shares whereas the last two output bits have $s_{\text{out}} = 4$ shares. We decrease the number of shares of the latter two as $C_1 \oplus C_2$ (resp. $D_1 \oplus D_2$) and $C_3 \oplus C_4$ (resp. $D_3 \oplus D_4$). The two-share output of the permutation is uniform. Similar sharings apply for permutations from the other aforementioned classes of which the examples for 4-bit permutations are given in Appendix B.1.

Constructing Second-Order TI of Some Quadratic Permutations

As stated in Section 5.1.1, all quadratic 3- and 4-bit permutations have uniform first-order five-share TIs with *direct sharing*. More precisely, consider the sharing in Equation (5.1) which is given for the function $A = f(X, Y, Z) = 1 \oplus X \oplus YZ$. If all of the shared linear and quadratic terms in the component functions of the analysed permutation are distributed to the output shares similar to the sharings of X and YZ respectively, the generated TI is both first-order DPA resistant and uniform (up to affine equivalence).

In order to generate a uniform second-order TI of a permutation, we start with this first-order uniform TI. We distribute each shared linear and quadratic term to component functions as in Equation (5.6) (given for the aforementioned function). This new sharing with $s_{\mathrm{in}} = 5$ and $s_{\mathrm{out}} = 10$ shares satisfies correctness (Property 2) and second-order non-completeness (Property 3). Hence, given Property 1, this sharing provides security against second-order DPA by Theorem 1.

$$
\begin{aligned}
A_1 &= 1 \oplus X_2 \oplus Y_2 Z_2 \oplus Y_2 Z_3 \oplus Y_3 Z_2 & A_2 &= Y_2 Z_4 \oplus Y_4 Z_2 \\
A_3 &= X_3 \oplus Y_3 Z_3 \oplus Y_3 Z_4 \oplus Y_4 Z_3 & A_4 &= Y_3 Z_5 \oplus Y_5 Z_3 \\
A_5 &= X_4 \oplus Y_4 Z_4 \oplus Y_4 Z_5 \oplus Y_5 Z_4 & A_6 &= Y_4 Z_1 \oplus Y_1 Z_4 & (5.6) \\
A_7 &= X_5 \oplus Y_5 Z_5 \oplus Y_5 Z_1 \oplus Y_1 Z_5 & A_8 &= Y_5 Z_2 \oplus Y_2 Z_5 \\
A_9 &= X_1 \oplus Y_1 Z_1 \oplus Y_1 Z_2 \oplus Y_2 Z_1 & A_{10} &= Y_1 Z_3 \oplus Y_3 Z_1
\end{aligned}
$$

In order to satisfy the uniformity of the sharing (Property 4), we reduce the number of shares from s_{out} to s_{in} using the following equation where **B** is the five-share output.

$$B_i = A_{2i-1} \oplus A_{2i} \text{ for } i \le 5$$

The final sharing of the permutation (or one of its affine equivalent) is uniform since it becomes equal to the sharing given in Equation (5.1) after this reduction. Hence, we can construct uniform second-order TI of all 3- and 4-bit quadratic permutations.

The area requirement of this second-order TI can be calculated by adding the register cost required by Remark 3 to the first-order five-share TI cost of the particular permutation.

5.2   5- and 6-bit Permutations

In this section, we discuss the first-order TI of 5- and 6-bit permutations. Due to the high amount of affine equivalence classes of these sizes, we limit our analysis to the AB and APN permutations which have a particular cryptographic significance as stated in Section 2.2.3. We leave the arguments about 5-bit KECCAK S-box which also has high importance to Chapter 6.

5.2.1   Finding Uniform Threshold Implementations

Similar to the 3- and 4-bit permutations in Section 5.1, we use $s_{\text{in}} \geq t + 1$ shares where t is the degree of the permutation and fix $s_{\text{out}} = s_{\text{in}}$. The algebraic degrees of AB permutations are provided in Table 2.2, namely $AB1$ and $AB2$ are quadratic whereas $AB3$ and $AB4$ are cubic. The algebraic degree of the permutations from the only known APN permutation class is four.

The direct sharing with three shares given in Equation (3.5) for linear and quadratic terms fails to produce a uniform TI for $AB1$ and $AB2$. Furthermore, it is unfeasible to go through all CT in order to find a uniform sharing with our current methods. Unfortunately, our limited search did not reveal a TI that satisfies all the properties. However, we find uniform four-share TIs using CT for those permutations which can be generated by applying Equation (5.3) for linear and quadratic terms of the coordinate function.

For $AB3$ and $AB4$, we could not find a uniform sharing up to five shares with our current methods. However, it is possible to use TIs with four shares with the cost of re-masking. Note that these results do not imply that a uniform TI with three shares for $AB1$ and $AB2$ or with four shares for $AB3$ and $AB4$ (resp. for their affine equivalents) is nonexistent since we abstain from going through all the CT due to its complexity.

The findings for AB permutations, namely the difficulty of finding uniform TIs even for 5-bit cubic permutations, lead us to work on a decomposition for this degree four APN permutation. Dillon shows in [44] that this permutation can be decomposed into two permutations of degree three and two. An example of an APN permutation with the decomposition $APN(X) = f(g(X))$, where f is cubic and g is quadratic, is provided in Table 5.5. Unfortunately, with our current methods it is unfeasible to find uniform sharings for f and g. However, with this decomposition, it is possible to have a 4-share uniform TI with re-masking.

As the S-box becomes bigger and more complicated, we observe that finding a

Table 5.5: Representative of the known APN permutation in $\mathbb{GF}(2^6)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
APN	0	16	60	54	17	14	23	59	29	62	63	10	39	8	49	51
f	0	13	63	50	2	15	48	61	54	58	22	26	38	42	11	7
g	0	48	37	8	19	18	41	42	39	21	2	45	26	40	17	33
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
APN	45	37	61	48	47	5	12	20	36	57	40	46	26	56	43	55
f	46	49	14	17	33	62	12	19	24	6	39	57	5	27	55	41
g	32	60	7	6	51	28	22	59	43	27	61	16	11	57	46	30
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
APN	11	31	24	6	27	13	53	19	15	30	1	4	33	34	28	35
f	45	51	1	31	34	60	3	29	8	23	59	36	21	10	43	52
g	14	35	24	25	29	1	56	23	5	53	34	62	20	36	49	50
	48	49	50	51	52	53	54	55	56	57	58	9	60	61	62	63
APN	21	52	58	3	9	7	18	32	25	22	41	50	44	2	38	42
f	16	28	35	47	18	30	44	32	53	56	25	20	37	40	4	9
g	44	47	9	38	63	15	52	55	58	10	31	3	54	4	12	13

uniform sharing becomes much more difficult and the search for CT becomes unfeasible. At this point, we leave the problem of finding TIs for APN permutations of size 6 and AB permutations of degree three of size 5 as an open question and suggest re-masking to satisfy uniformity.

The only known examples of using 5-bit AB or 6-bit APN permutations of size 6 are the authenticated encryption algorithms Fides [15] (AB1 and APN) and PRIMATEs [4] (AB1), which are designed to provide provable security against first-order DPA attacks. The selection of particular permutation is done by observing the number of AND and XOR gates required for each permutation in a class and finding a trade-off of this gate count between the unshared and the shared implementations.

Another example of using S-boxes with 6-bit inputs is the Data Encryption Standard (DES) [43]. A 6×4 S-box used by DES can be implemented as four 4×4 invertible S-boxes followed by a cubic selection function (4-to-1 multiplexer). The selection function can be implemented uniformly with direct sharing with four shares and we have TI for all 4×4 invertible S-boxes. However, the distribution of the input of the selection function, which combines the outputs of 4×4 S-boxes as input is not necessarily uniform. To avoid first-order leakage,

the input of the selection function should be checked carefully and should be re-masked when necessary.

5.2.2 Implementations

Since we failed to find TIs for cubic 5-bit AB permutations with 4 shares as described in Section 5.2.1, we chose to implement only the quadratic AB permutations. For each class we provide area results for a randomly selected representative with the library and conditions given in the beginning of this chapter. We observe that the cost of the TI is a little bit over four times the cost of the unshared version.

On the other hand, we implement the APN permutation in Table 5.5 with the given decomposition even though the sharing is not uniform since that is the only known class. Therefore for that implementation, an extra cost of re-masking should be added. We observe that even the unshared-implementation cost is high as a result of the high degree and with decomposition this cost might reduce. This four-share implementation is $3, 5$ times the cost of the unshared version. We summarize the results for 5- and 6-bit permutations in Table 5.6.

Table 5.6: Quadratic AB and APN S-boxes' sharing

Permutation	Sharing	Unshared		Shared
	Length	Original	Decomposed	4 shares
Class	(L)	1 reg	L reg	L reg
$AB1$	1	68	-	303
$AB2$	1	64	-	274
APN	2	224	192	795

5.3 Conclusion

In this chapter, we mainly discussed finding uniform $s_{\text{in}} \geq t + 1$, $s_{\text{out}} = s_{\text{in}}$ first-order TIs for 4-bit permutations which form the majority of the small S-boxes used in cryptographic algorithms. Our findings cover all 3-bit permutations due to the relationship between them and some of the 4-bit permutations given in Lemma 7. We showed that all these permutations have uniform TIs with

$s_{\text{in}} = s_{\text{out}} = 5$ shares that can be implemented in a single clock cycle. Since generating uniformly shared inputs to a nonlinear function requires randomness, we performed methods such as decomposition and using correction terms in order to decrease the number of shares. We showed that all 3- and 4-bit permutations have uniform 4-share TIs with different decomposition lengths as displayed in Figure 5.1. We used permutations from $\mathcal{Q}_1^4$, $\mathcal{Q}_3^4$, $\mathcal{Q}_{13}^4$ and $\mathcal{Q}_{301}^4$ in addition to the quadratic classes to decompose 4-bit permutations. Therefore, we focused on the area requirements of implementing permutations from these classes which are used as building blocks. We emphasized the use of registers in between functions in order to achieve security when implementation is decomposed then reflected their costs in our calculations. We showed that half of the classes from 4-bit permutations and all the 3-bit permutations are in $\mathcal{A}_{16}$ and $\mathcal{A}_8$ respectively which can be implemented using 3-shares. For both all the cubic classes in $\mathcal{A}_{16}$ and the special classes $\mathcal{Q}_3^3$ and $\mathcal{Q}_{300}^4$, we used decomposition into quadratic permutations in order to find a uniform three-share TI. For the rest of the classes in $\mathcal{A}_{16}$ and $\mathcal{A}_8$, we used direct sharing or sharing with CT. We left it as an open question to find why $\mathcal{Q}_3^3$ and $\mathcal{Q}_{300}^4$ behave different that the rest of the quadratic permutations.

By using the information provided in this chapter, in the tool-box [20] and in the decomposition list [21], we can apply the TI technique to any 3- or 4-bit permutation. Such a TI can be generated with high flexibility by using different number of shares or clock cycles. To generalize, a cubic 4-bit S-box with good cryptographic properties requires approximately five times the area of its unshared implementation when a five-share TI is used. Depending on the decomposition length, its four-share implementation is four to seven times bigger and up to three times slower than its unprotected implementation. A three-share TI can become up to four times slower and its area requirement is three to six times more than its plain implementation.

When 5- and 6-bit permutations are considered, the number of affine equivalent classes is too high to examine them all. Therefore, we choose to investigate the TI of AB and APN permutations only. Generating uniform TIs for these permutations which have cryptographic significance is challenging due to their relatively complex algebraic representations. We could only find uniform sharing for two out of four of the AB classes. These uniform sharings for these permutations which use four shares are approximately four times bigger than their unshared versions. For the rest of the 5-bit AB and 6-bit APN permutations, we are forced to use re-masking. We leave finding uniform TIs for these permutations as an open question. Note that we used these findings during the designs of FIDES and PRIMATEs authenticated encryption algorithms. Their S-boxes are chosen from the AB1 and the mentioned APN class such that both the unshared and the four-share TI have low area requirements compared

to their affine equivalents.

We believe that this work on small S-boxes is only complete after discussing first-order TI with two input shares and higher-order TIs. We used only 3- and 4-bit permutations for this discussion. We showed that there exists at least one permutation from all 3- and 4-bit quadratic classes that has a uniform TI with $s_{\mathrm{in}} = 2$ shares. However, such an implementation requires extreme care not to unmask the sensitive information as discussed in Section 3.3.2 and is secure only if the input sharings are independent of each other. Therefore, we do not suggest this as a first option to consider unless the implementor is experienced and the nonlinear function is deeply analyzed.

Higher-order implementations are relatively new compared to the rest of the information provided in this thesis. Here, we only provided uniform second-order TIs of quadratic 3- and 4-bit permutations with $s_{\mathrm{in}} = 5$ and $s_{\mathrm{out}} = 10$ shares. This sharing is generated from the five-share first-order TI of these permutations. We leave the more detailed analysis as a future work.

6

First-Order Threshold Implementations of KECCAK

Chapter 4 dealt with TI of a simple block cipher that uses only an AND/XOR gate to achieve nonlinearity whereas Chapter 5 employed uniform TIs of single small permutations that form the nonlinear substitution layer of many cryptographic algorithms. Neither of these chapters considered special properties of the cryptographic algorithm in hand. In this chapter, we investigate the KECCAK algorithm of which a subset is defined as the new hash function standard SHA-3. The goal of this chapter is to look beyond a single (5-bit) nonlinear permutation during a TI. We first discuss TI of the permutation layer composed of several of this permutation then extend TI to the full algorithm. We discuss several implementation trade-offs starting with Question 2, i.e. achieving uniformity of the nonlinear layer by using extra registers with increased area or by using re-masking with extra randomness requirements. We also provide algorithm specific optimizations to decrease the random bit usage. Even though we only discuss first-order TI with the same number of input and output shares, this work can be extended to higher-orders as will be discussed in Section 6.6.

Before moving further with the details, we first provide a brief introduction to KECCAK in Section 6.1. The first TI of KECCAK is suggested by its designers in [9]. Unfortunately, that three-share TI fails to satisfy Property 4 leading to possible insecure implementations. We start Section 6.2 with an off-the-shelf three-share TI of KECCAK using re-masking, which is then followed by

optimized versions to decrease the amount of required random bits. On the other hand, the four-share implementation provided in Section 6.3 satisfies all TI properties. In Section 6.4, we examine the performances of these implementations and their unprotected version using both serialized and parallel architectures. We use Faraday, FSA0A-D and FSC0H-D libraries which are standard cell libraries tailored for UMC 0.18μm and UMC 0.13μm logic processes respectively to observe and compare the size (GE) and the maximum frequency (MHz) accurately with the previous works. In addition we provide results from $45nm$ NanGate standard cell library for future comparison.

We conclude this chapter by suggesting a TI of KECCAK which uses only two shares for the affine transformations and three shares for the nonlinear permutations in order to further decrease the area with additional randomness needs.

This work is published in [16]. The work of reducing the randomness of three-share TI in Section 6.2 is mainly suggested by the coauthors Joan Daemen and Gilles Van Assche.

6.1 Introduction to KECCAK

KECCAK is a function with variable-length input and arbitrary-length output based on the *sponge construction* [10]. In this construction (Fig. 6.1), a *b*-bit permutation f, which is typically an SPN, is iterated. First, the input is padded and its blocks are *absorbed* sequentially into the state, with a simple XOR operation. Then, the output is *squeezed* from the state block by block. The size of the blocks is denoted by r and called the *bit-rate*. The remaining number of bits $c = b - r$ is called the *capacity* and determines the security level of the function.
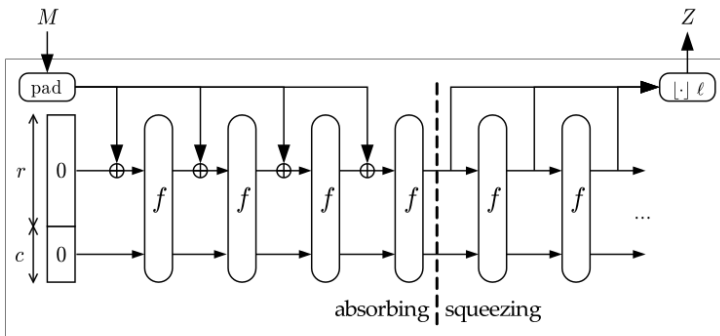
Figure 6.1: Sponge function construction [1]

The simplest use case of a sponge function is to use it as a hash function. However, a MAC function can be built by taking the concatenation of a secret key and a message as input. It is also possible to use a sponge function as a stream cipher. To this purpose, it suffices to use the secret key and a nonce as input so that the resulting output can be used as a key stream. More modes of use are described in [14].

Seven permutations, denoted by KECCAK-$f[b]$, are defined with width $b = 25w$ ranging from 25 to 1600 bits, with w increasing in powers of two. The state of KECCAK-$f[b]$ is organized as a $5 \times 5 \times w$ bit array with (x, y, z) coordinates each bit of which is denoted by $X^{(x,y,z)}$. Coordinates are taken modulo 5 for x and y and modulo w for z. A *row* is a set of 5 bits with given (y, z) coordinates, a *column* is a set of 5 bits with given (x, z) coordinates and a *lane* is a set of w bits with given (x, y) coordinates. Moreover, the set of 5×5 bits with given z coordinates is called a *slice*.
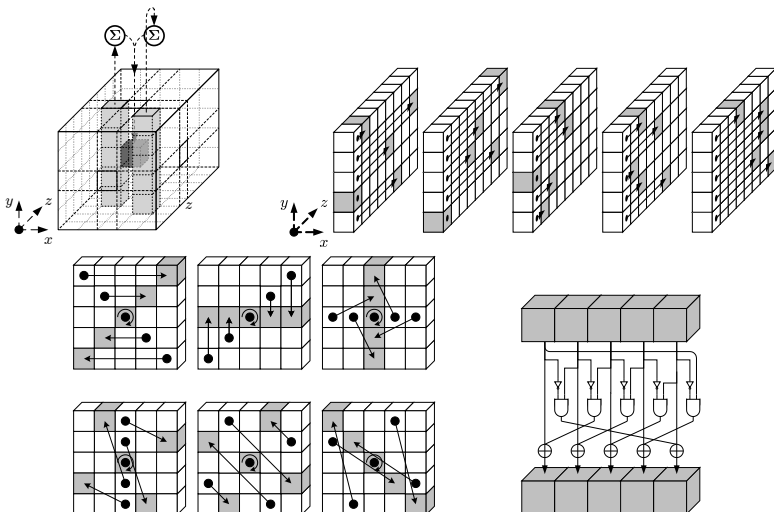
Figure 6.2: Steps of the round function; from top left picture, clock-wise, θ, ρ, χ and π [1].

The round function of KECCAK-$f[b]$ consists of the following steps, which are only briefly summarized here and pictured in Figure 6.2. For more details, we refer to the specifications [11].

- θ is a linear mixing layer that adds a pattern depending solely on the parity of the columns of the state.

- ρ and π displace bits without altering their value.

- χ is a nonlinear quadratic mapping that processes each row independently. It can be seen as the application of the following function for each output bit in a row in order to generate a 5-bit quadratic permutation (S-box) per row:

$$A^{(x,y,z)} = X^{(x,y,z)} \oplus (X^{(x+1,y,z)} \oplus 1)X^{(x+2,y,z)}.$$

- ι adds a round constant.

The linear operations θ, ρ and π together is represented as λ. The number of rounds in KECCAK-f is determined by the width b of the permutations. It is 12 for KECCAK-$f[25]$ and increases by two for each doubling of the size. So KECCAK-$f[1600]$, which we work with throughout this chapter, has 24 rounds.

6.2 Three-share Threshold Implementation

The first three-share TI of KECCAK is proposed in [9] by the KECCAK designers. In that document, given the uniform three-sharing $\mathbf{X} = (X_1, X_2, X_3)$ for a row of bits (X is a 5-bit element), where $\mathbf{X}^x = (X_1^x, X_2^x, X_3^x)$ represents one bit of the row, the sharing for the nonlinear χ operation is defined as in Equation (6.1).

$$A_1^i = \chi_1^i(X_2, X_3) = X_2^i \oplus (X_2^{i+1} \oplus 1)X_2^{i+2} \oplus X_2^{i+1}X_3^{i+2} \oplus X_2^{i+2}X_3^{i+1}$$

$$A_2^i = \chi_2^i(X_3, X_1) = X_3^i \oplus (X_3^{i+1} \oplus 1)X_3^{i+2} \oplus X_3^{i+1}X_1^{i+2} \oplus X_3^{i+2}X_1^{i+1} \qquad (6.1)$$

$$A_3^i = \chi_3^i(X_1, X_2) = X_1^i \oplus (X_1^{i+1} \oplus 1)X_1^{i+2} \oplus X_1^{i+1}X_2^{i+2} \oplus X_1^{i+2}X_2^{i+1}$$

This sharing maps a 15-bit vector (X_1, X_2, X_3) to a 15-bit vector (A_1, A_2, A_3). Moreover, the sharing is a direct three-share TI where the component functions are equal, i.e. generated in a cyclic manner; and the sharing highly resembles to the first-order sharing of the AND/XOR gate of KATAN (Section 4.2.2). From now on, we use the notation χ' to represent χ_1^i, χ_2^i and χ_3^i to ease the notation.

Upon inspection, we find that this three-share TI is not invertible and hence, not uniform. Similar to the 5-bit AB permutations' case in Section 5.2, it is infeasible to try all CT to find a uniform TI and our research with a limited number of CT did not reveal a positive result. Hence, we propose re-masking as defined in Equation (3.13) to remedy this problem.

KECCAK-f[1600] has 320 rows. For a three-share TI, this implies the application of Equation (6.1) together with the application of Equation (3.13) 320 times per round at an extra cost of 2 uniformly distributed random bits per state bit. Although from a theoretical point of view this re-masking method solves the uniformity issue raised above, the solution is unsatisfactory since it requires a random number generator (RNG) which generates many high-quality random bits at each clock cycle.

In the following subsections, we decrease the number of extra random bits required initially by each nonlinear operation, followed by a discussion on the whole substitution layer.

6.2.1 Less Randomness per Row

In this subsection, we reduce the number of required fresh random bits per χ operation by using specific properties of its sharing in Equation (6.1).

Lemma 13. *Let (A_1, A_2, A_3) be a (not necessarily uniform) three-sharing of the n-bit value A and (B_1, B_2, B_3) be a uniform three-sharing of the m-bit value B. Let (C_1, C_2, C_3) be uniform three-sharing of the n-bit value C statistically independent of (A_1, A_2, A_3) and (B_1, B_2, B_3). Then, $((A_1 \oplus C_1, B_1), (A_2 \oplus C_2, B_2), (A_3 \oplus C_3, B_3))$ is a uniform three-sharing of the $n + m$-bit value $(A \oplus C, B)$.*

Proof. First, since $A_1 \oplus A_2 \oplus A_3$, $B_1 \oplus B_2 \oplus B_3$ and $C_1 \oplus C_2 \oplus C_3$ take a fixed value with probability one, so does $(A_1 \oplus A_2 \oplus A_3 \oplus C_1 \oplus C_2 \oplus C_3, B_1 \oplus B_2 \oplus B_3)$. Then, it suffices to verify that for each fixed value $a_1 \oplus c_1, b_1, a_2 \oplus c_2, b_2$:

$$\Pr(A_1 \oplus C_1 = a_1 \oplus c_1, A_2 \oplus C_2 = a_2 \oplus c_2, B_1 = b_1, B_2 = b_2)$$

$$= \sum_{c_1, c_2} \Pr(C_1 = c_1, C_2 = c_2) \hookleftarrow$$

$$\hookrightarrow \Pr(A_1 = (a_1 \oplus c_1) \oplus c_1, A_2 = (a_2 \oplus c_2) \oplus c_2, B_1 = b_1, B_2 = b_2)$$

$$= 2^{-2n} \sum_{c_1, c_2} \Pr(A_1 = (a_1 \oplus c_1) \oplus c_1, A_2 = (a_2 \oplus c_2) \oplus c_2, B_1 = b_1, B_2 = b_2)$$

$$= 2^{-2n} \Pr(B_1 = b_1, B_2 = b_2)$$

$$= 2^{-2(n+m)}.$$

$\square$

The function χ in KECCAK operates on 5-bit rows. It can be seen as a specific case of a convolutional mapping operating on an n-bit circular array with updating function $A^i = X^i \oplus (X^{i+1} \oplus 1)X^{i+2}$. Next lemma is a general result that holds for any value n.

Lemma 14. *If the input $(X_1, X_2, X_3)^{0\ldots n-1}$ to the shared function of χ is an n-bit uniform masking, the output truncated to any $n-2$ consecutive bits, e.g., $(A_1, A_2, A_3)^{0\ldots n-3}$, is shared uniformly.*

Proof. First, consider $(A_1^{n-3}, A_2^{n-3}, A_3^{n-3})$ which is calculated as in Equation (6.1). It is the result of summing $(X_2^{n-3}, X_3^{n-3}, X_1^{n-3})$ with bits computed from X_1, X_2 and X_3 in positions $n-2$ and $n-1$. As $(X_2^{n-3}, X_3^{n-3}, X_1^{n-3})$ is a uniform sharing of X^{n-3} independent of input bits in positions $n-2$ and $n-1$, Lemma 13 applies and hence $(A_1^{n-3}, A_2^{n-3}, A_3^{n-3})$ is a uniform sharing.

Assuming $(A_1, A_2, A_3)^{i+1\ldots n-3}$ is a uniform sharing, we can prove that $(A_1, A_2, A_3)^{i\ldots n-3}$ is a uniform sharing as follows. (A_1^i, A_2^i, A_3^i) is the result of summing (X_2^i, X_3^i, X_1^i) with bits computed from $(X_1, X_2, X_3)^{i+1\ldots i+2}$. As (X_2^i, X_3^i, X_1^i) is a uniform sharing of X^i and is independent of input bits in positions $i+1$ and $i+2$ and of $(A_1, A_2, A_3)^{i+1\ldots n-3}$, Lemma 13 applies and hence $(A_1, A_2, A_3)^{i\ldots n-3}$ is a uniform sharing. This can be extended till $(A_1, A_2, A_3)^{0\ldots n-3}$. $\square$

Further (cyclic) extensions to include $(A_1, A_2, A_3)^{n-1}$ or $(A_1, A_2, A_3)^{n-2}$ is not possible as $(X_2^{n-2}, X_3^{n-2}, X_1^{n-2})$ is not independent of $(A_1, A_2, A_3)^{0\ldots n-3}$ and Lemma 13 no longer applies.

Lemma 14 says that the truncated output with two successive bits removed is uniform. As a consequence, one can repair uniformity using only 4 fresh random bits $M_1^{n-2}, M_2^{n-2}, M_1^{n-1}, M_2^{n-1}$. In particular, we just apply Equation (3.13) with $M_1^i = M_2^i = 0$ for $0 \leq i < n-2$.

We decreased the number of fresh random bits per round from 10 to 4 bits per row. However, for KECCAK-$f[1600]$ this is $320 \times 4 = 1280$ bits, still too expensive in practice.

6.2.2 Jointly Satisfying Uniformity

In this subsection, we consider uniformity of the full state rather than of the individual rows. We propose a TI of χ with interaction between the rows that achieves almost uniformity at the level of the full state, greatly reducing the required number of fresh random bits per round.

Let us for convenience refer to the number the rows with index $j = y + 5z$. The idea is to make the sharing at the output of row $j + 1$ uniform by using input at row j. In straightforward way, we add $(X_1 \oplus X_2, X_1, X_2)$ at the input of row j to the output (A_1, A_2, A_3) of row $j + 1$. This is again a straightforward application of Lemma 13. Note that to satisfy the independence required by Lemma 13, the last row still requires injection of four fresh random bits for achieving uniformity, as in Equation (3.13). The circuit complexity can be reduced greatly by combining this with Lemma 14. As a matter of fact, we have to add $(X_1 \oplus X_2, X_1, X_2)$ at the input of row j to the output (A_1, A_2, A_3) of row $j + 1$ in only two successive bit positions. Care must be taken in the bit positions used in each row so as to be able to rely on Lemma 14.

The above reasoning points out that each row individually can become uniform. The key point, however, is to show that the joint application on the entire state yields a uniform realization of χ. This is what the theorem below shows.

We denote the three shares of the whole state by (X_1, X_2, X_3), and a 5-bit row of the state as $(X_1[j], X_2[j], X_3[j])$ with $j \in \mathbb{Z}_{5w}$. Then, the implementation of χ becomes:

$$A_1[j]^i = \chi'(X_2[j], X_3[j]) \oplus X_1[j-1]^i \oplus X_2[j-1]^i,$$

$$A_2[j]^i = \chi'(X_3[j], X_1[j]) \oplus X_1[j-1]^i, \qquad\qquad (6.2)$$

$$A_3[j]^i = \chi'(X_1[j], X_2[j]) \oplus X_2[j-1]^i,$$

if $j > 0$ and $i \in \{3, 4\}$. Otherwise, Equation (3.13) applies when $j = 0$, and Equation (6.1) suffices for positions $i \leq 2$.

Theorem 9. *If the (whole state) input (X_1, X_2, X_3) to Equation (6.2) when $j > 0$ and $i \in \{3, 4\}$, to Equation (3.13) when $j = 0$ and $i \in \{3, 4\}$ and to Equation (6.1) when $i \leq 2$, is shared uniformly, then the (whole state) output (A_1, A_2, A_3) is shared uniformly.*

Proof. We can apply Lemma 13 recursively, with j starting at $j = 5w - 1$ and going down to $j = 0$. Every time, the reasoning is to show that if $(A_1[j+1 \ldots 5w - 1], A_2[j+1 \ldots 5w - 1], A_3[j+1 \ldots 5w - 1])$ is uniform, then it is also uniform for rows j to $5w - 1$.

Following Equation (6.2), the sharing $(A_1[j], A_2[j], A_3[j])$ is obtained by adding $(\chi'(X_2[j], X_3[j]), \chi'(X_3[j], X_1[j]), \chi'(X_1[j], X_2[j]))$ and $(X_1[j-1] \oplus X_2[j-1], X_1[j-1], X_2[j-1])$ for bit positions $i \in \{3, 4\}$. The latter expression is a uniform sharing of 0 and independent of the rows with indexes j and higher.

From Lemma 14, $(\chi'(X_2[j], X_3[j]), \chi'(X_3[j], X_1[j]), \chi'(X_1[j], X_2[j])$ is already uniform when restricted to bit positions 0 to 2. The conditions of Lemma 13 are thus satisfied and $(A_1[j+1\ldots 5w-1], A_2[j+1\ldots 5w-1], A_3[j+1\ldots 5w-1])$ is uniform

If $j = 0$, the same reasoning applies, except that bit positions $i \in \{3, 4\}$ are obtained as in Equation (3.13). $\qquad \square$

The new cost is four fresh random bits per round, some additional XORs, registers and extra routing. As far as randomness is concerned, this amounts to 96 bits for the 24 rounds of KECCAK-$f[1600]$, which is small compared to the 3200 random bits needed to represent the input state in three shares with the naive re-masking approach.

6.3 Four-share Threshold Implementation

Similar to the quadratic almost bent permutations, we find a uniform sharing of χ with four shares. Unlike the representation in Equation (6.1), this sharing is not cyclic. For $i = 0, 1, 2, 4$, we have:

$$A_1^i = X_2^i \oplus X_2^{i+2} \oplus ((X_2^{i+1} \oplus X_3^{i+1} \oplus X_4^{i+1})(X_2^{i+2} \oplus X_3^{i+2} \oplus X_4^{i+2}))$$

$$A_2^i = X_3^i \oplus X_3^{i+2} \oplus (X_1^{i+1}(X_3^{i+2} \oplus X_4^{i+2}) \oplus X_1^{i+2}(X_3^{i+1} \oplus X_4^{i+1}) \oplus X_1^{i+1}X_1^{i+2})$$

$$A_3^i = X_4^i \oplus X_4^{i+2} \oplus (X_1^{i+1}X_2^{i+2} \oplus X_1^{i+2}X_2^{i+1})$$

$$A_4^i = X_1^i \oplus X_1^{i+2},$$

$$(6.3)$$

and for the remaining (third) coordinate function we have:

$$A_3' = X_2^3 \oplus X_2^0 \oplus X_3^0 \oplus X_4^0 \oplus ((X_2^4 \oplus X_3^4 \oplus X_4^4)(X_2^0 \oplus X_3^0 \oplus X_4^0))$$

$$B_3' = X_3^3 \oplus A_0 \oplus (X_1^4(X_3^0 \oplus X_4^0) \oplus X_1^0(X_3^4 \oplus X_4^4) \oplus X_1^0X_1^4)$$

$$C_3' = X_4^3 \oplus (X_1^4X_2^0 \oplus X_1^0X_2^4)$$

$$D_3' = X_1^3.$$

$$(6.4)$$

We found this sharing by using Theorem 6. Namely, we first searched through all affine equivalent S-boxes of χ, i.e., $\bar{\chi} = \chi(l(X))$, where $l(X)$ is an affine permutation and we found the ones that can be shared with a direct sharing. Next, we applied the corresponding inverse affine transformation to the found direct sharing to generate a uniform sharing for the function χ. We chose the one that has the smallest area over all the candidates. Therefore, this uniform sharing (although derived and close to direct) is not a direct sharing and that is why the shares can not be computed in a circular manner.

6.4   Implementations

There are several reports on different implementations of unprotected KECCAK-f that use different platforms, architectures and libraries [2].   We provide unprotected (plain) implementations of KECCAK-f with a round-based (parallel, Fig. 6.3) and a slice-based (serial, Fig. 6.4) architecture and build TIs on them for fair comparison. The D flip-flops (DFFs) that take the output of a 2×1 multiplexer (MUX) as input are implemented as scan flip-flops (SFFs) to reduce the area.

6.4.1   Unprotected Implementations

Parallel Implementation

The rate of this version is fixed to be at most 1024 bits as depicted by Figure 6.3. It is assumed that the inputs are the 1600-bit state from the previous iteration of the KECCAK-f together with one message block.
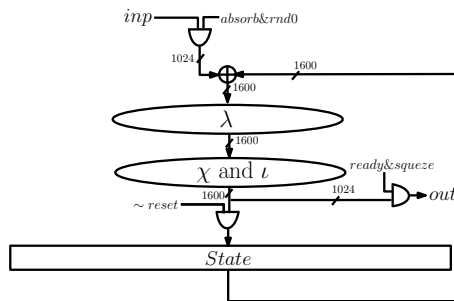
Figure 6.3: Schematic of the round-based implementation of KECCAK-f

The architecture of the round function KECCAK-f is straightforward with 320 parallel instances of χ. The function θ is implemented in a slice-based manner. Namely, the 5-bit XOR of every row in each slice (i.e., the column parity) P_i, where $i \in \{0, \ldots, 63\}$ is calculated in parallel [12]. For each slice, the rotated values of P_i and P_{i-1} are XORed. This new value is concatenated five times to generate a 25-bit value which is then XORed to its corresponding slice. With this method, the θ function can be calculated with a low cost. The rest of the linear layer, i.e., ρ and π, are executed on the whole 1600-bit state as a simple wiring and the output in each round is written to a 1600-bit register. Hence, one iteration of KECCAK-$f[1600]$ takes 24 clock cycles.

Serial Implementation

This version, of which the schematic is given in Figure 6.4, operates on 25-bit slices. Namely, it takes 25 bits in each clock cycle starting from slice 0 as input. The column parity P_i of each input slice is calculated and written to a five bit register to be used in the following clock cycle. The θ calculation is also performed in the same clock cycle for each input slice using its parity and the parity from the previous slice that was stored in the register with the exception of the first slice. The output of θ is first stored in the register R_{63}, then shifted from R_{i+1} to R_i for $i \in \{0, \ldots, 62\}$. θ for the first slice is completed in the 64th clock cycle together with the last slice. ρ and π are simple wirings executed on the 64th clock cycle as well. We can consider this one round of 64 cycles as the initialization round. For the following rounds, the input to θ is the output of the five χ functions executed in parallel on the slice R_0 followed by the XOR of the round constant. The output is taken from the output of the round constant injection starting from the first clock cycle of the 25th round. With this implementation, one iteration of KECCAK-$f[1600]$ takes $64 \times 25 = 1600$ clock cycles and costs around 10kGE in area.
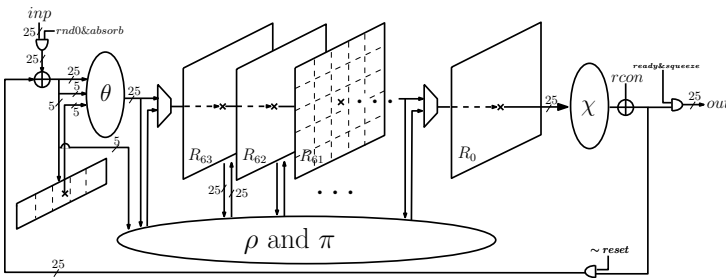
Figure 6.4: Schematic of the slice-based implementation of KECCAK-f

We note that it is possible to have implementations that operates on two or four slices per cycle to make the implementation faster by trading-off area. Here, we aim a small implementation.

Both of these unprotected implementations are noticeably smaller than the implementations reported so far which use standard cell libraries for state storage and still provide a high frequency. On the other hand, the smallest design so far, that is proposed in [82] uses RAM macros and requires more clock cycles for one iteration. More detailed comparison for after synthesis results is given in Table 6.1.

6.4.2   Threshold Implementations

We propose two different TIs for each architecture. The first one uses as few random bits as possible while operating on three shares. Namely, except for the initial sharing, we use at most four bits of fresh randomness per round as described in Section 6.2.2. In the second one, however, we relax the restriction on using minimum amount of shares and operate on four shares without the need of extra fresh randomness.

In both of these versions, we assume that the input shares are provided from an outside source, such that the sum of the shares is the unshared message and the masking satisfies Property 1. The usage of three (resp. four) shares throughout the entire implementations requires three (resp. four) times the registers compared to the unprotected implementations. The linear layers are also tripled (resp. quadrupled), such that each works on one share only. During the χ operations, these shares are used together as described in Section 6.2.2 (resp. Section 6.3). The round constant is introduced to only one share. The parallel and the serial implementations differ mainly for the masks storage of three-share implementation. The details of the re-masking and the implementations requirements are detailed as follows.

Parallel Implementation

We store the XOR of the masks $M_1^i \oplus M_2^i$ used for each row calculation in registers as shown in Figure 6.5(i) and suggested in [73] for three-share implementations. Therefore, we need 640-bit extra registers to store these mask XORs (two bits per row, 320 rows). Moreover, the output is ready one clock cycle after the last χ calculation is complete due to this re-masking. Therefore, one KECCAK-f takes 25 clock cycles.

Table 6.1: Synthesis results for different implementations of KECCAK-f

Design	State	θ	χ	Area (kGE) ANDs/XORs	Other	TOTAL	Rand. bit per round	Clock Cycles	FrEquation MHz
UMC 0.18μm standard cell library									
Parallel	9.0	9.3	7.0	8.1	0.1	33.5	-	24	572
Parallel-3sh	27.2	27.8	55.4	31.4	3.5	145.3	4	25	516
Parallel-4sh	36.3	37.1	68.8	31.9	0.1	174.2	-	24	513
Serial	10.1	0.1	0.1	0.2	0.3	10.8	-	1600	555
Serial-3sh	30.4	0.4	0.8	0.7	0.8	33.1	4	1625	553
Serial-4sh	40.5	0.6	1.0	0.7	0.3	43.1	-	1600	572
UMC 0.13μm standard cell library									
Parallel	8.0	8.6	6.4	7.5	0.1	30.6	-	24	855
Parallel-3sh	24.0	25.7	52.8	29.4	3.3	135.2	4	25	746
Parallel-4sh	32.0	34.2	61.6	29.7	0.1	157.6	-	24	735
Serial	10.0	0.1	0.1	0.2	0.2	10.6	-	1600	752
Serial-3sh	30.0	0.4	0.8	0.7	0.7	32.6	4	1625	820
Serial-4sh	40.0	0.5	0.9	0.7	0.3	42.4	-	1600	775
NANGATE 45nm standard cell library									
Parallel	9.0	6.4	5.6	7.0	0.1	28.1	-	24	690
Parallel-3sh	27.2	19.2	40.6	25.9	3.7	116.6	4	25	592
Parallel-4sh	36.3	25.6	48.7	28.7	0.1	139.4	-	24	588
Serial	12.2	0.1	0.1	0.2	0.2	12.8	-	1600	775
Serial-3sh	36.8	0.3	0.6	0.5	0.8	39.0	4	1625	645
Serial-4sh	49.0	0.4	0.8	0.6	0.3	51.1	-	1600	633
UMC 0.18μm standard cell library									
Parallel-[94]	N/A	N/A	N/A	N/A	N/A	56.7	-	25	488
STM and UMC 0.13μm standard cell library									
Parallel KECCAK team	N/A	N/A	N/A	N/A	N/A	48.0	-	24	526
Serial-[59]	N/A	N/A	N/A	N/A	N/A	20.0	-	1200	N/A
Serial-[82][1]	N/A	N/A	N/A	N/A	N/A	5.9	-	15427	61

[1]: Uses RAM macros

The costs of the combinational logics exceed the costs of the registers as expected, since there are too many instances of θ and χ (Table 6.1). Even though these implementations are fast, the parallel TIs are quite big and can no longer be called lightweight implementations, when applied to bigger versions of KECCAK.

Serial Implementation

For the three-share serialized implementation, we need one 4-bit register to keep the random bits from the previous χ function to the next (as described in Section 6.2.2) in addition to one 10-bit register (two bits per row, 5 rows) to store the mask XORs after the χ operation. Similar to the parallel implementation, this re-masking costs an extra clock cycle per round.

The register costs are the dominant costs in these architectures whereas the θ and χ layers together are only 4% of the overall implementation. The three-share TI of the serial architecture has the same size as the unprotected parallel implementation.

6.5   Using Two Shares for λ

KECCAK-f has an unavoidable 1600-bit state due to its construction. Therefore, a three or four share implementation which triples or quadruples the state automatically requires a big area. Furthermore, the cost of the linear θ layer is very close to the register cost as we converge to the parallel implementation (Table 6.1) because of multiple XORs per bit which also has a multiplied effect with sharing.

Here, we propose using two shares for the whole linear part λ to reduce the area at the cost of extra random bits. Using two shares for the linear parts implies increasing or decreasing the number of shares for the nonlinear layer since we need at least three-shares for a secure χ calculation. The re-masking from two to three shares can be done as in Fig. 6.5(ii). This expansion must be done one clock cycle before the χ computation as these three new shares need to be written to the registers to avoid leakage (Remark 3). Note, that we do not need a uniform χ implementation anymore since this expansion also serves as the re-masking the nonlinear function input (Observation 1). Therefore, we only consider the χ implementation with three shares and direct sharing in Equation (6.1). Moreover, reducing the number of shares from three to two can be done by only a single XOR as shown in Fig. 6.5(iii) since linear layers do not require uniform input shares.
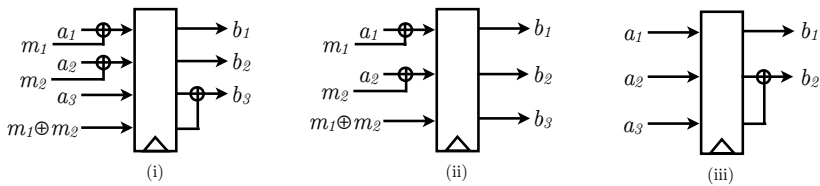

Figure 6.5: Re-masking (i) to make the masking uniform (ii) to increase the number of shares from two to three and (iii) to decrease the number of shares from three to two.

With this approach, we need 1 more clock cycle per round for the round-based architecture and 10 extra bits of randomness for each instance of the

χ function. Applying the method in a straightforward way costs 3200 bits of extra randomness. However, it is possible to use the idea of Section 6.2.2 and borrow randomness from the input of the previous instances of the χ function.

For the parallel implementation, this approach decreases the cost of the linear layer and the ANDs and XORs only. We need to put a register between the 2-to-3 re-masking and the χ layer, in order to safeguard against the possibility that some of the masks do not arrive on time. Moreover, there is the extra cost of the XORs during the re-masking to compensates for the area saved in the linear layer. In the end, such a parallel implementation does not save area and moreover it needs more randomness which is not preferable.

For the serial architecture, this approach is more efficient. When our slice-based implementation is considered, we need to increase the number of shares when we shift the data in the register R_1 to the register R_0 and decrease the number of shares during the shift from R_{63} to R_{62}. Even though the θ layer is still performed on three shares, the registers from R_1 to R_{62} only require two instances. Besides, the extra cost of re-masking is small since we only need to increase or decrease the number of shares of one slice. As a result, such an implementation requires approximately 30% less area.

6.6   Conclusion

We devoted this chapter to the first-order secure implementations of KECCAK that satisfy all the TI properties. We built these TIs on our own parallel and serial architectures which are significantly smaller than prior works using the same type of memory elements as shown in Table 6.1. At the moment, it seems that at least four shares are required in order to satisfy all TI properties simultaneously without the need of extra fresh random bits. Our efforts to fix the prior three-share TI [9], which fails to satisfy Property 4, lead to a solution using four fresh random bits per round. This number is only 0.125% of the randomness required if a straight forward re-masking would have been applied.

When KECCAK-f with wide states ($b \geq 400$) are considered, it is advantageous to implement a (pseudo) random number generator in order to use a three-share TI instead of four-share TI due to the 33.3% increase in shared state size in the latter option. In parallel implementations, each linear and nonlinear operation requires at least the area of the state which makes the parallel applications very big for practice. On the other hand, serial implementations in which the state occupies the majority of the area can be considered for near future lightweight applications. We acknowledge that when ($b \geq 400$), it is hard to consider KECCAK-f as lightweight with today's technology, however using the detailed

area analysis provided in Table 6.1, we can estimate the cost of smaller versions. Moreover, the information in the table combined with the knowledge from Chapter 4 can be used to design and estimate the requirements of higher-order secure implementations of KECCAK-f since the nonlinear layer χ resembles to the AND/XOR block of KATAN as mentioned in Section 6.2.

With this chapter, we have started the discussion of area/randomness trade-off. In addition, we provide a discussion on using two shares for the linear parts of the algorithm with additional randomness needs in order to keep the area as close as possible to the unprotected implementation. Before stating strict arguments on these matters, we will analyze similar trade-offs for the AES algorithm in the following chapter.

7

First-Order Threshold Implementations of AES

In this chapter, we investigate the first-order TI of one of the most widely used block ciphers, namely AES. A first-order TI which uses three shares throughout the algorithm is applied to AES by Moradi et al. [73]. It is based on an unprotected implementation which is presented in the same paper and known to be the smallest AES implementation so far. Therefore we also build our implementations based on this unprotected version.

Similar to KECCAK in Chapter 6, we investigate the trade-offs between circuit area and randomness requirements by changing the number of shares in different blocks of the algorithm. As mentioned in Section 2.2.4, the AES S-box has very complex coordinate functions to be implemented on hardware and typically tower field approach is used instead. We also follow the same approach with increased design flexibility. With this structure, we can vary the number of shares more frequently compared to the prior implementations in order to achieve compactness or different levels of security. Unlike the previous chapters where we did not provide a key revealing power analysis, in this chapter, we investigate the provided security of this first-order TI against high-order analysis using CPA and CEPACA described in Section 2.3.1 and 2.3.5 respectively. In addition, we question the effect of changing the number of shares in the linear layer to the security in terms of required traces.

In the following section, we first provide a very brief description of the AES algorithm and its building blocks for which the detailed information can be found in [80]. We suggest three different TIs in Section 7.2 each of which focus on a different trade-off. We used a standard tool chain to synthesize them with Faraday Standard Cell Library FSA0A__C__Generic__Core, which is based on UMC 0.18μm GenericII Logic Process with 1.8V voltage. We conclude this section by providing the performances of our designs and comparing them with the previous work in [73]. We should note that [73] uses a similar standard cell library based on UMC 0.18μm logic process with 1.8V voltage. Finally, we provide detailed analysis of these implementations in Section 7.3. We leave the extension of this work to higher-order as future work.

This work is published in [18] and [19]. The analysis in Section 7.3 is performed and mainly written by Benedikt Gierlichs. We provide it here for completeness.

7.1   Introduction to AES

AES processes 128-bit plaintext blocks in order to output 128-bit ciphertext blocks using an encryption algorithm and 128, 192 or 256-bit key. The main difference between these versions is the key schedule. Here we only focus on the version using a 128-bit key. The state can be considered as a 4×4 matrix of 8-bit elements. The plaintext is inserted to the state in the top to bottom, left to right manner.

The encryption algorithm is an SPN composed of ten rounds. The algorithm starts with the XOR of the plaintext and the secret key and repeats the following steps for each round in order.

- *SubBytes (SB)* is the nonlinear substitution layer composed of sixteen 8-bit permutations each of which is described in Section 2.2.4 and is based on multiplicative inversion in the given field.

- *ShiftRows (SR)* is a part of the linear layer that rotates the elements on row i by $i - 1$ elements to the left. Hence, the first row is not altered, second row is rotated to the left by one block etc.

- *MixColumns (MC)* completes the diffusion by mixing the elements of each column with the help of the affine transformation which is defined by a 4×4 matrix.

- *AddRoundKey (ARK)* takes the round key that went through the key-scheduling function and combines it with the state by means of an XOR.

Key schedule, which transforms the round key nonlinearly in each round, is composed of linear operations and four S-boxes, which are equal to the S-boxes in SB, substituting only four elements of the round key.

7.2 Implementation

We discuss three different TIs of AES which we refer to as raw, adjusted and nimble implementations. All implementations share the same data flow and timing. The implementations differ mostly in the S-box calculation and/or the number of shares that are used in different blocks of the algorithm. The *raw* implementation forms the basis of the other two implementations. Hence, we mainly describe the raw implementation and point out the differences with the other two. The main feature of the raw implementation is that it uses the smallest possible number of shares for each function, except the linear transformations in the S-box, provided that the shared functions are uniform. In other words, all nonlinear operations are performed with $s > 2$ shares such that the circuits are uniform and s is as small as possible. The linear operations outside the S-box are performed with two shares, whereas the linear operations in the S-box use two, three or four shares (see Sect. 7.2.2).

The *adjusted* implementation on the other hand ensures that at least three shares are used in every operation, including the linear ones. With this implementation we intend to observe the effect of moving from at least two shares to at least three shares in linear operations on the resistance against higher-order DPA, and to quantify the associated cost.

We observe that in both raw and adjusted implementations, we need to use extra fresh randomness to achieve uniformity due to parallel operations in the S-box that uses the same input as discussed in Section 3.3.2. In the *nimble* implementation the number of shares is always minimal, i.e. $s = t + 1$ where t is the degree of the unshared function ($d = 1$), even if the resulting shared function is not uniform. The uniformity of the circuit is satisfied by re-masking (Observation 1).

We first describe the general data flow of our implementations in Section 7.2.1. In Section 7.2.2 we introduce different approaches to apply the TI to the AES S-box. Finally, we provide performances of our implementations which are described by separating component functions in modules in order to satisfy non-completeness.

7.2.1 General Data Flow

We use a serial implementation for round operations and key schedule as proposed in [73] for area efficiency which requires only one S-box instance and loads the plaintext and key byte-wise in row-wise order. We also use one MixColumns instance that operates on the whole column and provides an output in one clock cycle. Due to this extreme serialization, one round requires at least 21 clock cycles even for the unprotected implementation [16]. All our TIs execute one round in 23 clock cycles. In the first 16 clock cycles, the plaintext is XORed with the key and sent to the S-box. Its output is taken from the 3rd to the 18th clock cycles and stored in the state registers, i.e. the S-box is executed in three clock cycles. The ShiftRows operation is performed in the 19th clock cycle followed by four cycles of the MixColumns calculation. The S-box takes its input from the key schedule for four cycles starting from the 18th cycle. In the 17th, 22nd and 23rd clock cycles the S-box inputs and unused random bits are set to 0. Therefore, the calculation of AES takes $23 \times 10 + 16 = 246$ clock cycles, including 16 cycles to output the ciphertext.
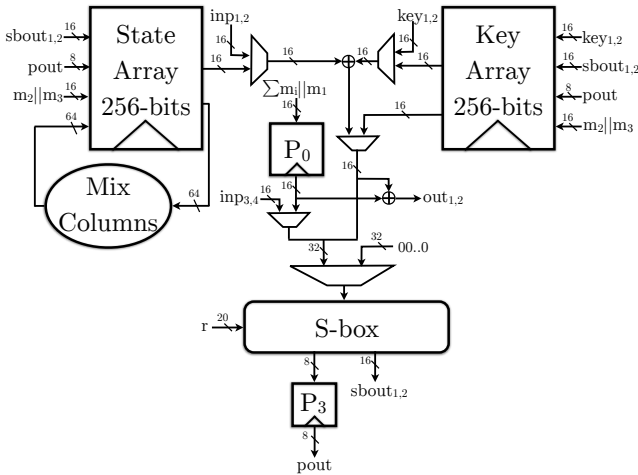
Figure 7.1: Schematic of the serialized TI of raw AES-128

Raw Implementation

We use two sets of state registers, each consisting of sixteen 16-bit registers, corresponding to the two shares of the state. MC and ARK operations are also performed with two shares. This can be seen in Figure 7.1, as the key and the state registers are 256 bits implying the two shares.

This TI of the S-box (details will be given in Section 7.2.2) requires four input shares, therefore we initially share the plaintext in four shares. We share the key in two shares and XOR them with two of the plaintext shares before the S-box operation. More details about the key scheduling will be given later in this section. Besides the shared input, the S-box needs 20-bits of randomness r. The first output share $sbout_{1,2}$ is written to the state register P_3 whereas the remaining two shares are written to register S_{33} (Fig. 7.2). The data in the state registers are shifted to the left for the following 16 cycles so that the next output of the S-box can be stored in the same registers. During this shift, the data in P_3 (*pout* in Fig. 7.1) is XORed with the second share of the S-box output, which is in the state register S_{33}, to reduce the number of shares from three to two. To achieve this, signal sig_2 is kept active from the 4th to the 19th clock cycle.
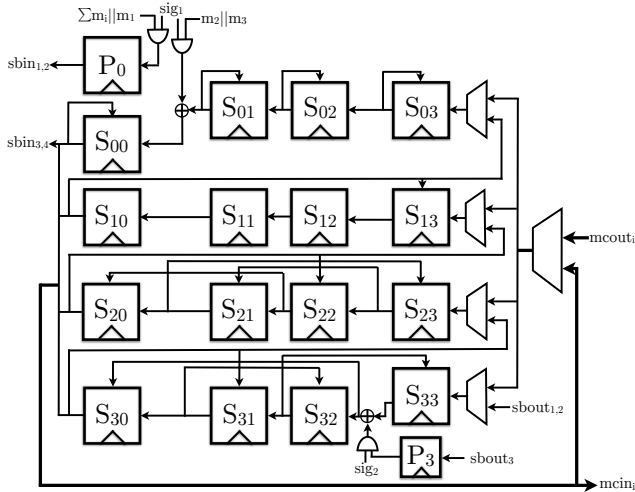
Figure 7.2: Schematic of the state array for our raw implementation where S_i, and P_0 hold two shares and P_3 holds one share; the registers P_0 and P_3 are used by the state and the key array; the XOR of the value in P_3 and S_{33} is on one share of the value in register S_{33} whereas all the other combinational operations are on two shares

The ShiftRows operation is performed in the 19th clock cycle with an irregular horizontal shift. In the next four clock cycles, the data in the registers S_{00}, S_{10}, S_{20} and S_{30} are sent to the MixColumns operation, the rest of the registers are shifted to the left horizontally and the output of the MixColumns operation is written to the registers S_{03}, S_{13}, S_{23} and S_{33}. The MixColumns operation is implemented column-wise as in [73] and with two shares working in parallel.

The registers except S_{10}, S_{11} and S_{12} are implemented as SFFs in order to operate with two inputs at reduced area cost. A single 2-to-1 MUX costs 3.33 GE and one-bit register costs 5.33 GE whereas one-bit SFF costs 6.33 GE in our library.

In the following AES rounds, we increase the number of shares of the S-box input from two to four, using 24 bits of randomness (three bytes each of which is referred to as m_i in the figures), one clock cycle before the S-box operation. To achieve this, signal sig_1 is kept active for sixteen clock cycles, starting from the last clock cycle of each round. We separate the increase of the number of shares and the nonlinear operation with registers to achieve the non-completeness property. The two additional shares are stored in P_0. The two shares in S_{00} are XORed with the two shares of the corresponding round key byte and sent to the S-box together with the two shares in P_0.

The registers P_0 and P_3 are used for both the round transformations and the key scheduling.
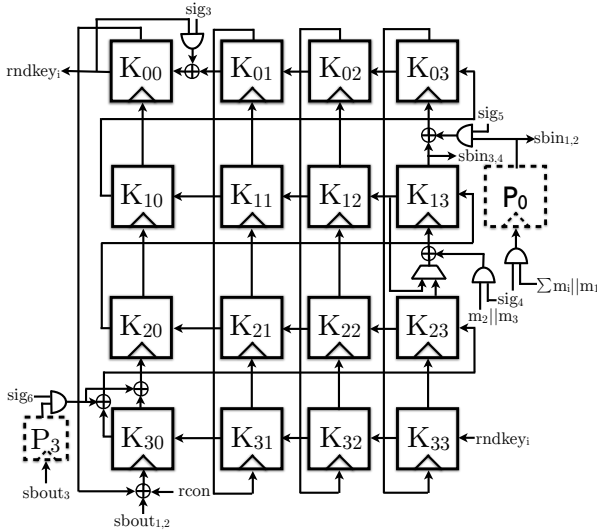
Figure 7.3: Architecture of the key arrays for our raw implementation where K_i and P_0 hold two shares and P_3 holds one share; the registers P_0 and P_3 are used by the state and the key array; the XOR of the value in P_3 and K_{30} is on one share of the value in register K_{30} whereas all the other combinational operations are on two shares

Similar to the state array, the key array also consists of sixteen 16-bit registers, implemented as SFFs, each corresponding to the two shares of a byte in the

key schedule (Fig. 7.3). The round key is inserted from the register K_{33} in the first sixteen clock cycles of each round. For the next three clock cycles, the registers except the last column (K_{03}, K_{13}, K_{23} and K_{33}) are not clocked. The registers K_{03}, K_{23} and K_{33} are also not clocked in the 17th clock cycle. In that clock cycle, we increase the number of shares in the register K_{13}. In the following three clock cycles this re-sharing is done during the vertical shift from the register K_{23} to K_{13}, i.e. the re-sharing signal sig_4 is active from the 17th to the 20th clock cycle. Signal sig_5 is active from the 18th to the 21st clock cycle to reduce the number of shares back to two. The registers K_{03}, K_{13}, K_{23} and K_{33} are not clocked in the remaining two clock cycles of each round. We choose this way of irregular clocking to avoid using extra MUXes in our design. Two shares of the S-box output are XORed to the data in K_{00} in the last four clock cycles of each round. In the 20th clock cycle the round counter *rcon* is additionally XORed to one of these shares. The number of shares is reduced back to two by XORing the share in P_3 to one of the shares in K_{30}. Signal sig_3 is active in the first sixteen clock cycles except the 4th, 8th, 12th and 16th clock cycles. The round key is taken from the register K_{00} to be XORed with the corresponding plaintext before going to the S-box operation.

Adjusted Implementation

This version works on three shares for both the state and the key schedule which increases the area significantly. The S-box still requires four input shares and outputs three shares, hence the register P_0 is reduced to 8-bits (one share) and the register P_3 is not required. Similar to the raw implementation, we use 24-bits of randomness to increase the number of shares from three to four one cycle before the S-box, i.e. each of the existing three shares is XORed with a random byte and the sum of these random bytes is taken as the fourth share. This also ensures uniformity of the S-box input. Together with the state, the number of shares for the MixColumns and the Key XOR operations increases to three.

Nimble Implementation

Similar to the raw implementation, this one also uses two shares for the state and key arrays. The main difference is that the S-box needs three input shares instead of four. Hence, the size of the register P_0 is reduced to 8-bits (one share). As a result, we need only 16-bits of randomness to increase the number of shares from two to three before the S-box operation, i.e. each share is XORed with one byte of randomness and the XOR of the random bytes is taken as the third share. The S-box requires 16-bits of extra randomness per iteration and

outputs three shares. Hence, the logic of the register P_3 to reduce the number of shares back to two stays the same.

7.2.2 TI of the AES S-box

The S-box implementations in [73] use the tower field approach up to $\mathbb{GF}(2^2)$ for a small implementation as given in Figure 2.1. Therefore, the only nonlinear operation is $\mathbb{GF}(2^2)$ multiplication which must be followed by registers and re-masking to avoid first-order leakages.

We also chose to use the tower field approach. However, we decided to go until $\mathbb{GF}(2^4)$ instead of $\mathbb{GF}(2^2)$. With this approach, the $\mathbb{GF}(2^4)$ inverter (coordinate functions provided in Appendix B.2.2) can be seen as a 4-bit permutation and the $\mathbb{GF}(2^4)$ multiplier (coordinate functions provided in Appendix B.2.1) as a 4-bit multiplication both of which are well studied in Chapter 5. Therefore, we can find uniform TIs for each of these nonlinear functions. This might allow us to reduce the number of fresh random bits needed since we will have fewer nonlinear blocks compared to [73] possibly requiring less re-masking in order to use their outputs. Moreover, with this approach the S-box calculation takes three clock cycles instead of five.

Raw implementation (Fig. 7.4)

The uniformity of each function is individually satisfied. The uniform sharing with four input and three output shares that is used to share each term in the multiplication is provided in Appendix B.2.3. For inversion, which belongs to class $\mathcal{C}_{282}^4$ (Table A.12), we consider two options. Either using four shares which is the minimum number of shares necessary for a uniform implementation in that class (Table A.6) and decomposing the function into three uniform sub-functions as $Inv(x) = F(G(H(x)))$, or using five shares without any decomposition. We remind that all 4-bit permutations including $\mathcal{C}_{282}^4$ have a uniform five-share TI as described in Section 5.1.1. Our experiments show that both versions have similar area requirements but need a different number of clock cycles. To reduce the number of cycles, we chose the version with five shares, generated by applying the formula in Appendix B.2.6 to each term of the inversion. This sharing is found by using the method described in Theorem 6 which is slightly different from the direct sharing. Namely, we found a permutation from $\mathcal{C}_{282}^4$ which has a uniform direct sharing. Then we adapted this direct sharing using the affine relation between the found permutation and the specific permutation we want to implement. We chose this sharing instead of cascaded affine-nonlinear-affine

transformation. Moreover, for our specific permutation this sharing can be implemented in hardware with less logic gates compared to the direct sharing.
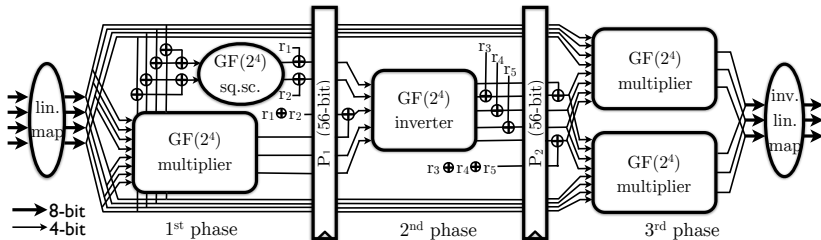
Figure 7.4: The S-box of the raw implementation

Even though it is enough to use only two shares for linear operations, we sometimes chose to work on more than two shares to avoid the need of extra random bits. The linear map of the tower-field S-box operates on four shares since the multiplication needs four input shares. The inverter requires five input shares and the multiplication outputs only three shares, therefore we use two shares for the square scalar to have five shares in the beginning of the 2nd phase. We use three shares for the inverse linear map of the tower-field S-box since the multiplication outputs three shares. For all the linear operations, the shared functions are created as instantiations of the unshared function for the first share and as unshared function without the constant term for the other shares.

During the combination of these uniform circuits, we face the challenges described at the end of Section 3.3.1 to keep the uniformity in the pipeline registers. We apply re-masking on the first pipeline register where we combine the two output shares of the square scaler and the three output shares of the multiplier to generate five shares. Note that this combination also acts as the XOR of the outputs of the square scaler and the multiplier. By Theorem 4, it is enough to re-mask only the output shares of one of the functions to achieve uniformity. We choose to re-mask the output of the square scaler since it operates on less shares, hence requires less random bits. The correction mask, i.e. the XOR of the masks, is XORed to one of the output shares of the multiplier to achieve correctness.

Another challenge is to satisfy the uniformity of the circuit as we increase or decrease the number of shares. This is achieved by introducing new masks before the S-box operation to increase from two to four shares and at the end of the 2nd phase to decrease from five to four shares. The output of the 3rd phase is not uniform when the three shares are considered together. However, we verified by simulation that each share individually is uniform, which implies that there is no first-order leakage in the following registers. We combine the

first two shares with an XOR and keep the third share as it is to go back to two shares. We also verified that, after we decrease the number of shares to two, the output shares are uniform.

We always keep the XOR of the masks in the pipeline registers and complete the re-masking in the next clock cycle as in [73] for a fair comparison. Overall, we need 44 fresh random bits per S-box operation including increasing the number of shares of the S-box input.

Adjusted implementation (Fig. 7.5)

As mentioned in the earlier sections, the only difference between the raw and the adjusted implementation is that the adjusted implementation requires at least three shares for all the blocks including the linear operations in the S-box. For that reason, the shared square scaler circuit is instantiated with three shares. This S-box also requires 44 bits of randomness per iteration.
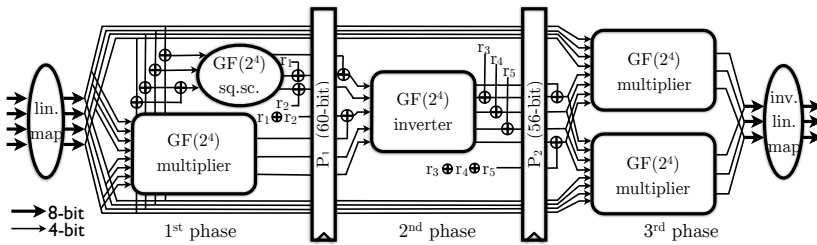
Figure 7.5: The S-box of the adjusted implementation

Nimble implementation (Fig. 7.6)

As can be observed in Figures 7.4 and 7.5, we use fresh randomness at the end of the 1st phase to satisfy uniformity during the combination of the square scaler's and the multiplier's outputs, and after the inverter to break the dependency between the inputs of the multipliers in the 3rd phase. Since these re-masking steps conserve the uniformity property and the security of each block is achieved only by the correctness and non-completeness properties (Observation 1), we can discard the uniformity property and implement these nonlinear functions with the smallest number of shares n s.t. $n > d$, i.e. $n = d + 1$, where d is the degree of the unshared functions. We use the sharing with three input and output shares provided in Appendix B.2.4 for each term of the multiplier and the sharing with four input and output shares provided in Appendix B.2.5 for

each term of the inverter. With this new construction, it is enough to have three input shares to the S-box since the multiplier block requires only three shares. We need to reduce the number of shares from five to four at the end of the 1st phase for the inverter and from four to three at the end of the 2nd phase for the following multipliers. This construction requires only 32 bits of extra randomness per S-box calculation, including increasing the number of shares for the S-box input.
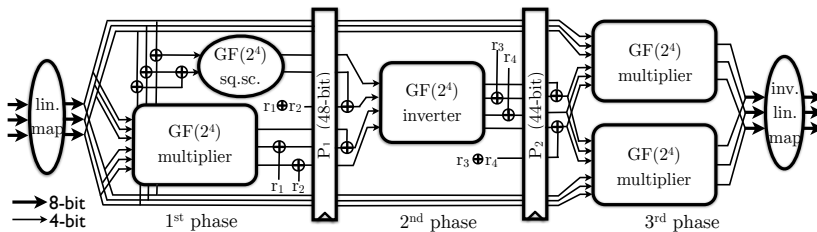
Figure 7.6: The S-box of the nimble implementation

7.2.3   Performance

Like any other DPA countermeasure, TI also allows trade-offs between area, randomness and the resistance against DPA. In Table 7.1, we provide the area costs (GE) and randomness requirements (bits) for the different S-box implementations. For all the implementations, we performed two different compilation methods. The first one is a regular compilation with the *compile* command, that does not optimize or merge modules, performed on the whole implementation. The second method on the other hand uses the *compile_ultra* command for each module to let the tool optimize each of them *individually* and combine the result. It is very important that the modules are not merged for area optimization in this step, to not violate the non-completeness property.

The total area results in Table 7.1 show that using non-uniformly shared functions as in the nimble implementation reduces the area cost significantly compared to the uniformly shared raw and adjusted implementations. This reduction is caused by the decreased number of shares used in the nonlinear blocks. Moreover, the required number of random bits per S-box also decreases together with the reduced number of shares since less shares need to be re-masked to satisfy uniformity.

In Table 7.2, we show the area, randomness requirements and timings of our AES implementations and compare them with the results in [73]. We

Table 7.1: Synthesis results for different versions of S-box TI with *compile / compile_ultra* commands

S-box	Raw	Adjusted	Nimble
Lin.Map.	168 / 120	168 / 120	126 / 90
Sq.Sc.	18	27	18
Multiplier	625 / 458	625 / 458	418 / 308
Inverter	618 / 490	618 / 490	594 / 375
Inv.Lin.Map	99 / 72	99 / 72	99 / 72
1st Ph.	919 / 704	916 / 701	646 / 500
2nd Ph.	690 / 562	702 / 574	654 / 435
3rd Ph.	1374 / 1013	1374 / 1013	959 / 713
Registers*	725	661	576
Total	3708 / 3004	3653 / 2949	2835 / 2224
Random.	44	44	32

*: including the registers P_0 and P_3

again provide our results using the same compilation techniques as the S-box implementations. The area costs for the state and the key arrays include the ANDs and XORs that are shown in Figures 7.2 and 7.3. As expected, in the raw and nimble implementations the cost of the state and key arrays together with the MixColumns are reduced by one third compared to [73] and the adjusted implementation, since we use two shares instead of three. All our versions have the same timing and use the same control module.

In our implementations, the S-box occupies 30% to 40% of the total area. Compared to the implementation in [73] our S-boxes with uniform blocks are 13% smaller and our S-box with non-uniform blocks is 33% smaller. These results show a significant area and randomness improvement for the nimble implementation, indicating that using nonuniform shared functions can be advantageous if the uniformity of the circuit is satisfied by re-masking.

7.3 Power Analysis

To evaluate the security of our designs in practice we implement them on a SASEBO-G board [3] using Xilinx ISE version 10.1. The Verilog descriptions of the designs are the same as for the ASIC evaluations, but we replaced all SFFs by DFFs and MUXes because SFFs are not available. We use the "keep hierarchy"

Table 7.2: Synthesis results for different versions of AES TI with *compile* / *compile_ultra* commands

Design	[73]	Raw	Adjusted	Nimble
State Ar.	2529	1698	2473	1687
Key Ar.	2526	1890	2762	1844
S-box	4244	3708 / 3004	3653 / 2949	2835 / 2224
MixCol.	1120	770 / 544	1156 / 816	770 / 544
Control[1]	255	242	242	242
Key XOR	64	48	72	48
MUXes	376	746	853	693
Total	11114 / 11031	9102 / 8172	11221 / 10167	8119 / 7282
Cycles	266	246	246	246
Random.[2]	48	44	44	32

[1] including round constant and other   [2] per S-box

constraint to prevent the tools from optimizing over module boundaries (see the paragraph before Sect. 3.2.1 and the last sentence before Table 7.1). Apart from that we use the standard tool chain. The board features two Xilinx Virtex-II Pro FPGA devices: we implement the TI AES and a PRNG on the crypto FPGA (xc2vp7) while the control FPGA (xc2vp30) handles I/O with the measurement computer and other equipment. The PRNG that generates all random bits is implemented as AES-128 in CTR mode.

We measure the power consumption of the crypto FPGA during the first 1.5 rounds of TI AES as the voltage drop over a 1Ω resistor in the FPGA core GND line. The output of the passive probe is sampled with a Tektronix DPO 7254C digital oscilloscope at 1GS/s sampling rate.

7.3.1 Methodology

We define two main goals for our practical evaluations. First, we want to verify our implementations' resistance against first-order attacks. But in practice adversaries are of course not restricted to applying such attacks. Therefore, our second goal is to assess the security that our implementations provide against other, e.g. higher-order, power analysis attacks.

Since there is no single, all-embracing test to evaluate the security of an implementation against key recovery attacks, we test its resistance against state-of-the-art attacks. We narrow the evaluation to univariate attacks because

our implementations process all shares of a value in parallel. Estimating the information-theoretic metric by Standaert et al. [93] is out of reach. It would require estimation of at least 2^{48} Gaussian templates.

We make several choices that are in favor of an adversary and make attacks easier. First, to minimize algorithmic noise the PRNG and the TI AES do not operate in parallel, i.e. the PRNG generates and stores a sufficient number of random bits before each TI AES operation. In practice, running them in parallel will increase the level of noise and thus the number of measurements needed for an attack to succeed. Second, we provide the crypto FPGA with a stable 3MHz clock frequency to ensure that the traces are well aligned and the power peaks of adjacent clock cycles do not overlap (this would also help to assign a possibly identified leak to a specific clock cycle). In practice, clocking the device at a faster or unstable clock will make attacks harder. Third, we let the adversary know the implementation. Specifically, if the PRNG was switched off the adversary would be able to correctly compute bit values and bit flips under the correct key hypothesis. In practice, obscurity is often used as an additional layer of security. Fourth, we use synchronous (over-)sampling [70] to avoid clock drift and achieve the best possible alignment. In practice, secure devices use an internal (and unstable) clock source which prevents synchronous sampling and increases the number of measurements needed for an attack to succeed.

7.3.2 PRNG Switched Off

To confirm that our setup works correctly and to get some reference values we first attack the implementations with the PRNG switched off. We expect that the implementations can be broken with many first-order attacks. As example, we used CPA attacks (Section 2.3.1) that use the HD of two consecutive S-box outputs as power model. The attacks require $2 \cdot 2^8$ key hypotheses. To reduce the computational complexity we let the adversary know one key byte and aim to recover the second one. The results for the raw implementation are shown in Figure 7.7

Since the adversary knows the implementation, he can choose to compute the HD over three 8-bit registers (all versions; S_{33} and P_3; output of the S-box in three shares), two 8-bit registers (raw and nimble; S_{32}; one cycle later; two shares) or ignore the details and compute the distance over a single 8-bit register as if it was a plain implementation. For all versions, only a few hundred traces are required to recover the key with any of these attacks. It is worth noting that the highest correlation peaks do not occur at the S-box output registers, but three resp. two clock cycles later when the same bit-flips occur in register S_{30}.
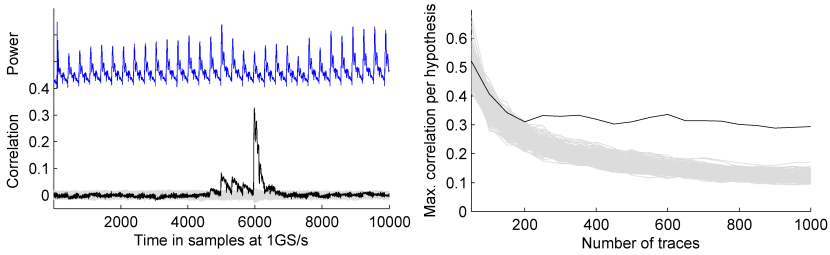
Figure 7.7: Results of CPA attacks using HD model over 3/2/1 registers with PRNG off; left: correlation traces for all key hypotheses computed using 50 000 power traces, correct hypothesis in black, and a scaled power trace; right: max. correlation coefficient per key hypothesis (from the overall time span) over number of traces used.

This register drives the MixColumns logic and therefore has a much greater fanout.

We also applied CEPACA (Section 2.3.5) that targets combinational logic. The attacks compute two sets of mean traces for the values of two processed plaintext bytes and shift the mean traces in the time domain to align them. They aim to recover the linear difference between the two key bytes involved. To do so, they permute one set of mean traces according to a hypothesis on the linear difference and then correlate both sets of mean traces. The results show that this attack is successful with a few thousand measurements for all versions. The ones regarding the raw implementation are given in Figure 7.8.
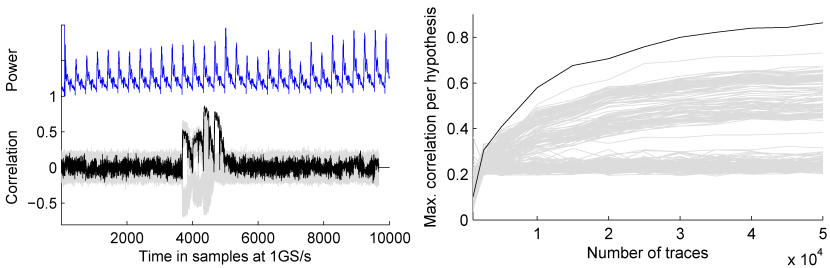
Figure 7.8: Result of CEPACA with PRNG off; left: correlation traces for all hypotheses on the linear difference computed using 50 000 power traces, correct hypothesis in black, and a scaled power trace; right: max. correlation coefficient per hypothesis on the linear difference (from the overall time span) over number of traces used.

7.3.3   PRNG Switched On

Next we repeat the evaluation with the PRNG switched on, i.e. the TI AES uses unknown and unpredictable random bits. For the CPA attacks using the HD over two or three registers as power model, we suppose these bits were zero.

Raw implementation

Figure 7.9 shows the results of the first-order attacks against the protected implementation using 10 million measurements. The results show that the attacks fail.
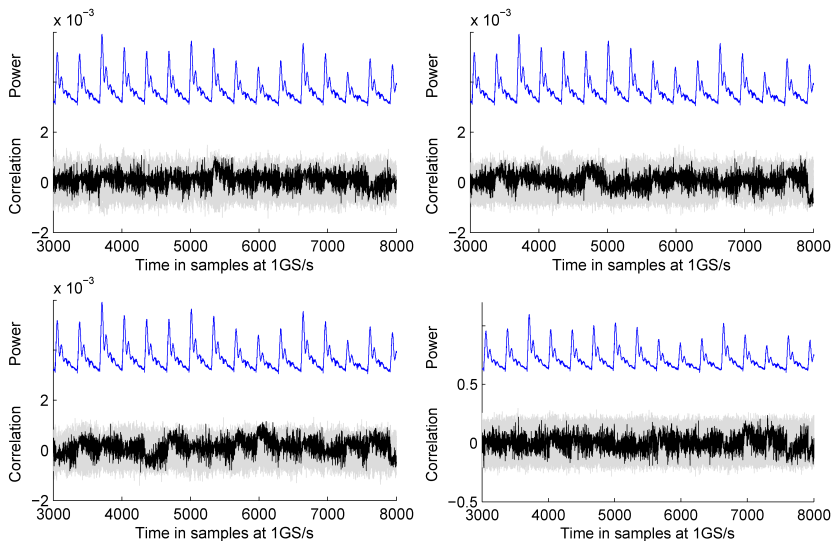
Figure 7.9: Results of first-order CPA and CEPACA on raw implementation with PRNG on computed using 10 million traces; top, left: HD over 1 register; top, right: HD over 2 registers; bottom, left: HD over 3 registers; bottom, right: CEPACA

We proceed with higher-order attacks to assess the level of security this implementation provides. For our second-order CPA attacks we use the same power models as before but center and then square the traces (for each time sample) before correlating (Section 2.3.3 [32, 85, 98]). Second-order CEPACA work as above with mean traces replaced by variance traces as described in Section 2.3.5.

Figure 7.10 (top, left) shows the results of the second-order CPA attack that uses the HD in a single register as power model (as if it was a plain implementation) using 10 million measurements. We note that the highest correlation peak occurs again when the same bitflips happen in register S_{30}, similar to when the PRNG was switched off. The attack requires about 600 000 traces to succeed, as shown in Figure 7.10 (top, right). Second-order CPA attacks using the HD over two resp. three registers as power model failed to recover the key, presumably because we do not know the masks' values and assume they are zero.

Figure 7.10 (bottom, left) shows the results of the second-order CEPACA using 10 million measurements. The attack requires about 3.5 million traces to succeed as shown in Figure 7.10 (bottom, right).

Figure 7.10: Results of second-order CPA (top) and CEPACA (bottom) on raw implementation with PRNG on computed using 10 million traces; right: min./max. correlation coefficient per hypothesis (from the overall time span) over number of traces used

Adjusted implementation

We performed the same analysis as on the raw implementation. Figure 7.11 shows that neither the first-order CPA attack that uses the HD in one register as power model nor the first-order CEPACA work with 10 million traces as expected.
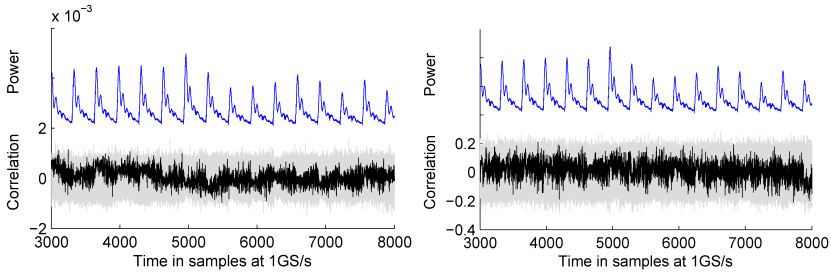
Figure 7.11: Results of first-order CPA (left) and CEPACA (right) on adjusted implementation with PRNG on computed using 10 million traces

Unlike our result for the raw implementation, we observe that second-order CPA does not work even with 10 million traces as shown in Figure 7.12 (top). This result is natural since the adjusted implementation uses three shares instead of two in register S_{33} (and the entire state array). We expect a 3rd-order CPA attack that exploits the third standardized moment of the traces to be possible, however the available 10 million traces were not enough.

Figure 7.12: Results of second-order CPA (top) and CEPACA (bottom) on adjusted implementation with PRNG on computed using 10 million traces; right: min./max. correlation coefficient per hypothesis (from the overall time span) over number of traces used

On the other hand, a second-order CEPACA still succeeds, indicating leakage from possible glitches in the S-box, as shown in Figure 7.12 (bottom, left). Recall that the adjusted implementation uses at least three shares in every

operation. Compared to Figure 7.10 (bottom, left) the second correlation peak does not show. This might be the reason why the attack becomes harder as shown in Figure 7.12 (bottom, right). It is successful with about 4 million traces but the separation of the correct key from the wrong keys is poor, even using 10 million traces. This observation indicates that the leakage that leads to the first correlation peak is almost linear and therefore harder to exploit.

Nimble implementation

We performed the same analysis as on the raw implementation and the results are similar. First-order CPA and CEPACA fail with 10 million traces. Both second-order CPA and CEPACA show peaks (Fig. 7.13, left) in the same clock cycle as for the raw implementation. They succeed with about 600 000 and 8.5 million traces, respectively, as shown in Figure 7.13 (right). However, we observe that the CEPACA requires more traces to be successful than for the the raw implementation. We suspect that this is due to the simpler component functions of the nimble implementation which cause less glitches in the circuit.

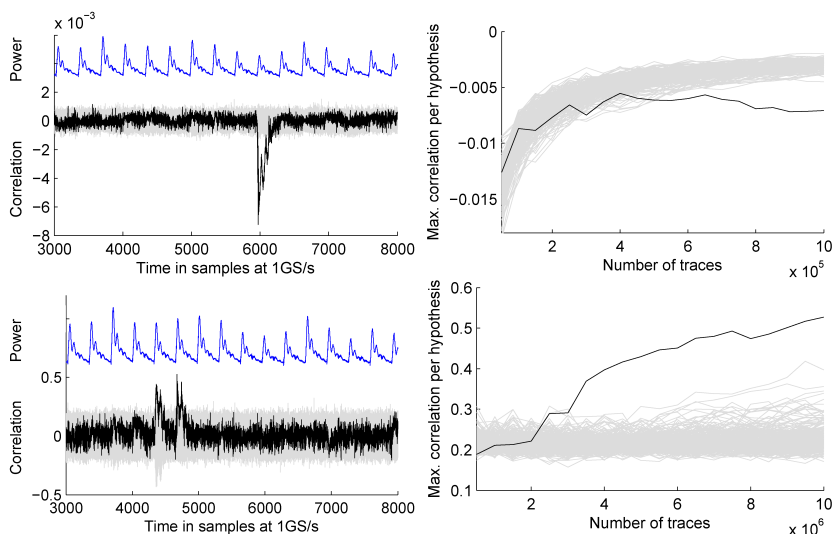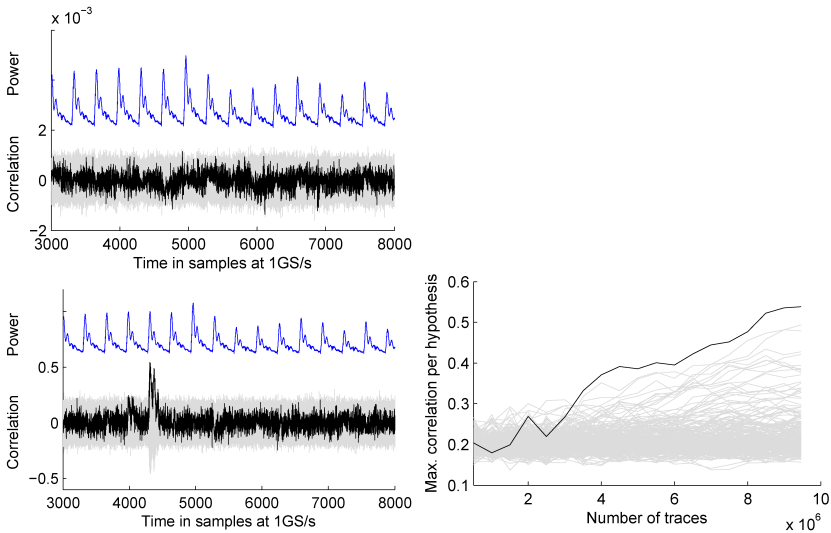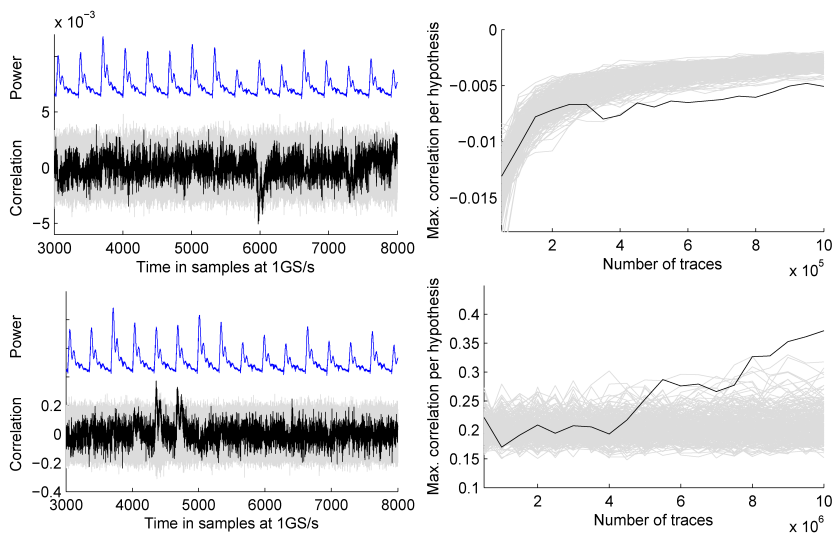Figure 7.13: Results of second-order CPA (top) and CEPACA (bottom) on nimble implementation with PRNG on computed using 1 million and 10 million traces, respectively; right: min./max. correlation coefficient per hypothesis (from the overall time span) over number of traces used

7.3.4 Discussion

The first goal of our evaluation is to verify our implementations' resistance against first-order attacks. But this goal is always limited by the number of measurements at hand. It is simply not possible to demonstrate resistance against attacks with an infinite number of traces. We have shown that our implementations resist state-of-the-art first-order attacks with 10 million traces in conditions that are strongly in favor of the adversary (no algorithmic noise from the PRNG, knowledge of the implementation, slow and stable clock, best possible alignment). Given the theoretical foundations of TI and the correctness of our implementations, we are convinced that our implementations resist first-order attacks with any number of measurements, but we have no way to demonstrate that.

The second goal of our evaluation is to assess the level of security our implementations provide against higher-order attacks and to relate the results to the area and randomness requirements. In the same adversary-friendly conditions, the most trace-efficient second-order attack in our evaluation requires about 600 000 traces for the raw and the nimble implementations. The attack exploits that the state array is in two shares, which is common to both implementations that mainly differ in the S-box implementation. Since the nimble implementation requires less resources and provides a similar level of security, it is preferable over the raw implementation.

As expected, the adjusted implementation with at least three shares in all operations provides better security than the raw implementation it is based on. The same second-order CPA attack that succeeded with 600 000 traces against the raw implementation fails against the adjusted implementation even with 10 million traces. Also a third-order CPA attack against the adjusted implementation fails with 10 million measurements. The trace requirement for a successful second-order CEPACA increases only slightly from 3.5 million to about 4 million, but the separation of the correct key from the wrong keys is much poorer. The price of this increase in security is a roughly 23% larger circuit (randomness requirements and timings are identical).

7.4 Conclusion

In this chapter, we provided first-order implementation of AES using the theory provided in Chapter 3. During our implementations, we had two main questions in mind, namely Question 2 and Question 3. In order to investigate answers, we implemented three different versions each of which have a specific purpose. The

raw implementation satisfies all TI properties in every step and uses two to five shares for different blocks of the algorithm. This implementation is faster, is 18% smaller (9 kGE) and requires less randomness compared to [73]. It is secure against first-order DPA and can only be broken with second-order DPA using 600 000 in our adversary friendly setup. Showing that changing the number of shares can be advantageous even if we need five shares in particular blocks is the main contribution of this implementation.

The adjusted implementation, which needs 11 kGE area similar to [73], trades-off area for security. The usage of at least three shares in each block of the implementation provides additional security against second-order DPA which can be profitable depending on the application.

The main goal of the nimble implementation was a small area implementation on the direction of Question 3. We initially expected to observe the area-randomness trade-off by using building blocks that fail to satisfy Property 4 but have smaller area. On the other hand, we have observed that using less shares for many of the building blocks have the side-effect of requiring less randomness for re-masking. This implementation, which benefits from Observation 1 can be considered lightweight with 8 kGE (7 kGE when optimized) and requires 28% less randomness than the raw implementation. We acknowledge that the randomness expectations can be demanding for some applications and leave further improvements as an open question. We note that this implementation provides similar security compared to raw implementation therefore is preferable.

8

Conclusion

In this chapter, we summarize the contributions of this thesis and discuss the answers to the research questions provided in Section 1.3. Then we propose directions for future research.

8.1 Summary

Increased usage of embedded devices brought the necessity of using cryptographic algorithms to provide security. New cryptographic algorithms are constantly developed in order to improve different aspects of former algorithms. The security claims of these algorithms in the black-box model need to be evaluated using several cryptanalysis techniques. Today's standardized algorithms, such as AES and SHA-3, have gone through detailed cryptanalysis which confirms their security in the black-box model with today's knowledge. However, straightforward implementations of these algorithms leak secret information due to the behavior of the device in the gray-box adversary model. In this thesis, we consider an attacker capable of performing higher-order differential power analysis (DPA), which has minimum requirements, is hardly detectable and is realistic, in order to reveal the secret.

Several countermeasures are suggested to counteract DPA. In this thesis, we considered the threshold implementation (TI) method which provides security

on various platforms with minimal assumptions. In addition, it does not require a cell level investigation and is compatible with standard design flows and libraries.

Originally, TI was proposed to provide security against first-order DPA. In this thesis, we provide the theory of dth-order TI that counteracts d^{th}-order DPA which was proposed as the first research question in Section 1.3. Such a TI relies on four properties; namely uniform masking, correctness, d^{th}-order non-completeness and uniform sharing of a function. We explained the particular contributions of each property to the claimed security and discussed several methods to achieve them. Specifically, uniform masking brings a proper randomization of intermediate values whereas correctness is required for a valid implementation.

Satisfying d^{th}-order non-completeness is especially important to provide security of a glitchy circuit. We showed that this property can be satisfied by using at least $d + 1$ input shares. However, minimizing the number of input shares might blow up the number of output shares and must be applied with extreme care. On the other hand, there always exists a sharing of a function with algebraic degree t using $s_{\text{in}} = t \times d + 1$ input and $s_{\text{out}} = \binom{s_{\text{in}}}{t}$ output shares which can be applied with less considerations.

Uniform sharing of a function is required when the design comprises of functions of which the outputs together become the input to a nonlinear function or cascaded operations in the design. Our priority was to satisfy the first three properties and then work on the sharing in order to satisfy its uniformity. This becomes challenging on nonlinear functions. We investigated how to achieve all these properties for an AND/XOR gate; all 3×3 and 4×4 permutations; 5×5 AB and 6×6 APN permutations; and KECCAK and AES S-boxes.

During our investigation, we always started with a directly shared nonlinear function where each shared term is assigned to a specific component function such that the sharing satisfies correctness and d^{th}-order non-completeness. If given a uniform masking this shared function fails to satisfy uniformity, we used one or more of the following approaches to solve the problem: shuffling the terms in each component function by using correction terms, splitting the unshared function using decomposition or tower field approach and increasing the number of shares. Re-masking was applied in order to satisfy all the properties for the rare occasions where non of the mentioned methods revealed a solution or was feasible. We leave as an open question if more performant solutions exist and can be found efficiently. For KATAN, KECCAK and AES we extended our research from the nonlinear layer to the full block cipher implementation.

We always tested our implementations using simulated traces. We acknowledge

that simulations do not necessarily represent the exact behavior of the device hence, can be misleading. Therefore, for KATAN and AES implementations, we also performed analysis using the power traces gathered from an FPGA implementation. Several DPA confirm the claimed security of our implementations. We note that this evaluation process must be performed for all the suggested countermeasures since it can reveal inaccurate assumptions on the behavior of the device.
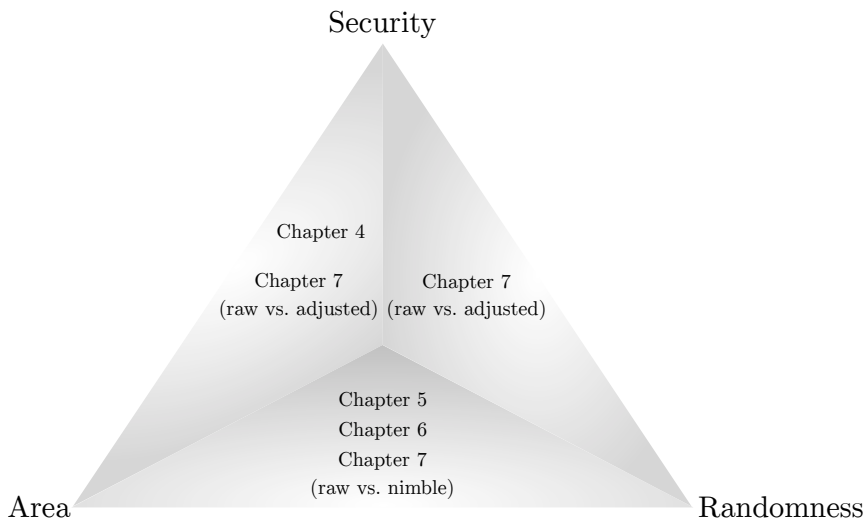
Figure 8.1: Overview of the trade-offs considered in this thesis

Minimizing the extra resource requirements arising from having a counter-measure is suggested as a research question in Section 1.3. We changed the number of shares in order to find the implementation with minimum resource requirements which produced several trade-offs answering the final research question concerning to area-randomness-security trade-offs. In Figure 8.1, we summarize which chapters focus on which trade-offs. Precisely, in Chapter 4, we observed the increase in area with increased security using up to third-order TIs of the block cipher KATAN. In Chapters 5 and 6, we fixed the security to first-order then investigated the area-randomness trade-off. In Chapter 5, we observed that decomposing a nonlinear function in order to use TI with less shares or to reduce the additional re-masking can have an area overhead due to the additional pipelined registers. On the other hand, we showed in Chapter 6 that the cost of re-masking can be reduced significantly using algorithm dependent optimizations. In Chapter 7, we first described a raw first-order TI of AES. We then increased the number of shares in order to increase the

security against second-order DPA. This adjusted implementation showed the area-security and randomness-security trade-offs. Finally, we omit Property 4 and provided security using re-masking to observe the area-randomness trade-off. This smaller nimble implementation requires less additional random bits for re-masking compared to the adjusted implementation caused by the decreased number of shares.

A somewhat surprising result of our work is that that using less shares does not necessarily imply smaller implementations. Similarly, using more shares and/or uniformly shared functions do not imply requiring less random bits in addition to the initial masking. Using two shares for linear layers and changing the number of shares when necessary should be considered as an option to decrease the area which sometimes trades-off security. It is possible to re-use the randomness in certain cases in order to decrease the need of additional random bits. However, this method should be applied with extreme care. The trade-offs differ for each algorithm and application therefore should be carefully examined for each particular implementation.

To conclude, we introduced a countermeasure against higher-order DPA which provides provable security on a wide range of platforms including the ones where glitches occur. A serial implementation of a cryptographic algorithm which provides security against first-order DPA following this methodology requires approximately three times the area of an unprotected implementation with negligible time increase. Precisely, the smallest first-order TIs of the standardized AES and SHA-3 algorithms need approximately 8kGE and 30kGE respectively. Moreover, the increase of area in order to implement a function securely is approximated to depend linearly on the order of security. The moderate increase in resources combined with the provided security and a detailed investigation on different functions and trade-offs makes this technique a valuable candidate to be used in practice.

8.2 Directions for Future Research

In addition to the open questions provided in different chapters of this dissertation, our main considerations for future research are as follows.

Analyzing TI on Different Platforms

Embedded systems can be constructed using ASICs (application-specific integrated circuit) or FPGAs each of which has its advantages. FPGAs are more

expensive; however, their reconfigurability is advantageous. On the other hand, ASICs which are relatively cheap per piece when a large number of them are taped-out are unchangeable. Hence, a mistake or an unforeseen weakness can cause loss of millions. Therefore, there are several steps to follow before an ASIC tape-out with a cryptographic algorithm that claims DPA resistance. Namely, the claimed security should be confirmed initially by the simulated traces followed by traces gathered from an FPGA implementation before investing a lot of resources on an ASIC implementation. In this dissertation, we stopped after confirming security of our implementations on FPGA.

Moreover, we mainly considered hardware implementations since TI differs from other masking schemes by providing security even on glitchy circuits, leaving the software implementations uncovered. On the other hand, implementations on co-designed systems which require hardware and software interactions or high-end devices also require security against DPA.

Even though our security claims *should* be reflected to a well designed ASIC or a software implementation, such designs might come with their own challenges. Completing the investigation of the claimed security and the resource requirements of TI on different platforms increases the applicability of the TI method and closes the gap between research and industry.

Changing the Leakage Model

TI assumes that the total leakage of the device is a linear combination of leakages gathered from different operations. Even if there is cross-talk between wires (shares), it is assumed that leakages from these shares are not combined nonlinearly. This is a common assumption for low-end CMOS technologies. That is why we used SASEBO-G board with Virtex-II Pro FPGAs on board, which is known to leak in that model, in our experimental setup.

The semiconductor industry is decreasing the process size and increasing the performance exponentially following Moore's law. The smaller process sizes such as 22 or 14nm processes bring a higher possibility of cross-talk between wires. Experiments on these high-end platforms which will often be used in the near future brings us one more step closer to future-proof implementations.

Strengthening the Adversary Model

As mentioned in Section 1.2, our ultimate goal is to have a countermeasure that is secure against all known (and possibly unknown) attacks. In this dissertation, we limit our adversary to perform passive-noninvasive d^{th}-order DPA. This DPA

adversary model can be extended by considering mutual information analysis. Observing the strength that TI provides against this adversary when higher-order implementations are considered completes the investigation of security against DPA.

It has been shown that the higher-order DPA model has a direct correspondence with the passive-(semi-)invasive d^{th}-order probing model. However, neither active adversaries nor combined adversaries are considered so far in TI context. Improving TI such that it is secure against the mentioned stronger adversaries completes a bigger portion of the ultimate goal. Finding inspiration from multi-party computation protocols which deal with faulty results could be the starting point.

A

Tables

A.1 3-**bit Permutations**

Table A.1: The 4 classes of 3-bit permutations

Class	Truth table	Sharing
$\mathcal{A}_0^3$	01234567	1,1
$\mathcal{Q}_1^3$	01234576	1,1
$\mathcal{Q}_2^3$	01234675	1,1
$\mathcal{Q}_3^3$	01243675	2,2

A.2 4-**bit Permutations**

Table A.2: The 302 classes of 4-bit permutations

Class	Truth table	Sh.	Class	Truth table	Sh.
$\mathcal{A}_0^4$	0123456789ABCDEF	1,1	$\mathcal{C}_5^4$	0123456789ACDBFE	-,2
$\mathcal{C}_1^4$	0123456789ABCDFE	-,1	$\mathcal{C}_6^4$	0123456789ACBDFE	3,3
$\mathcal{C}_2^4$	0123456789ABCEFD	3,3	$\mathcal{C}_7^4$	0123456789ACBEFD	-,3
$\mathcal{C}_3^4$	0123456789ABDEFC	-,1	$\mathcal{C}_8^4$	0123456789ACDEFB	3,3
$\mathcal{Q}_4^4$	0123456789ABDCFE	1,1	$\mathcal{C}_9^4$	0123456789ACDEBF	-,3

Table A.3: The 302 classes of 4-bit permutations cont'd

Class	Truth table	Sh.	Class	Truth table	Sh.
$\mathcal{C}^4_{10}$	0123456789BCAEFD	3,3	$\mathcal{C}^4_{48}$	012345786AC9EDFB	-,3
$\mathcal{C}^4_{11}$	0123456789BCEFDA	-,2	$\mathcal{C}^4_{49}$	012345786A9CDEBF	3,3
$\mathcal{Q}^4_{12}$	0123456789CDEFAB	1,1	$\mathcal{C}^4_{50}$	012345786A9CFDBE	3,3
$\mathcal{C}^4_{13}$	0123456789CDEFBA	-,1	$\mathcal{C}^4_{51}$	012345786ABCDE9F	-,3
$\mathcal{C}^4_{14}$	0123456879CDEFBA	3,3	$\mathcal{C}^4_{52}$	012345786ACBDE9F	3,3
$\mathcal{C}^4_{15}$	012345687A9CBEFD	-,3	$\mathcal{C}^4_{53}$	012345786ACBDFE9	3,3
$\mathcal{C}^4_{16}$	012345687A9CDFBE	3,3	$\mathcal{C}^4_{54}$	012345786A9BCEFD	-,2
$\mathcal{C}^4_{17}$	0123456879CDEFAB	-,2	$\mathcal{C}^4_{55}$	012345786AB9CFDE	3,3
$\mathcal{C}^4_{18}$	0123456879ACDBFE	3,3	$\mathcal{C}^4_{56}$	012345786AC9BFDE	-,3
$\mathcal{C}^4_{19}$	0123456879ACDFBE	-,3	$\mathcal{C}^4_{57}$	012345786A9CBEFD	3,3
$\mathcal{C}^4_{20}$	0123456879ACDEBF	3,3	$\mathcal{C}^4_{58}$	012345786ACFDE9B	-,3
$\mathcal{C}^4_{21}$	0123456879ACBDFE	-,3	$\mathcal{C}^4_{59}$	012345786ACEDFB9	-,2
$\mathcal{C}^4_{22}$	0123456879ACFEDB	3,3	$\mathcal{C}^4_{60}$	012345786ACFB9DE	3,3
$\mathcal{C}^4_{23}$	0123456879BCEFAD	-,3	$\mathcal{C}^4_{61}$	012345786ACFDEB9	3,3
$\mathcal{C}^4_{24}$	012345687A9CFBDE	4,3	$\mathcal{C}^4_{62}$	012345786A9CBFED	-,3
$\mathcal{C}^4_{25}$	0123456879ABCEFD	-,3	$\mathcal{C}^4_{63}$	012345786AC9DEFB	3,3
$\mathcal{C}^4_{26}$	0123456879BCDEFA	3,3	$\mathcal{C}^4_{64}$	012345786ABCED9F	3,3
$\mathcal{C}^4_{27}$	012345687ABCDEF9	-,3	$\mathcal{C}^4_{65}$	012345786A9CFDEB	-,3
$\mathcal{C}^4_{28}$	0123456879BCEAFD	3,3	$\mathcal{C}^4_{66}$	012345786ACB9EFD	3,3
$\mathcal{C}^4_{29}$	012345687ABCEFD9	-,3	$\mathcal{C}^4_{67}$	012345786ACF9DBE	3,3
$\mathcal{C}^4_{30}$	012345687ABCE9FD	-,3	$\mathcal{C}^4_{68}$	0123457869ACDFEB	-,3
$\mathcal{C}^4_{31}$	0123456879ACBEFD	3,3	$\mathcal{C}^4_{69}$	0123457869ACDEBF	-,3
$\mathcal{C}^4_{32}$	0123456879ACFBDE	-,3	$\mathcal{C}^4_{70}$	012345786ACBF9ED	3,3
$\mathcal{C}^4_{33}$	0123456879BCEFDA	3,3	$\mathcal{C}^4_{71}$	012345786ACEBD9F	3,3
$\mathcal{C}^4_{34}$	0123456879BCFEAD	3,3	$\mathcal{C}^4_{72}$	012345786ACDF9EB	-,3
$\mathcal{C}^4_{35}$	0123456879CEAFDB	-,3	$\mathcal{C}^4_{73}$	012345786ACDF9BE	3,3
$\mathcal{C}^4_{36}$	0123456879CEAFBD	3,3	$\mathcal{C}^4_{74}$	012345786ACDE9FB	3,3
$\mathcal{C}^4_{37}$	0123456879ACDEFB	-,3	$\mathcal{C}^4_{75}$	012345786AC9FBED	-,3
$\mathcal{C}^4_{38}$	0123456879ABDEFC	3,3	$\mathcal{C}^4_{76}$	012345786ACEBFD9	3,3
$\mathcal{C}^4_{39}$	012345768A9CBEFD	-,3	$\mathcal{C}^4_{77}$	012345786A9CEFDB	-,3
$\mathcal{C}^4_{40}$	012345768A9CBFDE	-,2	$\mathcal{C}^4_{78}$	0123457869ACBEDF	3,3
$\mathcal{C}^4_{41}$	012345768A9CBFED	3,3	$\mathcal{C}^4_{79}$	0123457869ACBFDE	-,3
$\mathcal{C}^4_{42}$	012345786ACBED9F	-,3	$\mathcal{C}^4_{80}$	0123457869ACBEFD	-,3
$\mathcal{C}^4_{43}$	012345786ABCF9DE	3,3	$\mathcal{C}^4_{81}$	0123457869ACEFDB	3,3
$\mathcal{C}^4_{44}$	012345786AC9BFED	3,3	$\mathcal{C}^4_{82}$	0123457869ACEBDF	-,3
$\mathcal{C}^4_{45}$	012345786A9CFBDE	-,3	$\mathcal{C}^4_{83}$	0123457869ACEBFD	3,3
$\mathcal{C}^4_{46}$	012345786ABCDEF9	3,3	$\mathcal{C}^4_{84}$	012345786ACF9EBD	-,3
$\mathcal{C}^4_{47}$	012345786AC9DEBF	-,3	$\mathcal{C}^4_{85}$	012345786A9CEBDF	3,3

Table A.4: The 302 classes of 4-bit permutations cont'd

Class	Truth table	Sh.	Class	Truth table	Sh.
$\mathcal{C}^4_{86}$	012345786A9CFBED	3,3	$\mathcal{C}^4_{124}$	0123458A69CEBDF7	3,3
$\mathcal{C}^4_{87}$	012345786ACD9EFB	-,3	$\mathcal{C}^4_{125}$	0123458A69CB7EFD	-,3
$\mathcal{C}^4_{88}$	012345786ACD9FBE	-,2	$\mathcal{C}^4_{126}$	012345786AC9EDBF	3,3
$\mathcal{C}^4_{89}$	012345786ACD9EBF	3,3	$\mathcal{C}^4_{127}$	012345786ABC9FED	3,3
$\mathcal{C}^4_{90}$	012345786ABCF9ED	-,3	$\mathcal{C}^4_{128}$	0123458A6B9CDE7F	-,2
$\mathcal{C}^4_{91}$	012345786ACFBD9E	-,3	$\mathcal{C}^4_{129}$	0123458A6BC7F9ED	-,3
$\mathcal{C}^4_{92}$	012345786ABC9EDF	3,3	$\mathcal{C}^4_{130}$	0123458A6CBDE79F	3,2
$\mathcal{C}^4_{93}$	012345786ABC9EFD	-,3	$\mathcal{C}^4_{131}$	0123458A6CE9BDF7	3,2
$\mathcal{C}^4_{94}$	012345786ACED9FB	-,3	$\mathcal{C}^4_{132}$	0123458A6CBD7E9F	-,3
$\mathcal{C}^4_{95}$	012345786A9CDFEB	3,3	$\mathcal{C}^4_{133}$	0123458A6C9FBD7E	-,3
$\mathcal{C}^4_{96}$	012345786A9CEDFB	3,3	$\mathcal{C}^4_{134}$	0123458A69C7DEBF	3,3
$\mathcal{C}^4_{97}$	0123458A6BCEDF97	-,3	$\mathcal{C}^4_{135}$	0123458A69CDE7FB	-,3
$\mathcal{C}^4_{98}$	0123458A6BCF97ED	-,3	$\mathcal{C}^4_{136}$	0123458A69C7FBED	3,3
$\mathcal{C}^4_{99}$	0123458A6BC97FDE	3,3	$\mathcal{C}^4_{137}$	0123458967CEAFBD	-,3
$\mathcal{C}^4_{100}$	0123458A6B9CF7ED	-,3	$\mathcal{C}^4_{138}$	0123458967CEAFDB	3,3
$\mathcal{C}^4_{101}$	0123458A6BCFED79	3,3	$\mathcal{C}^4_{139}$	0123456879BCAEFD	-,3
$\mathcal{C}^4_{102}$	012345786A9CDBEF	-,3	$\mathcal{C}^4_{140}$	012345687ABC9FDE	3,3
$\mathcal{C}^4_{103}$	0123458A69C7DFEB	3,3	$\mathcal{C}^4_{141}$	0123458967CEBFDA	-,3
$\mathcal{C}^4_{104}$	0123458A69C7FDBE	3,3	$\mathcal{C}^4_{142}$	012345786ACD9FEB	3,3
$\mathcal{C}^4_{105}$	0123458A697CBEFD	-,3	$\mathcal{C}^4_{143}$	0123458A69CFB7DE	-,3
$\mathcal{C}^4_{106}$	0123458A697CBFDE	-,3	$\mathcal{C}^4_{144}$	0123458A69CFDEB7	-,3
$\mathcal{C}^4_{107}$	0123458A69CE7FDB	3,3	$\mathcal{C}^4_{145}$	0123458A69BCF7ED	3,3
$\mathcal{C}^4_{108}$	0123458A6C9FEB7D	-,2	$\mathcal{C}^4_{146}$	0123458A69CB7FDE	-,3
$\mathcal{C}^4_{109}$	0123458A6CB9F7ED	-,3	$\mathcal{C}^4_{147}$	012345786ABCFDE9	3,3
$\mathcal{C}^4_{110}$	0123458A69CFD7BE	3,3	$\mathcal{C}^4_{148}$	012345786ABCE9FD	3,3
$\mathcal{C}^4_{111}$	0123458A69BC7FDE	3,3	$\mathcal{C}^4_{149}$	012345786ABCFD9E	-,3
$\mathcal{C}^4_{112}$	0123458A6C7EBFD9	-,3	$\mathcal{C}^4_{150}$	0123458A6BCFDE97	2,2
$\mathcal{C}^4_{113}$	0123458A6C7FBE9D	-,3	$\mathcal{C}^4_{151}$	0123458A6BCF97DE	2,2
$\mathcal{C}^4_{114}$	012345786ACFBDE9	3,3	$\mathcal{C}^4_{152}$	0123458A6BCF7E9D	-,3
$\mathcal{C}^4_{115}$	012345786ACBE9DF	3,3	$\mathcal{C}^4_{153}$	0123458A6B9CEDF7	-,3
$\mathcal{C}^4_{116}$	0123458A6C9D7FBE	-,2	$\mathcal{C}^4_{154}$	0123467859CFBEAD	3,3
$\mathcal{C}^4_{117}$	0123458A6C9D7EFB	-,3	$\mathcal{C}^4_{155}$	0123467859CFEBDA	3,3
$\mathcal{C}^4_{118}$	0123458A6C9FDB7E	3,3	$\mathcal{C}^4_{156}$	0123458A69CFE7BD	-,3
$\mathcal{C}^4_{119}$	012345786ACB9FED	-,3	$\mathcal{C}^4_{157}$	0123458A69CEFB7D	-,3
$\mathcal{C}^4_{120}$	0123458A6C7EBDF9	3,3	$\mathcal{C}^4_{158}$	0123458A6BCF7D9E	2,2
$\mathcal{C}^4_{121}$	0123458A6C7FBD9E	3,3	$\mathcal{C}^4_{159}$	0123458A6BCED79F	2,2
$\mathcal{C}^4_{122}$	0123458A6BCE79FD	-,3	$\mathcal{C}^4_{160}$	0123468B59CED7AF	-,3
$\mathcal{C}^4_{123}$	0123458A69BCE7DF	3,3	$\mathcal{C}^4_{161}$	0123458A6B7CEDF9	3,3

Table A.5: The 302 classes of 4-bit permutations cont'd

Class	Truth table	Sh.	Class	Truth table	Sh.
$\mathcal{C}^4_{162}$	0123458A6B7CDFE9	3,3	$\mathcal{C}^4_{200}$	0123458A6BCFD79E	-,3
$\mathcal{C}^4_{163}$	0123468C59BDE7AF	-,3	$\mathcal{C}^4_{201}$	012345786ACB9FDE	3,3
$\mathcal{C}^4_{164}$	0123458A6B7C9FDE	3,3	$\mathcal{C}^4_{202}$	012345786ACE9DFB	3,3
$\mathcal{C}^4_{165}$	0123458A6B7C9EFD	3,3	$\mathcal{C}^4_{203}$	012345786ACF9BDE	-,3
$\mathcal{C}^4_{166}$	012345896ABCE7DF	-,2	$\mathcal{C}^4_{204}$	012345786ACE9BFD	-,2
$\mathcal{C}^4_{167}$	0123458A67BC9EFD	-,3	$\mathcal{C}^4_{205}$	012345786ACDB9EF	3,3
$\mathcal{C}^4_{168}$	0123458A6CBFE7D9	2,2	$\mathcal{C}^4_{206}$	012345896ABCEDF7	-,3
$\mathcal{C}^4_{169}$	012345786ACFB9ED	-,3	$\mathcal{C}^4_{207}$	0123458A67BCEDF9	-,3
$\mathcal{C}^4_{170}$	012345786ACEB9DF	-,2	$\mathcal{C}^4_{208}$	0123458A69C7BFDE	3,3
$\mathcal{C}^4_{171}$	0123458A6CBF7E9D	2,2	$\mathcal{C}^4_{209}$	0123468B59CF7DAE	-,3
$\mathcal{C}^4_{172}$	0123458A6C9DBF7E	2,2	$\mathcal{C}^4_{210}$	0123468A5BCF7D9E	-,3
$\mathcal{C}^4_{173}$	012345786A9CBDFE	-,3	$\mathcal{C}^4_{211}$	0123458A69CED7FB	3,3
$\mathcal{C}^4_{174}$	0123458A69CF7EBD	3,3	$\mathcal{C}^4_{212}$	0123458A69BC7EFD	3,3
$\mathcal{C}^4_{175}$	012345786ACDE9BF	-,3	$\mathcal{C}^4_{213}$	012345896ABC7EFD	-,2
$\mathcal{C}^4_{176}$	0123457869ACFEBD	3,3	$\mathcal{C}^4_{214}$	0123458A67CEB9FD	2,2
$\mathcal{C}^4_{177}$	0123457869BCEAFD	-,3	$\mathcal{C}^4_{215}$	012345896ACEB7FD	2,2
$\mathcal{C}^4_{178}$	0123458A6C7DBFE9	3,3	$\mathcal{C}^4_{216}$	0123457869CDEFBA	-,2
$\mathcal{C}^4_{179}$	012345786A9CEDBF	-,3	$\mathcal{C}^4_{217}$	012345687ABC9EFD	3,3
$\mathcal{C}^4_{180}$	0123458A6C9D7FEB	3,3	$\mathcal{C}^4_{218}$	0123457869BCDEFA	-,3
$\mathcal{C}^4_{181}$	012345896ABC7FDE	-,3	$\mathcal{C}^4_{219}$	012345786ACF9BED	3,3
$\mathcal{C}^4_{182}$	0123458A67BC9FDE	-,3	$\mathcal{C}^4_{220}$	0123468A59CFDE7B	-,3
$\mathcal{C}^4_{183}$	012345896ACF7BED	3,3	$\mathcal{C}^4_{221}$	0123457869CEAFDB	3,3
$\mathcal{C}^4_{184}$	0123458A67CF9BED	3,3	$\mathcal{C}^4_{222}$	0123467859CFEADB	-,3
$\mathcal{C}^4_{185}$	012345896ACE7BFD	-,3	$\mathcal{C}^4_{223}$	0123468A5BCFDE79	2,2
$\mathcal{C}^4_{186}$	0123458A67CF9BDE	-,3	$\mathcal{C}^4_{224}$	0123457869CEBFDA	-,3
$\mathcal{C}^4_{187}$	012345786ACEFB9D	3,3	$\mathcal{C}^4_{225}$	0123456879CEBFDA	3,3
$\mathcal{C}^4_{188}$	012345786ACFEB9D	-,3	$\mathcal{C}^4_{226}$	012345786ABC9FDE	-,3
$\mathcal{C}^4_{189}$	0123457869CEFBDA	3,3	$\mathcal{C}^4_{227}$	012345786ACFD9BE	-,3
$\mathcal{C}^4_{190}$	0123458A6C7DBEF9	-,3	$\mathcal{C}^4_{228}$	0123458A69BCEDF7	3,3
$\mathcal{C}^4_{191}$	0123458A6C7FB9DE	-,3	$\mathcal{C}^4_{229}$	0123458A6C9DBFE7	-,3
$\mathcal{C}^4_{192}$	0123458A6C7FBED9	3,3	$\mathcal{C}^4_{230}$	0123458A6CEB7FD9	-,3
$\mathcal{C}^4_{193}$	0123458A6C7FDB9E	-,3	$\mathcal{C}^4_{231}$	0123468B59CEDA7F	3,3
$\mathcal{C}^4_{194}$	012345786ACFED9B	3,3	$\mathcal{C}^4_{232}$	0123458A6C9FDBE7	-,3
$\mathcal{C}^4_{195}$	0123458A6BC7DE9F	-,3	$\mathcal{C}^4_{233}$	0123458A67B9CFDE	2,2
$\mathcal{C}^4_{196}$	0123468C59BDEA7F	3,3	$\mathcal{C}^4_{234}$	012345896AB7CFDE	2,2
$\mathcal{C}^4_{197}$	0123458A6CBDE97F	-,3	$\mathcal{C}^4_{235}$	0123458A69B7CEFD	-,3
$\mathcal{C}^4_{198}$	0123458A69C7BEFD	3,3	$\mathcal{C}^4_{236}$	0123458A6B97CFDE	2,2
$\mathcal{C}^4_{199}$	0123458A6BCFD9E7	-,2	$\mathcal{C}^4_{237}$	0123458A69B7CFDE	-,3

Table A.6: The 302 classes of 4-bit permutations cont'd

Class	Truth table	Sh.	Class	Truth table	Sh.
$\mathcal{C}^4_{238}$	0123457689CEAFBD	2,2	$\mathcal{C}^4_{270}$	0123468B5C9DEA7F	3,3
$\mathcal{C}^4_{239}$	0123457689CEAFDB	-,3	$\mathcal{C}^4_{271}$	0123468B5C9DAFE7	-,3
$\mathcal{C}^4_{240}$	012345768A9CDEFB	3,3	$\mathcal{C}^4_{272}$	0123468B5CD79FAE	-,3
$\mathcal{C}^4_{241}$	012345768A9CDEBF	-,2	$\mathcal{C}^4_{273}$	0123458A6C7FEB9D	3,3
$\mathcal{C}^4_{242}$	012345768A9CDFEB	-,3	$\mathcal{C}^4_{274}$	0123458A6BCED97F	-,3
$\mathcal{C}^4_{243}$	012345768ACF9BDE	2,2	$\mathcal{C}^4_{275}$	0123458A6CF7BE9D	3,3
$\mathcal{C}^4_{244}$	012345768ACE9BFD	2,2	$\mathcal{C}^4_{276}$	0123458A6CF7BD9E	-,3
$\mathcal{C}^4_{245}$	012345768ACF9BED	-,3	$\mathcal{C}^4_{277}$	0123458A6BC9DE7F	3,3
$\mathcal{C}^4_{246}$	0123456879BAEFDC	-,2	$\mathcal{C}^4_{278}$	0123468B5CD7AF9E	3,3
$\mathcal{C}^4_{247}$	012345687AB9DEFC	3,3	$\mathcal{C}^4_{279}$	0123458A6BC7DFE9	-,3
$\mathcal{C}^4_{248}$	0123456879CEFBDA	-,2	$\mathcal{C}^4_{280}$	0123457869ACEDBF	3,3
$\mathcal{C}^4_{249}$	0123458A69CFEB7D	3,3	$\mathcal{C}^4_{281}$	0123457869ACFBDE	3,3
$\mathcal{C}^4_{250}$	0123458A69CD7FEB	-,3	$\mathcal{C}^4_{282}$	0123468B5CD7F9EA	-,3
$\mathcal{C}^4_{251}$	0123458A69CEF7DB	-,3	$\mathcal{C}^4_{283}$	0123468B5C9DE7AF	-,3
$\mathcal{C}^4_{252}$	0123458A69CEFBD7	2,2	$\mathcal{C}^4_{284}$	0123458A6BCF9D7E	-,3
$\mathcal{C}^4_{253}$	0123458A69CE7FBD	-,3	$\mathcal{C}^4_{285}$	0123457869CEAFBD	-,2
$\mathcal{C}^4_{254}$	0123458A69BCFD7E	3,3	$\mathcal{C}^4_{286}$	0123458967CEFBDA	2,2
$\mathcal{C}^4_{255}$	012345786ABCEDF9	-,3	$\mathcal{C}^4_{287}$	012345768A9CDFBE	3,3
$\mathcal{C}^4_{256}$	012345896ACF7BDE	-,3	$\mathcal{C}^4_{288}$	0123456789CEFBDA	2,2
$\mathcal{C}^4_{257}$	012345896ABCFD7E	-,2	$\mathcal{C}^4_{289}$	0123456789CEBFDA	-,3
$\mathcal{C}^4_{258}$	012345896ACE7BDF	2,2	$\mathcal{C}^4_{290}$	0123456789BCEAFD	-,3
$\mathcal{C}^4_{259}$	012345896ACEFDB7	2,2	$\mathcal{C}^4_{291}$	012345768A9BCFED	-,3
$\mathcal{C}^4_{260}$	012345896AB7CEFD	2,2	$\mathcal{C}^4_{292}$	012345768A9BCEFD	2,2
$\mathcal{C}^4_{261}$	0123458A69CEB7FD	-,3	$\mathcal{Q}^4_{293}$	0123457689CDEFBA	1,1
$\mathcal{C}^4_{262}$	0123458A6C7DB9FE	2,2	$\mathcal{Q}^4_{294}$	0123456789BAEFDC	1,1
$\mathcal{C}^4_{263}$	0123458A6BC7EDF9	-,3	$\mathcal{C}^4_{295}$	0123468C59DFA7BE	-,3
$\mathcal{C}^4_{264}$	0123458A6C7DFEB9	2,2	$\mathcal{C}^4_{296}$	0123468A5BCF7E9D	2,2
$\mathcal{C}^4_{265}$	0123458A6BCDE9F7	-,3	$\mathcal{C}^4_{297}$	0123468A5BCF79DE	2,2
$\mathcal{C}^4_{266}$	0123468A5BCFED97	2,2	$\mathcal{C}^4_{298}$	012345687ACEB9FD	-,2
$\mathcal{C}^4_{267}$	012345786ABCE9DF	-,3	$\mathcal{Q}^4_{299}$	012345678ACEB9FD	1,1
$\mathcal{C}^4_{268}$	0123458A69CFBED7	3,3	$\mathcal{Q}^4_{300}$	0123458967CDEFAB	2,1
$\mathcal{C}^4_{269}$	0123458A69CEBFD7	-,3	$\mathcal{C}^4_{301}$	0123458967CDEFBA	-,1

Table A.7: Quadratic decomposition length 2

Class # in A_{16}	Quadratic Decomposition length 2: quadratic $\times$ quadratic	# simple solutions
$\mathcal{C}^4_{130}$	300×299	1
$\mathcal{C}^4_{131}$	299×300	1
$\mathcal{C}^4_{150}$	$12 \times 293, 293 \times 300, 300 \times 12, 300 \times 300$	4
$\mathcal{C}^4_{151}$	$12 \times 300, 293 \times 12, 300 \times 293, 300 \times 300$	4
$\mathcal{C}^4_{158}$	299×293	1
$\mathcal{C}^4_{159}$	293×299	1
$\mathcal{C}^4_{168}$	$12 \times 300, 293 \times 293, 300 \times 12, 300 \times 300$	4
$\mathcal{C}^4_{171}$	$293 \times 12, 293 \times 300, 294 \times 293, 294 \times 300$	4
$\mathcal{C}^4_{172}$	$12 \times 293, 293 \times 294, 300 \times 293, 300 \times 294$	4
$\mathcal{C}^4_{214}$	$4 \times 299, 12 \times 12, 12 \times 294, 12 \times 299, 293 \times 4, 293 \times 12, 293 \times 294, 293 \times 299, 294 \times 12, 294 \times 294, 294 \times 299, 300 \times 4, 300 \times 12, 300 \times 294, 300 \times 299$	15
$\mathcal{C}^4_{215}$	$4 \times 293, 4 \times 300, 12 \times 12, 12 \times 293, 12 \times 294, 12 \times 300, 294 \times 12, 294 \times 293, 294 \times 294, 294 \times 300, 299 \times 4, 299 \times 12, 299 \times 293, 299 \times 294, 299 \times 300$	15
$\mathcal{C}^4_{223}$	$12 \times 293, 293 \times 293, 293 \times 294, 294 \times 293, 294 \times 294, 299 \times 12, 299 \times 299$	7
$\mathcal{C}^4_{233}$	$12 \times 12, 293 \times 293, 293 \times 300, 294 \times 12, 294 \times 300, 299 \times 12, 300 \times 293, 300 \times 300$	8
$\mathcal{C}^4_{234}$	$12 \times 12, 12 \times 294, 12 \times 299, 293 \times 293, 293 \times 300, 300 \times 293, 300 \times 294, 300 \times 300$	8
$\mathcal{C}^4_{236}$	$12 \times 12, 293 \times 293, 293 \times 294, 293 \times 300, 294 \times 293, 294 \times 294, 299 \times 299, 300 \times 293, 300 \times 300$	9
$\mathcal{C}^4_{238}$	$12 \times 300, 293 \times 293, 300 \times 12, 300 \times 300$	4

Table A.8: Quadratic decomposition length 2 cont'd

Class # in A_{16}	Quadratic Decomposition length 2: quadratic × quadratic	# simple solutions
$\mathcal{C}_{243}^4$	$4 \times 293, 4 \times 294, 12 \times 4, 12 \times 293, 12 \times 294, 12 \times 299, 293 \times 12, 293 \times 294, 294 \times 4, 294 \times 12, 294 \times 293, 294 \times 294, 299 \times 4, 299 \times 293, 299 \times 294, 300 \times 12, 300 \times 294, 300 \times 299$	18
$\mathcal{C}_{244}^4$	$4 \times 12, 4 \times 294, 4 \times 299, 12 \times 293, 12 \times 294, 12 \times 300, 293 \times 4, 293 \times 12, 293 \times 294, 293 \times 300, 294 \times 4, 294 \times 12, 294 \times 293, 294 \times 294, 294 \times 299, 294 \times 300, 299 \times 12, 299 \times 300$	18
$\mathcal{C}_{252}^4$	$299 \times 300, 300 \times 299$	2
$\mathcal{C}_{258}^4$	$4 \times 12, 4 \times 300, 12 \times 4, 12 \times 12, 12 \times 293, 12 \times 294, 12 \times 299, 12 \times 300, 293 \times 12, 293 \times 294, 293 \times 299, 294 \times 12, 294 \times 293, 294 \times 299, 294 \times 300, 299 \times 12, 299 \times 293, 299 \times 294, 299 \times 300, 300 \times 4, 300 \times 12, 300 \times 294, 300 \times 299$	23
$\mathcal{C}_{259}^4$	$4 \times 12, 4 \times 300, 12 \times 12, 12 \times 293, 12 \times 294, 12 \times 299, 12 \times 300, 293 \times 4, 293 \times 12, 293 \times 294, 293 \times 299, 294 \times 4, 294 \times 12, 294 \times 293, 294 \times 294, 294 \times 300, 299 \times 12, 299 \times 293, 299 \times 294, 299 \times 300, 300 \times 12, 300 \times 294, 300 \times 299$	23
$\mathcal{C}_{260}^4$	$4 \times 293, 4 \times 294, 12 \times 4, 12 \times 12, 12 \times 293, 12 \times 294, 12 \times 299, 12 \times 300, 293 \times 12, 293 \times 294, 293 \times 299, 294 \times 12, 294 \times 293, 294 \times 294, 294 \times 299, 294 \times 299, 299 \times 12, 299 \times 293, 299 \times 300, 300 \times 4, 300 \times 12, 300 \times 294, 300 \times 299$	23
$\mathcal{C}_{262}^4$	$12 \times 299, 294 \times 299, 299 \times 12, 299 \times 294$	4
$\mathcal{C}_{264}^4$	$12 \times 294, 293 \times 293, 293 \times 300, 294 \times 12, 294 \times 300, 299 \times 299, 300 \times 293, 300 \times 294$	8
$\mathcal{C}_{266}^4$	$12 \times 12, 293 \times 300, 294 \times 299, 299 \times 294, 299 \times 299, 300 \times 293, 300 \times 300$	7
$\mathcal{C}_{286}^4$	$12 \times 293, 12 \times 300, 293 \times 12, 293 \times 300, 300 \times 12, 300 \times 293, 300 \times 300$	7
$\mathcal{C}_{288}^4$	$12 \times 12, 293 \times 300, 300 \times 293, 300 \times 300$	4

Table A.9: Quadratic decomposition length 2 cont'd

Class # in A_{16}	Quadratic Decomposition length 2: quadratic $\times$ quadratic	# simple solutions
$\mathcal{C}^4_{292}$	$4 \times 4, 4 \times 12, 4 \times 294, 12 \times 4, 12 \times 12, 12 \times 293, 12 \times 294, 12 \times 300, 293 \times 12, 293 \times 294, 293 \times 299, 294 \times 4, 294 \times 12, 294 \times 293, 294 \times 294, 294 \times 299, 294 \times 300, 299 \times 293, 299 \times 294, 299 \times 300, 300 \times 12, 300 \times 294, 300 \times 299$	23
$\mathcal{C}^4_{296}$	$12 \times 299, 293 \times 293, 293 \times 300, 294 \times 12, 294 \times 300, 299 \times 294, 299 \times 299$	7
$\mathcal{C}^4_{297}$	$12 \times 294, 293 \times 293, 294 \times 299, 299 \times 12, 299 \times 299, 300 \times 293, 300 \times 294$	7

Table A.10: Known S-boxes and their classes

Class	Cipher
$\mathcal{C}^4_{39}$	DESL Row2, DESL Row3
$\mathcal{C}^4_{46}$	DES7 Row3
$\mathcal{C}^4_{59}$	DES7 Row1
$\mathcal{C}^4_{69}$	DES3 Row1, DES7 Row0
$\mathcal{C}^4_{74}$	DES6 Row1
$\mathcal{C}^4_{80}$	DES8 Row2
$\mathcal{C}^4_{85}$	DES1 Row0, DES1 Row1, DES1 Row2, DES8 Row3
$\mathcal{C}^4_{97}$	DES8 Row0
$\mathcal{C}^4_{108}$	Twofish q1 t1
$\mathcal{C}^4_{117}$	DES2 Row0, DES6 Row3
$\mathcal{C}^4_{120}$	Twofish q0 t3
$\mathcal{C}^4_{137}$	DES8 Row1
$\mathcal{C}^4_{139}$	DES3 Row0, DES5 Row0
$\mathcal{C}^4_{142}$	Twofish q1 t3
$\mathcal{C}^4_{145}$	Gost K6
$\mathcal{C}^4_{148}$	DES5 Row3
$\mathcal{C}^4_{153}$	Twofish q1 t0

Table A.11: Known S-boxes and their classes cont'd

Class	Cipher
$\mathcal{C}^4_{154}$	Gost K5
$\mathcal{C}^4_{160}$	Serpent3, Serpent7, Clefia2, Clefia3, HB1 S1, HB1 S3, HB2 S0, Optimal G_9
$\mathcal{C}^4_{163}$	Clefia1, HB1 S2, HB2 S1, Optimal G_{10}
$\mathcal{C}^4_{166}$	DES2 Row1, DESL Row0
$\mathcal{C}^4_{172}$	Gost K1
$\mathcal{C}^4_{177}$	Gost K8
$\mathcal{C}^4_{184}$	DES1 Row3
$\mathcal{C}^4_{188}$	Lucifer S0
$\mathcal{C}^4_{190}$	Twofish q0 t0
$\mathcal{C}^4_{196}$	Optimal G_7
$\mathcal{C}^4_{197}$	Lucifer S1
$\mathcal{C}^4_{203}$	DESL Row1
$\mathcal{C}^4_{204}$	DES2 Row2, DES3 Row2
$\mathcal{C}^4_{206}$	Gost K7
$\mathcal{C}^4_{208}$	Twofish q0 t1
$\mathcal{C}^4_{209}$	Serpent4, Serpent5, HB2 S2, Optimal G_{15}
$\mathcal{C}^4_{210}$	Clefia0, Twofish q0 t2, HB1 S0, HB2 S3, Optimal G_{14}
$\mathcal{C}^4_{220}$	DES6 Row0
$\mathcal{C}^4_{221}$	DES5 Row2
$\mathcal{C}^4_{223}$	Noekeon, Luffa v1, Piccolo, Optimal G_8
$\mathcal{C}^4_{229}$	Twofish q1 t2
$\mathcal{C}^4_{231}$	JH S0, JH S1, Optimal G_{13}
$\mathcal{C}^4_{253}$	Gost K3
$\mathcal{C}^4_{254}$	DES5 Row1
$\mathcal{C}^4_{257}$	DES3 Row3
$\mathcal{C}^4_{266}$	Present, Serpent2, Serpent6, Luffa v2, Hamsi, Optimal G_1
$\mathcal{C}^4_{267}$	Gost K4
$\mathcal{C}^4_{270}$	Klein, KhazadP, KhazadQ, Iceberg G0, Iceberg G1, Puffin, Optimal G_4
$\mathcal{C}^4_{272}$	Optimal G_6
$\mathcal{C}^4_{275}$	Gost K2
$\mathcal{C}^4_{278}$	Optimal G_5

Table A.12: Known S-boxes and their classes cont'd

Class	Cipher
$\mathcal{C}^4_{279}$	DES2 Row3, DES4 Row0, DES4 Row1, DES4 Row2, DES4 Row3, DES7 Row2
$\mathcal{C}^4_{281}$	DES6 Row2
$\mathcal{C}^4_{282}$	Inversion in $\mathbb{GF}(2^4)$, Optimal G_3 mCrypton S0,S1,S2,S3
$\mathcal{C}^4_{283}$	Optimal G_{12}
$\mathcal{C}^4_{295}$	Optimal G_{11}
$\mathcal{C}^4_{296}$	Serpent1, Optimal G_0
$\mathcal{C}^4_{297}$	Serpent0, Optimal G_2

B
Equations

B.1 Equations Used for First-order TI of Quadratic 4-bit Permutations with Two Input Shares

Each permutation $f(W, X, Y, Z) = (A, B, C, D)$ has 4 input and output bits. The component functions f^1, f^2, f^3, f^4 outputs A, B, C, D respectively. W (resp. A) is the most significant bit whereas Z (resp. D) is the least significant bit. In some cases the sharing of an output variable is not uniform and requires remasking if used as is. We also describe the two-sharing (e.g. $\bar{\mathbf{A}}$ with shares $\bar{A}_i$) after the decrease of the number of shares such that the two sharing of f is uniform.

B.1.1 Class Q_4^4

$f = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 12, 15, 14]$

$$A = f^1(W, X, Y, Z) = W$$

$$B = f^2(W, X, Y, Z) = X$$

$$C = f^3(W, X, Y, Z) = Y$$

$$D = f^4(W, X, Y, Z) = WX \oplus Z$$

$$A_1 = f_1^1(\mathbf{W,X,Y,Z}) = W_1$$

$$A_2 = f_2^1(\mathbf{W,X,Y,Z}) = W_2 \qquad\qquad \bar{A}_1 = A_1$$

$$B_1 = f_1^2(\mathbf{W,X,Y,Z}) = X_1 \qquad\qquad \bar{A}_2 = A_2$$

$$B_2 = f_2^2(\mathbf{W,X,Y,Z}) = X_2 \qquad\qquad \bar{B}_1 = B_1$$

$$C_1 = f_1^3(\mathbf{W,X,Y,Z}) = Y_1 \qquad\qquad \bar{B}_2 = B_2$$

$$C_2 = f_2^3(\mathbf{W,X,Y,Z}) = Y_2 \qquad\qquad \bar{C}_1 = C_1$$

$$D_1 = f_1^4(\mathbf{W,X,Y,Z}) = W_1 X_1 \oplus Z_1 \qquad\qquad \bar{C}_2 = C_2$$

$$D_2 = f_2^4(\mathbf{W,X,Y,Z}) = W_1 X_2 \qquad\qquad \bar{D}_1 = D_1 \oplus D_2$$

$$D_3 = f_3^4(\mathbf{W,X,Y,Z}) = W_2 X_1 \qquad\qquad \bar{D}_2 = D_3 \oplus D_4$$

$$D_4 = f_4^4(\mathbf{W,X,Y,Z}) = W_2 X_2 \oplus Z_2$$

B.1.2 Class Q_{12}^4

$$f = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 15, 10, 11]$$

$$A = f^1(W, X, Y, Z) = W$$

$$B = f^2(W, X, Y, Z) = WY \oplus X$$

$$C = f^3(W, X, Y, Z) = WX \oplus WY \oplus Y$$

$$D = f^4(W, X, Y, Z) = Z$$

$$A_1 = f_1^1(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_1$$

$$A_2 = f_2^1(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_2$$

$$B_1 = f_1^2(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_1Y_1 \oplus X_1 \qquad\qquad \bar{A}_1 = A_1$$

$$B_2 = f_2^2(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_1Y_2 \qquad\qquad \bar{A}_2 = A_2$$

$$B_3 = f_3^2(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_2Y_1 \oplus X_2 \qquad\qquad \bar{B}_1 = B_1 \oplus B_2$$

$$B_4 = f_4^2(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_2Y_2 \qquad\qquad \bar{B}_2 = B_3 \oplus B_4$$

$$C_1 = f_1^3(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_1X_1 \oplus W_1Y_1 \oplus Y_1 \qquad\qquad \bar{C}_1 = C_1 \oplus C_2$$

$$C_2 = f_2^3(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_1X_2 \oplus W_1Y_2 \qquad\qquad \bar{C}_2 = C_3 \oplus C_4$$

$$C_3 = f_3^3(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_2X_1 \oplus W_2Y_1 \qquad\qquad \bar{D}_1 = D_1$$

$$C_4 = f_4^3(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = W_2X_2 \oplus W_2Y_2 \oplus Y_2 \qquad\qquad \bar{D}_2 = D_2$$

$$D_1 = f_1^4(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = Z_1$$

$$D_2 = f_2^4(\mathbf{W},\mathbf{X},\mathbf{Y},\mathbf{Z}) = Z_2$$

B.1.3 Class Q_{293}^4

$$f = [0, 1, 2, 3, 4, 5, 7, 6, 8, 9, 12, 13, 14, 15, 11, 10]$$

$$A = f^1(W, X, Y, Z) = W$$

$$B = f^2(W, X, Y, Z) = WY \oplus X$$

$$C = f^3(W, X, Y, Z) = WX \oplus WY \oplus Y$$

$$D = f^4(W, X, Y, Z) = XY \oplus Z$$

$$A_1 = f_1^1(\mathbf{W,X,Y,Z}) = W_1$$

$$A_2 = f_2^1(\mathbf{W,X,Y,Z}) = W_2$$

$$B_1 = f_1^2(\mathbf{W,X,Y,Z}) = W_1Y_1 \oplus X_1$$

$$B_2 = f_2^2(\mathbf{W,X,Y,Z}) = W_1Y_2 \qquad\qquad \bar{A}_1 = A_1$$

$$B_3 = f_3^2(\mathbf{W,X,Y,Z}) = W_2Y_1 \oplus X_2 \qquad\qquad \bar{A}_2 = A_2$$

$$B_4 = f_4^2(\mathbf{W,X,Y,Z}) = W_2Y_2 \qquad\qquad \bar{B}_1 = B_1 \oplus B_2$$

$$C_1 = f_1^3(\mathbf{W,X,Y,Z}) = W_1X_1 \oplus W_1Y_1 \oplus Y_1 \qquad \bar{B}_2 = B_3 \oplus B_4$$

$$C_2 = f_2^3(\mathbf{W,X,Y,Z}) = W_1X_2 \oplus W_1Y_2 \qquad\qquad \bar{C}_1 = C_1 \oplus C_2$$

$$C_3 = f_3^3(\mathbf{W,X,Y,Z}) = W_2X_1 \oplus W_2Y_1 \qquad\qquad \bar{C}_2 = C_3 \oplus C_4$$

$$C_4 = f_4^3(\mathbf{W,X,Y,Z}) = W_2X_2 \oplus W_2Y_2 \oplus Y_2 \qquad \bar{D}_1 = D_1 \oplus D_2$$

$$D_1 = f_1^4(\mathbf{W,X,Y,Z}) = X_1Y_1 \oplus Z_1 \qquad\qquad \bar{D}_2 = D_3 \oplus D_4$$

$$D_2 = f_2^4(\mathbf{W,X,Y,Z}) = X_1Y_2$$

$$D_3 = f_3^4(\mathbf{W,X,Y,Z}) = X_2Y_1 \oplus Z_2$$

$$D_4 = f_4^4(\mathbf{W,X,Y,Z}) = X_2Y_2$$

B.1.4 Class Q_{294}^4

$$f = [0, 1, 2, 3, 4, 5, 7, 6, 8, 9, 12, 13, 14, 15, 11, 10]$$

$$A = f^1(W, X, Y, Z) = W$$

$$B = f^2(W, X, Y, Z) = X$$

$$C = f^3(W, X, Y, Z) = WX \oplus Y$$

$$D = f^4(W, X, Y, Z) = WY \oplus Z$$

$$A_1 = f_1^1(\mathbf{W,X,Y,Z}) = W_1$$

$$A_2 = f_2^1(\mathbf{W,X,Y,Z}) = W_2$$

$$B_1 = f_1^2(\mathbf{W,X,Y,Z}) = X_1 \qquad\qquad \bar{A}_1 = A_1$$

$$B_2 = f_2^2(\mathbf{W,X,Y,Z}) = X_2 \qquad\qquad \bar{A}_2 = A_2$$

$$C_1 = f_1^3(\mathbf{W,X,Y,Z}) = W_1X_1 \oplus Y_1 \qquad\qquad \bar{B}_1 = B_1$$

$$C_2 = f_2^3(\mathbf{W,X,Y,Z}) = W_1X_2 \qquad\qquad \bar{B}_2 = B_2$$

$$C_3 = f_3^3(\mathbf{W,X,Y,Z}) = W_2X_1 \oplus Y_2 \qquad\qquad \bar{C}_1 = C_1 \oplus C_2$$

$$C_4 = f_4^3(\mathbf{W,X,Y,Z}) = W_2X_2 \qquad\qquad \bar{C}_2 = C_3 \oplus C_4$$

$$D_1 = f_1^4(\mathbf{W,X,Y,Z}) = W_1Y_1 \oplus Z_1 \qquad\qquad \bar{D}_1 = D_1 \oplus D_2$$

$$D_2 = f_2^4(\mathbf{W,X,Y,Z}) = W_1Y_2 \qquad\qquad \bar{D}_2 = D_3 \oplus D_4$$

$$D_3 = f_3^4(\mathbf{W,X,Y,Z}) = W_2Y_1 \oplus Z_2$$

$$D_4 = f_4^4(\mathbf{W,X,Y,Z}) = W_2Y_2$$

B.1.5 Class Q_{299}^4

$f = [0, 1, 2, 3, 4, 5, 7, 8, 10, 12, 14, 11, 9, 15, 13]$

$$A = f^1(W, X, Y, Z) = W$$

$$B = f^2(W, X, Y, Z) = WX \oplus WY \oplus X$$

$$C = f^3(W, X, Y, Z) = WX \oplus WY \oplus WZ \oplus Y$$

$$D = f^4(W, X, Y, Z) = WX \oplus WZ \oplus Z$$

$$A_1 = f_1^1(\mathbf{W,X,Y,Z}) = W_1$$

$$A_2 = f_2^1(\mathbf{W,X,Y,Z}) = W_2$$

$$B_1 = f_1^2(\mathbf{W,X,Y,Z}) = W_1X_1 \oplus W_1Y_1 \oplus X_1$$

$$B_2 = f_2^2(\mathbf{W,X,Y,Z}) = W_1X_2 \oplus W_1Y_2 \qquad\qquad \bar{A}_1 = A_1$$

$$B_3 = f_3^2(\mathbf{W,X,Y,Z}) = W_2X_1 \oplus W_2Y_1 \qquad\qquad \bar{A}_2 = A_2$$

$$B_4 = f_4^2(\mathbf{W,X,Y,Z}) = W_2X_2 \oplus W_2Y_2 \oplus X_2 \qquad\quad \bar{B}_1 = B_1 \oplus B_2$$

$$C_1 = f_1^3(\mathbf{W,X,Y,Z}) = W_1X_1 \oplus W_1Y_1 \oplus W_1Z_1 \oplus Y_1 \qquad \bar{B}_2 = B_3 \oplus B_4$$

$$C_2 = f_2^3(\mathbf{W,X,Y,Z}) = W_1X_2 \oplus W_1Y_2 \oplus W_1Z_2 \qquad\quad \bar{C}_1 = C_1 \oplus C_2$$

$$C_3 = f_3^3(\mathbf{W,X,Y,Z}) = W_2X_1 \oplus W_2Y_1 \oplus W_2Z_1 \qquad\quad \bar{C}_2 = C_3 \oplus C_4$$

$$C_4 = f_4^3(\mathbf{W,X,Y,Z}) = W_2X_2 \oplus W_2Y_2 \oplus W_2Z_2 \oplus Y_2 \qquad \bar{D}_1 = D_1 \oplus D_2$$

$$D_1 = f_1^4(\mathbf{W,X,Y,Z}) = W_1X_1 \oplus W_1Z_1 \oplus Z_1 \qquad\qquad \bar{D}_2 = D_3 \oplus D_4$$

$$D_2 = f_2^4(\mathbf{W,X,Y,Z}) = W_1X_2 \oplus W_1Z_2$$

$$D_3 = f_3^4(\mathbf{W,X,Y,Z}) = W_2X_1 \oplus W_2Z_1$$

$$D_4 = f_4^4(\mathbf{W,X,Y,Z}) = W_2X_2 \oplus W_2Z_2 \oplus Z_2$$

B.1.6   Class Q_{300}^4

$$f = [0, 1, 2, 3, 4, 5, 8, 9, 13, 12, 7, 6, 11, 10, 15, 14]$$

$$A = f^1(W, X, Y, Z) = WY \oplus XY \oplus W$$

$$B = f^2(W, X, Y, Z) = XY \oplus W \oplus X$$

$$C = f^3(W, X, Y, Z) = WX \oplus XY \oplus Y$$

$$D = f^4(W, X, Y, Z) = W \oplus Z$$

$$A_1 = f_1^1(\mathbf{W,X,Y,Z}) = W_1Y_1 \oplus X_1Y_1 \oplus W_1$$

$$A_2 = f_2^1(\mathbf{W,X,Y,Z}) = W_1Y_2 \oplus X_1Y_2$$

$$A_3 = f_3^1(\mathbf{W,X,Y,Z}) = W_2Y_1 \oplus X_2Y_1 \oplus W_2$$

$$A_4 = f_4^1(\mathbf{W,X,Y,Z}) = W_2Y_2 \oplus X_2Y_2$$

$$B_1 = f_1^2(\mathbf{W,X,Y,Z}) = X_1Y_1 \oplus W_1 \oplus X_1 \qquad\qquad \bar{A}_1 = A_1 \oplus A_2$$

$$B_2 = f_2^2(\mathbf{W,X,Y,Z}) = X_1Y_2 \qquad\qquad\qquad\qquad \bar{A}_2 = A_3 \oplus A_4$$

$$B_3 = f_3^2(\mathbf{W,X,Y,Z}) = X_2Y_1 \qquad\qquad\qquad\qquad \bar{B}_1 = B_1 \oplus B_2$$

$$B_4 = f_4^2(\mathbf{W,X,Y,Z}) = X_2Y_2 \oplus W_2 \oplus X_2 \qquad\qquad \bar{B}_2 = B_3 \oplus B_4$$

$$C_1 = f_1^3(\mathbf{W,X,Y,Z}) = W_1X_1 \oplus X_1Y_1 \oplus Y_1 \qquad\qquad \bar{C}_1 = C_1 \oplus C_2 \oplus C_3$$

$$C_2 = f_2^3(\mathbf{W,X,Y,Z}) = W_1X_2 \qquad\qquad\qquad\qquad \bar{C}_2 = C_4 \oplus C_5 \oplus C_6$$

$$C_3 = f_3^3(\mathbf{W,X,Y,Z}) = X_1Y_2 \qquad\qquad\qquad\qquad \bar{D}_1 = D_1$$

$$C_4 = f_4^3(\mathbf{W,X,Y,Z}) = W_2X_1 \qquad\qquad\qquad\qquad \bar{D}_2 = D_2$$

$$C_5 = f_5^3(\mathbf{W,X,Y,Z}) = X_2Y_1$$

$$C_6 = f_6^3(\mathbf{W,X,Y,Z}) = W_2X_2 \oplus X_2Y_2 \oplus Y_2$$

$$D_1 = f_1^4(\mathbf{W,X,Y,Z}) = W_1 \oplus Z_1$$

$$D_2 = f_2^4(\mathbf{W,X,Y,Z}) = W_2 \oplus Z_2$$

B.2 Equations Used for AES Implementations

B.2.1 Multiplier in $\mathbb{GF}(2^4)$

Multiplication takes two 4-bit inputs and outputs 4 bits. X (resp. K and A) is the most significant bit whereas W (resp. N and D) is the least significant bit.

$(A, B, C, D) = (X, Y, Z, W) \times (K, L, M, N)$

$$A = XK \oplus ZK \oplus WK \oplus YL \oplus ZL \oplus XM \oplus YM \oplus ZM \oplus WM$$
$$\oplus XN \oplus ZN$$
$$B = YK \oplus ZK \oplus XL \oplus YL \oplus WL \oplus XM \oplus ZM \oplus YN \oplus WN$$
$$C = XK \oplus YK \oplus ZK \oplus WK \oplus XL \oplus ZL \oplus XM \oplus YM \oplus ZM$$
$$\oplus XN \oplus WN$$
$$D = XK \oplus ZK \oplus YL \oplus WL \oplus XM \oplus WM \oplus YN \oplus ZN \oplus WN$$

B.2.2 Inverter in $\mathbb{GF}(2^4)$

Inverter has 4-bit input and output. X (resp. A) is the most significant bit whereas W (resp. D) is the least significant bit.

$(A, B, C, D) = Inv(X, Y, Z, W)$

$$A = Z \oplus W \oplus XZ \oplus YZ \oplus YZW$$
$$B = W \oplus XZ \oplus YZ \oplus YW \oplus XZW$$
$$C = X \oplus Y \oplus XZ \oplus XW \oplus XYW$$
$$D = Y \oplus XZ \oplus XW \oplus YW \oplus XYZ$$

B.2.3 Sharing with 4 Input 3 Output Shares

We use this sharing with 4 input and 3 output shares for each input and output variable respectively in order to implement shared $\mathbb{GF}(2^4)$ multiplier. This

multiplier is composed of only quadratic terms. Therefore, here we demonstrate how one of these quadratic terms (such as XK from the equation in B.2.1) is implemented. Shared one-bit of multiplication output can be generated by XORing the corresponding shared quadratic terms. Hence X and Y represent one-bit inputs and A represents one-bit output.

$A = XY$

$$A_1 = (X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3) \oplus Y_4$$

$$A_2 = ((X_1 \oplus X_3)(Y_1 \oplus Y_4)) \oplus X_1 Y_3 \oplus X_4$$

$$A_3 = ((X_2 \oplus X_4)(Y_1 \oplus Y_4)) \oplus X_1 Y_2 \oplus X_4 \oplus Y_4$$

B.2.4   Sharing with 3 Input 3 Output Shares

Here we exemplify sharing linear and quadratic terms using 3 input and output shares for each variable. X, Y and Z represent one-bit inputs and A represents one-bit output. Note that the sharings of Z and XY can be used separately. Here we describe the combined sharing using one function for convenience. Similarly, the sharing for a function using more than one linear and quadratic term can be generated by doubling the sharing of the corresponding term with correct inputs.

$A = XY \oplus Z$

$$A_1 = ((X_2 \oplus X_3)(Y_2 \oplus Y_3)) \oplus Z_2$$

$$A_2 = (X_1 Y_3 \oplus Y_1 X_3 \oplus X_1 Y_1) \oplus Z_3$$

$$A_3 = (X_1 Y_2 \oplus Y_1 X_2) \oplus Z_1$$

B.2.5   Sharing with 4 Input 4 Output Shares

Sharing linear, quadratic and cubic terms using 4 input and output shares for each variable is given in follows. X, Y and Z represent one-bit inputs and A represents one-bit output. Note that the sharings of Z, XY and XYZ can be used separately. Here we describe the combined sharing using one function for convenience. Similarly, the sharing for a function using more than one linear, quadratic or cubic term can be generated by doubling the sharing of the

corresponding term with correct inputs. Note that if the function does not have any cubic or linear terms, the following sharing produces only 3 output shares.

$$A = XYZ \oplus XY \oplus Z$$

$$A_1 = ((X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3 \oplus Y_4)(Z_2 \oplus Z_3 \oplus Z_4))$$

$$\oplus ((X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3 \oplus Y_4)) \oplus Z_2$$

$$A_2 = (X_1(Y_3 \oplus Y_4)(Z_3 \oplus Z_4) \oplus Y_1(X_3 \oplus X_4)(Z_3 \oplus Z_4) \oplus Z_1(X_3 \oplus X_4)(Y_3 \oplus Y_4)$$

$$\oplus X_1Y_1(Z_3 \oplus Z_4) \oplus X_1Z_1(Y_3 \oplus Y_4) \oplus Y_1Z_1(X_3 \oplus X_4) \oplus X_1Y_1Z_1)$$

$$\oplus (X_1(Y_3 \oplus Y_4) \oplus Y_1(X_3 \oplus X_4) \oplus X_1Y_1) \oplus Z_3$$

$$A_3 = (X_1Y_1Z_2 \oplus X_1Y_2Z_1 \oplus X_2Y_1X_1 \oplus X_1Y_2Z_2 \oplus X_2Y_1Z_2 \oplus X_2Y_2Z_1 \oplus X_1Y_2Z_4$$

$$\oplus X_2Y_1Z_4 \oplus X_1Y_4Z_2 \oplus X_2Y_4Z_1 \oplus X_4Y_1Z_2 \oplus X_4Y_2Z_1) \oplus (X_1Y_2 \oplus Y_1X_2) \oplus Z_4$$

$$A_4 = (X_1Y_2Z_3 \oplus X_1Y_3Z_2 \oplus X_2Y_1Z_3 \oplus X_2Y_3Z_1 \oplus X_3Y_1Z_2 \oplus X_3Y_2Z_1) \oplus 0 \oplus Z_1$$

B.2.6   Sharing with 5 Input 5 Output Shares

Here, we describe sharing linear, quadratic and cubic terms using 5 input and output shares for each variable. X, Y and Z represent one-bit inputs and A represents one-bit output. Note that the sharings of Z, XY and XYZ can be used separately. Here we describe the combined sharing using one function for convenience. Similarly, the sharing for a function using more than one linear, quadratic or cubic term can be generated by doubling the sharing of the corresponding term with correct inputs. Note that if the function does not have any linear terms, the following sharing produces only 4 output shares. Moreover, if the function lacks both linear and cubic terms, the sharing produces only 3 output shares.

$$A = XYZ \oplus XY \oplus Z$$

$$A_1 = ((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)(Z_2 \oplus Z_3 \oplus Z_4 \oplus Z_5))$$
$$\oplus ((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)) \oplus Z_2$$

$$A_2 = (X_1(Y_3 \oplus Y_4 \oplus Y_5)(Z_3 \oplus Z_4 \oplus Z_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5)(Z_3 \oplus Z_4 \oplus Z_5)$$
$$\oplus Z_1(X_3 \oplus X_4 \oplus X_5)(Y_3 \oplus Y_4 \oplus Y_5) \oplus X_1Y_1(Z_3 \oplus Z_4 \oplus Z_5) \oplus X_1Z_1(Y_3 \oplus Y_4 \oplus Y_5)$$
$$\oplus Y_1Z_1(X_3 \oplus X_4 \oplus X_5) \oplus X_1Y_1Z_1) \oplus (X_1(Y_3 \oplus Y_4 \oplus Y_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5)$$
$$\oplus X_1Y_1) \oplus Z_3$$

$$A_3 = (X_1Y_1Z_2 \oplus X_1Y_2Z_1 \oplus X_2Y_1X_1 \oplus X_1Y_2Z_2 \oplus X_2Y_1Z_2 \oplus X_2Y_2Z_1 \oplus X_1Y_2Z_4$$
$$\oplus X_2Y_1Z_4 \oplus X_1Y_4Z_2 \oplus X_2Y_4Z_1 \oplus X_4Y_1Z_2 \oplus X_4Y_2Z_1 \oplus X_1Y_2Z_5 \oplus X_2Y_1Z_5$$
$$\oplus X_1Y_5Z_2 \oplus X_2Y_5Z_1 \oplus X_5Y_1Z_2 \oplus X_5Y_2Z_1) \oplus (X_1Y_2 \oplus Y_1X_2) \oplus Z_4$$

$$A_4 = (X_1Y_2Z_3 \oplus X_1Y_3Z_2 \oplus X_2Y_1Z_3 \oplus X_2Y_3Z_1 \oplus X_3Y_1Z_2 \oplus X_3Y_2Z_1) \oplus 0 \oplus Z_5$$

$$A_5 = 0 \oplus 0 \oplus Z_1$$

Bibliography

[1] The KECCAK sponge function family. http://keccak.noekeon.org. pages 88, 89

[2] ATHENa: Automated tool for hardware evaluation. http://cryptography.gmu.edu/athena/. pages 95

[3] AIST. Side-channel Attack Standard Evaluation BOard. `http://staff.aist.go.jp/akashi.satoh/SASEBO/en/`. pages 60, 114

[4] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, F. Mendel, B. Mennink, N. Mouha, Q. Wang, and K. Yasuda. PRIMATEs. http://competitions.cr.yp.to/round1/primatesv1.pdf. pages 9, 66, 83

[5] J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F.-X. Standaert. On the cost of lazy engineering for masked software implementations. In M. Joye and A. Moradi, editors, *Smart Card Research and Advanced Applications - CARDIS 2014*, LNCS. Springer, 11 2014. pages 20, 40

[6] J. Balasch, B. Gierlichs, R. Verdult, L. Batina, and I. Verbauwhede. Power analysis of Atmel CryptoMemory – recovering keys from secure EEPROMs. In O. Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *LNCS*, pages 19–34. Springer Berlin Heidelberg, 2012. pages 4

[7] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. Cryptology ePrint Archive, Report 2004/100, 2004. `http://eprint.iacr.org/`. pages 3

[8] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *J. Cryptology*, 24(2):269–291, 2011. pages 6

[9] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Building power analysis resistant implementations of Keccak. Second SHA-3 candidate conference, August 2010. pages 87, 90, 100

[10] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Cryptographic sponge functions, January 2011. pages 88

[11] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The Keccak reference, January 2011. pages 89

[12] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer. Keccak implementation overview, September 2011. pages 96

[13] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak specifications. NIST SHA-3 contest 2008. pages 14, 16

[14] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *LNCS*, pages 320–337. Springer Berlin Heidelberg, 2012. pages 89

[15] B. Bilgin, A. Bogdanov, M. Kneževic, F. Mendel, and Q. Wang. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In G. Bertoni and J.-S. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 142–158. Springer Berlin Heidelberg, 2013. pages 9, 66, 83

[16] B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche. Efficient and first-order DPA resistant implementations of Keccak. In A. Francillon and P. Rohatgi, editors, *Smart Card Research and Advanced Applications*, volume 8419 of *LNCS*, pages 187–199. Springer International Publishing, 2014. pages 10, 88

[17] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-order threshold implementations. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 326–343. Springer Berlin Heidelberg, 2014. pages 9, 12, 30, 56

[18] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. A more efficient AES threshold implementation. In D. Pointcheval and D. Vergnaud, editors, *Progress in Cryptology – AFRICACRYPT 2014*, volume 8469 of *LNCS*, pages 267–284. Springer International Publishing, 2014. pages 10, 12, 30, 104

[19] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Trade-offs for threshold implementations illustrated on AES. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–13, 2015. pages 10, 104

[20] B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen. TI toolkit, 2013. `http://homes.esat.kuleuven.be/~snikova/ti_tools.html`. pages 10, 66, 85

[21] B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen. List of decompositions of 4-bit permutations, 2014. `http://homes.esat.kuleuven.be/~bbilgin/other.html`. pages 10, 74, 85

[22] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold implementations of all 3x3 and 4x4 S-boxes. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *LNCS*, pages 76–91. Springer Berlin Heidelberg, 2012. pages 9, 66

[23] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup. Threshold implementations of small S-boxes. *Cryptography and Communications*, 7(1):3–33, 2015. pages 9, 12, 66

[24] A. Biryukov, C. De Cannière, A. Braeken, and B. Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In E. Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 33–50. Springer Berlin Heidelberg, 2003. pages 15

[25] A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer Berlin Heidelberg, 2007. pages 14

[26] C. Boura and A. Canteaut. On the influence of the algebraic degree of f^{-1} on the algebraic degree of $g \circ f$. Cryptology ePrint Archive, Report 2011/503, 2011. `http://eprint.iacr.org/`. pages 12, 16

[27] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer Berlin Heidelberg, 2004. pages 19

[28] M. Brinkmann and G. Leander. On the classification of APN functions up to dimension five. *Designs, Codes and Cryptography*, 49(1-3):273–288, 2008. pages 17, 18

[29] D. Canright. A very compact S-box for AES. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer Berlin Heidelberg, 2005. pages 19

[30] C. Carlet. Vectorial Boolean functions for cryptography. In Y. Crama and P. L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering.* Cambridge University Press, New York, NY, USA, 2010. pages 15, 68

[31] C. Carlet, P. Charpin, and V. Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, Nov. 1998. pages 17

[32] S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, volume 1666 of *LNCS*, pages 398–412. Springer Berlin Heidelberg, 1999. pages 7, 118

[33] H. Chen and D. Feng. An effective genetic algorithm for self-inverse S-boxes. In *CIS*, pages 618–622. IEEE Computer Society, 2007. pages 18

[34] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot. White-box cryptography and an AES implementation. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *LNCS*, pages 250–270. Springer Berlin Heidelberg, 2003. pages 2

[35] J. Cooper, E. D. Mulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test vector leakage assessment (TVLA) methodology in practice. International Cryptographic Module Conference, 2013. `http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf`. pages 26, 61, 62

[36] J.-S. Coron. Higher order masking of look-up tables. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 441–458. Springer Berlin Heidelberg, 2014. pages 23

[37] J. Daemen, J. Daemen, J. Daemen, V. Rijmen, and V. Rijmen. AES proposal: Rijndael, 1998. pages 18

[38] J. Daemen, R. Govaerts, and J. Vandewalle. A new approach to block cipher design. In R. Anderson, editor, *Fast Software Encryption*, volume 809 of *LNCS*, pages 18–32. Springer Berlin Heidelberg, 1994. pages 15

[39] J. Daemen, M. Peeters, and G. Assche. Bitslice ciphers and power analysis attacks. In G. Goos, J. Hartmanis, J. Leeuwen, and B. Schneier, editors, *Fast Software Encryption*, volume 1978 of *LNCS*, pages 134–149. Springer Berlin Heidelberg, 2001. pages 15

[40] C. De Canniere. *Analysis and Design of Symmetric Encryption Algorithms*. PhD thesis, 2007. pages 14, 15, 16

[41] C. De Canniere, V. Nikov, S. Nikova, and V. Rijmen. S-box decompositions for SCA-resisting implementations, 2012. Poster presented at CHES 2011, Nara, Japan. pages 72

[42] C. De Cannière, O. Dunkelman, and M. Knežević. KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer Berlin Heidelberg, 2009. pages 55, 56

[43] DES. Data encryption standard. In *In FIPS PUB 46, Federal Information Processing Standards Publication*, pages 46–2, 1977. pages 14, 65, 83

[44] J. F. Dillon. APN polynomials: An update, July 2009. pages 18, 82

[45] A. Duc, S. Dziembowski, and S. Faust. Unifying leakage models: From probing attacks to noisy leakage. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer Berlin Heidelberg, 2014. pages 23

[46] F. Durvaux, M. Renauld, F.-X. Standaert, L. van Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon. Cryptanalysis of the CHES 2009/2010 random delay countermeasure. Cryptology ePrint Archive, Report 2012/038, 2012. `http://eprint.iacr.org/`. pages 5

[47] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 293–302. IEEE, 2008. pages 5

[48] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani. Physical cryptanalysis of KeeLoq code hopping applications. Cryptology ePrint Archive, Report 2008/058, 2008. `http://eprint.iacr.org/`. pages 4

[49] S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 135–156. Springer Berlin Heidelberg, 2010. pages 23

[50] C. for Embedded Systems (CES). Optimizing energy efficient mobile computing: Explicit data communication and management tool, 2014. `http://faculty.washington.edu/scottcs/NSF/2014/CES.pdf`. pages 1

[51] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer Berlin Heidelberg, 2001. pages 3

[52] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop, 2011. `http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf`. pages 61

[53] L. Goubin and J. Patarin. DES and differential power analysis the "duplication" method. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *LNCS*, pages 158–172. Springer Berlin Heidelberg, 1999. pages 7

[54] V. Grosso, G. Leurent, F.-X. Standaert, and K. Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In C. Cid and C. Rechberger, editors, *FSE 2014*, LNCS (LNCS). Spinger, 3 2014. pages 18

[55] H. M. Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, July 2002. pages 2

[56] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer Berlin Heidelberg, 2003. pages 7, 23, 25

[57] D. Kahn. *The codebreakers : the story of secret writing*. Scribner, New York, 1996. pages 1

[58] S.-M. S. Kang and Y. Leblebici. *CMOS Digital Integrated Circuits Analysis and Design*. McGraw-Hill, Inc., New York, NY, USA, 3 edition, 2003. pages 24

[59] E. B. Kavun and T. Yalcin. A lightweight implementation of KECCAK hash function for radio-frequency identification applications. In S. Ors Yalcin, editor, *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *LNCS*, pages 258–269. Springer Berlin Heidelberg, 2010. pages 98

[60] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, 1883. pages 2

[61] L. Knudsen, G. Leander, A. Poschmann, and M. J. Robshaw. PRINTcipher: A block cipher for IC-printing. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 16–32. Springer Berlin Heidelberg, 2010. pages 15

[62] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer Berlin Heidelberg, 1996. pages 3

[63] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, volume 1666 of *LNCS*, pages 388–397. Springer Berlin Heidelberg, 1999. pages 3, 4

[64] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*, 1999. pages 3

[65] S. Kutzner, P. H. Nguyen, and A. Poschmann. Enabling 3-share threshold implementations for any 4-bit S-box. Cryptology ePrint Archive, Report 2012/510, 2012. http://eprint.iacr.org/. pages 75

[66] G. Leander and A. Poschmann. On the classification of 4 bit S-Boxes. In C. Carlet and B. Sunar, editors, *Arithmetic of Finite Fields*, volume 4547 of *LNCS*, pages 159–176. Springer Berlin Heidelberg, 2007. pages 15, 16

[67] R. Lidl and H. Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1997. pages 15

[68] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. pages 28

[69] M. Medwed, F. Standaert, and A. Joux. Towards super-exponential side-channel security with efficient leakage-resilient prfs. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *LNCS*, pages 193–212. Springer Berlin Heidelberg, 2012. pages 5

[70] T. S. Messerges. *Power analysis attacks and countermeasures on cryptographic algorithms.* PhD thesis, University of Illinois at Chicago, 2000. pages 116

[71] T. S. Messerges. Securing the AES finalists against power analysis attacks. In G. Goos, J. Hartmanis, J. van Leeuwen, and B. Schneier, editors, *Fast Software Encryption*, volume 1978 of *LNCS*, pages 150–164. Springer Berlin Heidelberg, 2001. pages 7

[72] A. Moradi, O. Mischke, and T. Eisenbarth. Correlation-enhanced power analysis collision attack. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 125–139. Springer, 2010. pages 25

[73] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of AES. In *EUROCRYPT*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011. pages 7, 10, 43, 97, 103, 104, 106, 107, 110, 112, 113, 114, 115, 123

[74] NanGate.   The   NanGate   45nm   Open   Cell   Library. http://www.nangate.com. pages 66

[75] S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In P. Ning, S. Qing, and N. Li, editors, *Information and Communications Security*, volume 4307 of *LNCS*, pages 529–545. Springer Berlin Heidelberg, 2006. pages 7, 46, 48, 58

[76] S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011. pages 7, 33, 35, 38, 70

[77] S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In P. Lee and J. Cheon, editors, *Information Security and Cryptology – ICISC 2008*, volume 5461 of *LNCS*, pages 218–234. Springer Berlin Heidelberg, 2009. pages 70

[78] NXP-Semiconductors.   Mifare.   `http://www.mifare.net/en/home/`. pages 1

[79] NXP-Semiconductors.   Security   of   mf3icd40. http://www.mifare.net/en/technology/security/mifare-desfire-d40/. pages 4

[80] N. I. of Standards and T. (NIST). Announcing the advanced encryption standard (AES). Federal Information Processing Standards Publication 197., Nov. 2001. pages 14, 18, 104

[81] D. Oswald and C. Paar. Breaking Mifare DESFire MF3ICD40: Power analysis and templates in the real world. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*, pages 207–222. Springer Berlin Heidelberg, 2011. pages 4

[82] P. Pessl and M. Hutter. Pushing the limits of sha-3 hardware implementations to fit on rfid. In G. Bertoni and J.-S. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 126–141. Springer Berlin Heidelberg, 2013. pages 97, 98

[83] A. Poschmann, A. Moradi, K. Khoo, C.-W. Lim, H. Wang, and S. Ling. Side-channel resistant crypto for less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, 2011. pages 7, 70, 71, 75

[84] E. Prouff. DPA attacks and S-boxes. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption*, volume 3557 of *LNCS*, pages 424–441. Springer Berlin Heidelberg, 2005. pages 20

[85] E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009. pages 118

[86] E. Prouff and T. Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *LNCS*, pages 63–78. Springer Berlin Heidelberg, 2011. pages 7

[87] M. Renauld, F. Standaert, and N. Veyrat-Charvillon. Algebraic side-channel attacks on the AES: Why time also matters in DPA. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *LNCS*, pages 97–111. Springer Berlin Heidelberg, 2009. pages 3

[88] M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems - CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer Berlin Heidelberg, 2010. pages 23

[89] J. J. Rotman. *An introduction to the theory of groups.* Springer-Verlag, 1999. pages 14

[90] M.-J. Saarinen. Cryptographic analysis of all 4 x 4-bit S-boxes. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *LNCS*, pages 118–133. Springer Berlin Heidelberg, 2012. pages 16

[91] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979. pages 31

[92] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949. pages 13

[93] F.-X. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009. pages 116

[94] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, and A. Szekely. Uniform evaluation of hardware implementations of the round-two SHA-3 candidates. In *The Second SHA-3 Candidate Conference, Santa Barbara, USA, August 23-24, 2010*, pages 1 – 16, 2010. pages 98

[95] K. Tiri and I. Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *DATE*, pages 246–251. IEEE Computer Society, 2004. pages 5

[96] E. Trichina, T. Korkishko, and K. Lee. Small size, low power, side channel-immune AES coprocessor: Design and synthesis results. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Advanced Encryption Standard – AES*, volume 3373 of *LNCS*, pages 113–127. Springer Berlin Heidelberg, 2005. pages 7

[97] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 740–757. Springer Berlin Heidelberg, 2012. pages 5

[98] J. Waddle and D. Wagner. Towards efficient second-order power analysis. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *LNCS*, pages 1–15. Springer Berlin Heidelberg, 2004. pages 118

[99] R. Wernsdorf. The round functions of rijndael generate the alternating group. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption*, volume 2365 of *LNCS*, pages 143–148. Springer Berlin Heidelberg, 2002. pages 14

[100] P. Wright. *Spy Catcher: The Candid Autobiography of a Senior Intelligence Officer*. Viking Press, 1987. pages 3

List of Publications

International Journals

[1] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. *Trade-offs for Threshold Implementations Illustrated on AES*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 13 pages, to appear

[2] Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N., and Vitkup, V. *Threshold Implementations of Small S-boxes*. Cryptography and Communications, 7(1):3–33, 2015.

International Conferences

[1] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. *Higher-Order Threshold Implementations*. In Advances in Cryptology – ASIACRYPT 2014, P. Sarkar and T. Iwata, Eds., vol. 8874 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 326–343

[2] De Cnudde, T., Bilgin, B., Reparaz, O., and Nikova, S. *Higher-order Glitch Resistant Implementation of the PRESENT S-Box*. In BalkanCryptSec 2014, 20 pages

[3] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. *A More Efficient AES Threshold Implementation*. In Progress in Cryptology – AFRICACRYPT 2014, D. Pointcheval and D. Vergnaud, Eds., vol. 8469 of Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 267–284

[4] Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., and Yasuda, K. *APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography*, In 21st International Workshop on Fast Software Encryption FSE 2014, Lecture Notes in Computer Science, Springer-Verlag, 2014, 16 pages.

[5] Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., and Van Assche, G. *Efficient and First-Order DPA Resistant Implementations of* KECCAK. In Smart Card Research and Advanced Applications, A. Francillon and P. Rohatgi, Eds., vol. 8419 of Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 187–199.

[6] Bilgin, B., Bogdanov, A., Kneževic, M., Mendel, F., and Wang, Q. *Fides: Lightweight Authenticated Cipher with Side-channel Resistance for Constrained Hardware.* In Cryptographic Hardware and Embedded Systems - CHES 2013, G. Bertoni and J.-S. Coron, Eds., vol. 8086 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 142–158.

[7] Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., and Stütz, G. *Threshold Implementations of All 3x3 and 4x4 S-boxes.* In Cryptographic Hardware and Embedded Systems - CHES 2012, E. Prouff and P. Schaumont, Eds., vol. 7428 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 76–91.

Unpublished Manuscripts

[1] Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., and Yasuda, K. *PRIMATEs*, Submission to the CAESAR Competition, 2014, 20 pages.

[2] De Meyer, L., Bilgin, B., and Preneel, B. *Extended Analysis of DES S-boxes.* KU Leuven COSIC Internal Report, August 2013, 15 pages.

[3] Bilgin, B., Nikov, V., Nikova, S., Rijmen, V., and Stütz, G. *Threshold Implementations of All 3x3 and 4x4 S-boxes.* In Cryptology ePrint Archive, Report 2012/300, 25 pages.

Index

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
COMPUTER SECURITY AND INDUSTRIAL CRYPTOGRAPHY GROUP
Kasteelpark Arenberg 10, bus 2452
B-3001 Leuven-Heverlee
begul.bilgin@esat.kuleuven.be
http://www.esat.kuleuven.be/cosic/