From the Institute for IT Security
of the University of Lübeck
Director: Prof. Dr. Thomas Eisenbarth

Advances in Algorithmic Side-channel Countermeasures for Modern Cryptography

Dissertation

---

for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck
from the Department of Computer Sciences

Submitted by
Okan Seker
from Ankara

Lübeck, 2021

First referee: Prof. Dr.-Ing. Thomas Eisenbarth
Second referee: Prof. Dr. Amir Moradi
Date of oral examination: 18.03.2022
Approved for printing. Lübeck, 21.03.2022

To my beloved family

Abstract

The world has changed with the era of the Internet of Things (IoT) which has evolved the ways of everyday life by connecting our devices. Moreover, the last decade has witnessed a rapid increase in the importance of online-connection and with the help of this connection the modern society is trying to overcome the Covid-19 pandemic. We, as individuals, have separated, while our devices, which become an inseparable part of our life, have connected. This connection has produced an exponentially growing amount of data containing sensitive information. Thus, cryptography that provides privacy and security for our data developed into something essential. Cryptographic algorithms are employed by IoT which makes them as cryptographic devices for these purposes. However the story does not end here as cryptographic devices present adversaries with new attack surfaces.

Modern cryptography started with an adversarial model known as black-box model, where only inputs and/or outputs are known. However this model is not sufficient anymore, since cryptographic devices are accessible by the adversaries. The new adversarial model is known as gray-box model and it implies additional information which depends on the sensitive information being accessible by an adversary. This dependency between sensitive information and such side information has been exploited to recover sensitive information with a little effort. The attacks are divided into two classes passive (Side-channel Analysis (SCA)) and active (Fault Attacks (FA)) attacks. SCA employs passive observations that depend on the processed data or operations such as power consumption or timing behavior to obtain sensitive information, Differential Power Analysis (DPA).

FA interferes with the device and manipulates the behavior of the algorithm, e.g., faulting an intermediate variable or execution order. The most important example of such attacks is Differential Fault Analysis (DFA). As these attacks evolve each day, the countermeasures are widely investigated. The most prominent methods are masking for SCA and redundancy for FA. However an adversary can always implement both attacks simultaneously to obtain more advanced attacks. Moreover the attacks have been pushing the boundaries of our understanding of security. Not only are attacks evolving, but the adversarial models have been evolving as well. The white-box model gives adversaries unlimited control over the execution environment and the cryptographic

implementation. Thus we should consider advanced adversarial models and combined attacks for the current and the incoming cryptographic algorithms.

In this thesis, we delve into design of advanced countermeasures for cryptographic implementations. We provide countermeasures for widely-used cryptographic protocols and yet have not been analyzed in the gray-box model. Moreover we improve well-known countermeasures to resist combined or advanced attacks. Our design methodology combines both theoretical and practical aspects. We provide security proofs alongside practical analysis and formal verification.

First, we focus on improving SCA countermeasures. Our initial aim is to provide a combined countermeasure against SCA and FA. In order to achieve this, we adopt a well-known SCA countermeasure —Secure Multi-party Computation (SMC)— and extend it to resist FA by cooperating with redundancy. We study the effect of faults on masked implementations and present a design where a detectable fault stays detectable until the end of the implementation. Next we propose a combined countermeasure against Differential Computation Analysis (DCA) and Algebraic Differential Computation Analysis (ADCA) two major attacks in white-box model. Nearly all existing white-box implementations in the literature are vulnerable to at least one of these attacks. For both proposed countermeasures, we provide formal security proofs, experimental verification and performance analyses with the corresponding proof-of-concept implementations.

In the second part of the thesis, we focus on attacks and countermeasures for Post-quantum Cryptography (PQC) algorithms. As the possible advent of a quantum computer threatens the security of widely deployed cryptographic schemes, the design of new quantum-resilient alternatives is a pressing task. Motivated by this issue, the US National Institute for Standards and Technology (NIST) is currently holding the PQC Standardization Process, in which Round 3 “finalists” and “alternate candidates” have been recently announced. Among them is Picnic, a signature scheme, which follows Ishai et al.’s MPC-in-the-head paradigm for constructing zero-knowledge proof systems. We show that MPC-in-the-head protocols with or without preprocessing are vulnerable to side-channel attacks due to the protocol itself. Motivated by this vulnerability, we show that MPC-in-the-head protocols can be protected against side-channel attacks in a very natural way. The countermeasures are shown to satisfy provable security notions and are supported by formal verifications. We provide comprehensive leakage analysis using either practical setup or simulation. With practical implementations we show that the resulting overheads are comparably low.

Zusammenfassung

Die Welt wandelt sich in der Ära des Internets der Dinge, das durch die Vernetzung unserer Geräte die Art und Weise unseres täglichen Lebens verändert. Mit Hilfe dieser zunehmenden Vernetzung versucht die moderne Gesellschaft, die Covid-19-Pandemie zu überwinden. Wir als Individuen haben uns weiter voneinander entfernt, während unsere Geräte sich immer enger vernetzt haben. Diese enge Vernetzung führt zu einer exponentiell wachsenden Menge an sensiblen Daten. Daher hat sich die Kryptographie, die den Datenschutz und die Sicherheit unserer Daten gewährleistet, zu einem wesentlichen Faktor entwickelt. Kryptographische Algorithmen werden im Internet der Dinge eingesetzt, was unsere Geräte zu kryptographischen Geräten macht, die besseren Schutz bieten, aber Angreifern auch neue Angriffsflächen bieten.
Die moderne Kryptographie verwendet klassischerweise ein Angreifer-Modell (bekannt als Black-Box-Modell), bei dem nur die Eingaben und/oder Ausgaben bekannt sind. Dieses Modell reicht heutzutage jedoch nicht mehr aus, da die kryptographischen Geräte nun einen direkten physischen Zugriff durch den Angreifer erlauben. Das Gray-Box-Modell berücksichtigt aus physischem Zugriff resultierende Angriffe, indem es dem Angreifer zusätzliche Nebeninformationen gibt, welche von sensiblen Informationen abhängen. Die physischen Angriffe werden in zwei Klassen eingeteilt: passive (Seitenkanalanalyse) und aktive (Fehlerangriffe) Angriffe. Seitenkanalanalyse nutzt passive Messungen von Seiteneffekten, die von den verarbeiteten Daten oder Operationen abhängen, wie z. B. Stromverbrauch oder Zeitverhalten, um sensible Informationen zu erhalten. Fehlerangriffe stören die normale Funnktionalität des Geräts und manipulieren beispielsweise das Verhalten des Algorithmus oder interne Zustände, um aus fehlerhaften Ausgaben informationen zu gewinnen. Da sich diese Angriffe kontinuierlich weiterentwickeln, werden mögliche Gegenmaßnahmen umfassend untersucht und verbessert. Die bekanntesten Methoden sind Maskierung für Seitenkanalanalyse, sowie Redundanz für Fehlerangriffe. Ein Angreifer kann jedoch stets auch beide Angriffe gleichzeitig durchführen, und so ausgefeiltere Angriffe erhalten. Darüber hinaus haben die Angriffe die Grenzen unseres Verständnisses von Sicherheit verschoben. Mit der Verbesserung der Angriffe entwickeln sich auch die Modelle weiter. Das White-Box-Modell beispielsweise gibt den Angreifern unbegrenzte Kontrolle über die kryptografische Implementierung. Daher sollten stets starke Angreifer-Modelle und kombinierte Angriffe für aktuelle und künftige

kryptografische Implementierungen in Betracht geyogen werden.

Diese Arbeit beschäftigt sich mit der Entwicklung von modernen, beweisbaren Seitenkanal-Gegenmaßnahmen für kryptographische Implementierungen. Wir konstruieren neue Gegenmaßnahmen für weit verbreitete kryptografische Protokolle, die bisher noch nicht im Gray-Box-Modell analysiert wurden, konstruiert. Außerdem verbessern wir bekannte Gegenmaßnahmen, die kombinierten oder weiterführenden Angriffen widerstehen. Unsere Entwurfsmethodik kombiniert theoretische und praktische Aspekte. So werden Sicherheitsbeweise mit praktischer Analyse und formaler Verifikation kombiniert.

Im ersten Teil der Arbeit konzentrieren wir uns auf die Verbesserung der aktuellen Seitenkanalanalyse-Gegenmaßnahmen. Unser erstes Ziel ist es, eine kombinierte Gegenmaßnahme gegen Seitenkanalanalyse und Fehlerangriffe zu entwickeln. Um dies zu erreichen, übernehmen wir eine bekannte Seitenkanalanalyse-Gegenmaßnahme—Secure Multi-party Computation—und erweitern sie, um auch Fehlerangriffen zu widerstehen, indem wir Redundanz hinzufügen. Wir untersuchen die Auswirkungen von Fehlern auf maskierte Implementierungen und stellen ein Design vor, bei dem ein erkennbarer Fehler bis zum Ende der Implementierung erkennbar bleibt. Als nächstes schlagen wir Gegenmaßnahmen für das stärkste Angreifersmodell vor: das White-Box-Modell. Wir erreichen Sicherheit gegen die zwei wichtigsten Angriffe in diesem Modell: Differential Computation Analysis und Algebraic Differential Computation Analysis. Fast alle bekannten White-Box-Implementierungen in der Literatur können mit diesen angriffen gebrochen werden. Für beide Gegenmaßnahmen werden sowohl formale Sicherheitsbeweise als auch experimentelle Verifikation und Leistungsanalysen mit den entsprechenden Proof-of-Concept-Implementierungen vorgestellt.

Im zweiten Teil der Arbeit konzentrieren wir uns auf Angriffe und Gegenmaßnahmen für Post-Quantum-Kryptographie-Algorithmen. Da das mögliche Aufkommen eines Quantencomputers die Sicherheit weit verbreiteter kryptographischer Verfahren bedroht, ist die Entwicklung neuer quantenresistenter Alternativen eine dringende Aufgabe. Aus diesem Grund führt das US National Institute for Standards and Technology (NIST) derzeit den PQC-Standardisierungsprozess durch, bei dem kürzlich “Finalisten” und “Alternativkandidaten” der Runde 3 bekannt gegeben wurden. Unter ihnen ist Picnic, ein Signaturverfahren, das dem MPC-in-the-head-Paradigma von Ishai et al. zur Konstruktion von Zero-Knowledge-Proof-Systemen folgt. Wir zeigen, dass MPC-in-the-Head-Protokolle mit oder ohne Preprocessing aufgrund des Protokolls selbst anfällig für Seitenkanalangriffe sind. Motiviert durch diese Verwundbarkeit zeigen wir, dass MPC-in-the-Head-Protokolle auf sehr natürliche Weise gegen Seitenkanalangriffe geschützt werden können. Es wird gezeigt, dass die Gegenmaßnahmen beweisbaren

Sicherheitsbegriffen genügen, was zusätzlich noch durch formale Verifikation unterstützt wird. Wir bieten eine umfassende Leakage-Analyse, die entweder durch einen praktischen Aufbau oder eine Simulation durchgeführt wird. Mit praktischen Implementierungen zeigen wir, dass die resultierenden Overheads vergleichsweise gering sind.

Acknowledgements

"The Road goes ever on and on
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can,
Pursuing it with eager feet,
Until it joins some larger way
Where many paths and errands meet.
And whither then? I cannot say"

– J.R.R. Tolkien, *The Fellowship of the Ring*

This is the story of 6 years of work in my life spreading over three countries. I started this journey with a lot of hope and dreams also with unknowns and nightmares. Sometimes it was a hurdle to continue but as Bilbo said, going out our door is a dangerous business and if we don't keep our feet, there's no knowing where we might be swept off to and that is what I did. I just kept my pace. This is a story of failed experiments, unaccepted papers, and abandoned ideas, nevertheless, this is a story of not stepping back, sleepless nights and persistence. And I know that this is just the beginning.

First off all I would like to thank Thomas Eisenbarth for giving me this opportunity and trusting me (twice!). He is an excellent advisor, mentor and most importantly a good friend. And I am grateful for his endless patience and encouragement whenever I stumble throughout this research.

I also thank Maciej Liśkiewicz, for his wisdom and guidance. Our weekly meetings were a milestone for this thesis. And I am glad that I had the opportunity to express myself every week and discuss the research questions.

Special thanks to Sebastian Berndt for his guidance, friendship and endless help during the last two years of my life. I am grateful for his help that resulted in an amazing acceleration in this research. I cannot imagine if we had more time and would like to quote: *So much universe, and so little time* by Terry Pratchett.

I am glad that I have such a great team at the University of Lübeck, Institute of IT Security. Without such a special team this work wouldn't be happened. They are always there for me whenever I need support or a coffee break. I would like to thank Luca Wilke for his amazing cooperation on the practical aspect of this work (please don't break up with the oscilloscope!), Thore Tiemann for answering all of my engineering related or *nonsense* questions, Ida Bruhns for being an awesome friend and officemate and many thanks to Claudius Pott, Esfandiar Mohammadi, Florian Sieck, Jan Wichelmann, and Ines Schiebahn. And of course I should thank our honorable member Samira Briongos. I thank Diego Aranha, Marc Gourjon, Akira Takahashi, and Greg Zaverucha for being an excellent colleague. It was an amazing experience for me to work with you even if it was all done remotely. I would like to thank my students; Tim Gellersen, Pajam Pauls and Alexander Treff for working on their thesis with me. I also would like to thank my original mentors; Ali Doğanaksoy, Muhiddin Uğuz and excellent companies Murat Burhan İlter, Onur Koçak and Ziya Akcengiz. I also thank to my colleagues in Worcester Polytechnic Institute, as they are an inseparable part of this story.

I would like to thank my *best persons*, my dearest friends, who stayed with me and backed me whenever I needed to; Dilşad Susam, Ela İstanbullu, Elif Kuyucu, Gizem Şentürk, Irmak Sancar, and Merve Yurtcan (Arkadaşlar önce bi Ela'yı oylayalım). We might feel like we *wander* through space but we are not –and will never be– *lost*.

Ve tabi ki yanımdan ayrılmayan ve beni desteklemekten asla vazgeçmeyen Çınar ve Şeker ailesi üyelerine sonsuz teşekkürler etmek istiyorum. Bu kocaman ailenin bir üyesi olmak benim için büyük bir onur ve hepinizi çok seviyorum! Her zaman benimle olduğunuz için, beni asla yalnız bırakmadığınız için ve en önemlisi, sadece bu yolculukta değil, hayatta attığım her adımda güç verdiğiniz için size çok teşekkür ediyorum.

Finally, I thank my wife and best friend, Hazal. This work would not be possible without her help and endless support; *to infinity and beyond.* I am proud of what we have achieved and eager to see what comes next. Finally I would like to thank Buddy for being a wonderful companion. With his help we were able to keep our sanity during lockdowns.

Okan Seker
November 2021

Contents

II Attacks and Countermeasures for Post-quantum Cryptography

List of Figures

List of Tables

1

Introduction

"A box without hinges, key, or lid,
Yet golden treasure inside is hid."

– Bilbo Baggins, *The Hobbit*

Although this is not a hard riddle, Bilbo Baggins asked to gain time against Gollum in *The Hobbit* by J. R. R. Tolkien [Tol37]. The answer is *eggs* as it *clearly* satisfies the riddle. More importantly, the only way to get the *golden treasure* is to *crack* the egg. Surprisingly, the embedded devices, on which today's world depends, meet the same conditions. The modern world requires us to live in a connected world where not only us but our devices are linked and reside in a hostile environment. Embedded devices store our *golden treasures*, our secrets or sensitive data, without a key[1] or lid and there is always an adversary who tries to *crack* the shell and reach the *golden treasure*.
The information age which started in mid-20^{th} century, accelerated the advances in cryptography with the help of modern computer science and bright minds such as Alan Turing and Claude Shannon. At first, cryptography was considered a part of military science. But everything changed with the commercialization of computers and the importance of cryptography has been increasing rapidly since then.
Commercialization of computers evolved into the age of Internet of Things (IoT) in which all the devices such as mobile phones, cars, refrigerators are connected, and the need for modern cryptography to protect the privacy becomes necessary. Therefore we are obligated to use the cryptography to attain the desired security level and objectives such as confidentiality, authentication, data integrity and non-repudiation. Here is a brief description of cryptographic goals as listed in [MvOV96].

- **Confidentiality** is the functionality of keeping the information from all but legitimate users.
- **Authentication** is a service of rightful identification. The parties that communicate should be able to correctly identify each other (entity authentication) or an

[1] Here *key* means a physical key.

information received from a channel should be authenticated with respect to origin, content and time etc. (data origin authentication).

- **Data integrity** addresses the unauthorized modification of data including insertion, deletion, and substitution.
- **Non-repudiation** prevents a user from modifications to a committed action. This hinders parties to deny their actions.

Cryptography enables us to achieve the desired properties in the form of mathematical expressions that uses an input with a *secret key*, as simple as in Eq. (1.1), where $\mathbf{E}$ is a publicly available function, p and c are an input/output pair and k is a *secret key*. The cryptographic algorithm is known and publicly available while the only unknown (or the secret) is the *secret key* [Sha49].

$$\mathbf{E}(k, p) = \mathbf{E}_k(p) = c. \tag{1.1}$$

Depending on the usage of the secret key, cryptographic algorithms are classified into symmetric and asymmetric cryptography. The first one (*symmetric cryptography*) uses the same key for both encryption and decryption, such as the Advanced Encryption Standard (AES) [DR20] while the latter one (*asymmetric cryptography*, also known as public-key cryptography) uses two different keys for encryption and decryption such as RSA (named after its creators: Rivest, Shamir and Adleman) [RSA83].
A cryptographic algorithm's strength relies on its mathematical foundation. Here a secure algorithm essentially means that the only *possible* way of getting correct ciphertext and plaintext pairs is by trying all the possible key candidates i.e. using a brute force attack. Modern cryptographic algorithms need thousand of years to be broken by brute force attacks (e.g. the simplest AES has a key space of 2^{128}), therefore the main aim of cryptanalysis is to reduce the key space by analysing the algorithm such that a relatively small key space leads to a relatively practical brute force attack that results in a reasonable time.
However the security in theoretical foundation becomes insufficient when these algorithms are applied to real-world applications such as microcontrollers, IoT devices and personal computers. The devices that run a cryptographic algorithm and store cryptographic keys (i.e. *secret keys*) are called *cryptographic devices* [MOP07]. As these devices become an essential part of our life, new adversarial models and attack surfaces appear. Therefore, cryptographic devices need to be analyzed with respect to their implementation of the cryptographic algorithm in addition to its mathematical foundation.

The adversarial objective is to undertake the cryptographic algorithm or a device that runs an algorithm to recover the secret information. However the abilities of the adversary and assumptions depend on the adversarial models that are expanded below.

1.1 Adversarial Models

In the following we introduce the three main adversarial models in the literature; the black-box model, the gray-box model and the white-box model classified according to the information accessible by the adversary.

1.1.1 Black-box Model

The first model used in the literature is the black-box model. In this model the adversary is able to access *only* the plaintext-ciphertext (input-output) pairs. Although the whole algorithm except the secret key is known [Ker83], the adversary cannot receive any additional information or characteristics. In this model the attacks are implementation-free that is, the analysis depends *only* on the mathematical structure of the algorithm as seen in Fig. 1.1. Until late 90s, the security assessment of cryptographic implementations was measured by this model. Using this model, two powerful attacks emerged: differential [BS91] and linear cryptanalysis [Mat93]. The first one studies the affect of certain mappings of input differences to the output differences, and the second one exploits the non-uniformity of linear expressions involving input bits, output bits and key bits. Here is the list of attacks that can be implemented by an adversary.

1. *Ciphertext only attack:* The adversary accesses only the ciphertext and tries to reveal the secret using only this information e.g. linear cryptanalysis.

2. *Known-plaintext attack:* The adversary knows a *arbitrary* set of plaintexts and receives the corresponding ciphertexts.

3. *Chosen-plaintext attack:* The adversary chooses a *specific* set of plaintexts and receives the corresponding ciphertexts. It is used in differential cryptanalysis.

4. *Chosen-ciphertext attack:* The adversary chooses a *specific* set of ciphertexts and receives the corresponding plaintexts.

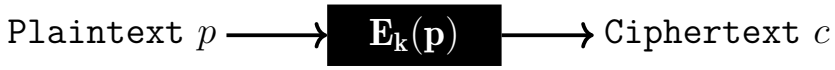


Figure 1.1: A brief summary of black-box model. The adversary can only access the plaintext-ciphertext (or input-output) pairs.

1.1.2 Gray-box Model

The second model is the gray-box model which gained attention after the seminal work by Kocher et al. [Koc96]. The model covers the implementation security of devices that run a cryptographic algorithm. The adversary may invoke the implementation multiple times and adaptively choose inputs (e.g. plaintext or ciphertext) and record any side-channel information such as the power consumption or the timing behavior of the device while running the implementation as summarized in Fig. 1.2. Moreover the adversary can interfere with the implementation by means of injecting faults or tampering with the environment such as high or low temperature. Therefore this model leads to a widely accepted attacker model that deals not only with the mathematical structure of the cryptographic algorithms but also the physical implementation of it.
The first attack that uses this model was introduced in 1996 by Kocher et al. [Koc96] where the timing behavior of a crypto system depends on the secret key and thus an analysis of the timing of the operations reveals the secret key. Three years later, Kocher et al. [KJJ99] showed that the power consumption or the electromagnetic radiations of a device can also be exploited to recover the secret key. As stated earlier, the model also allows an adversary to inject faults, interfere the operations and collect faulty ciphertext. The impact of such attacks is shown in the literature with the works on AES [BDL97] and RSA [Muk09] where even a single faulty ciphertext can lead to a key recovery. In Chapter 2, we delve into these attacks and give more detailed descriptions.

1.1.3 White-box Model

The third model, the white-box model, is presumably the strongest adversarial model. White-box cryptography promises implementation security of cryptographic services in pure software solutions, mainly by protecting keys and intermediate cipher states through layers of obfuscation. While white-box cryptography is successfully sold by several companies as one ingredient of secure software solutions (e.g. [Gem]), analysis of deployed solutions is lacking, as is a sound theoretical framework to analyze white-box implementations.
The model assumes the cryptographic primitive to run in an untrusted environment

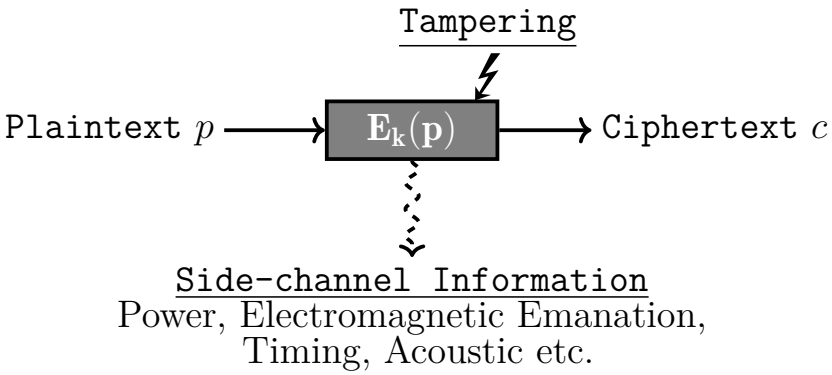


Figure 1.2: A brief summary of the gray-box model. The adversary can access plaintext-ciphertext (or input-output) pairs and receive additional information (side-channel information) while the device operates on the input. Moreover the adversary can tamper with the implementation and observe effects of the tampering.

where the adversary has complete control over the implementation. As successor of the gray-box model, the adversary has all the capabilities of a gray-box model and additionally may read and modify every memory access or intermediate state and can interrupt the implementation at will as given in Fig. 1.3.

White-box cryptography was introduced in 2002 by Chow et al. [CEJVO03b,CEJvO03a]. The main idea of their scheme is to represent a cryptographic algorithm as a network of look-up tables and key-dependent tables. In order to protect the key dependent tables, Chow et al. proposed to use *input and output encodings*. Although the method provides security guarantees for individual tables, the combinations of protected tables still leaks information [BGEC05]. In fact, all published academic proposals for White-box cryptosystems (WBC) [Kar10, BCD06, LN05, XL09] have been practically broken [BGEC05, DMWP10, LRDM$^+$14, WMGP07].

Another hit for the white-box cryptography is the application of attacks from gray-box model. Side-channel information such as observable intermediate states leads to attacks called Differential Computation Analysis (DCA) [BHMT16]. These attacks allows an adversary to perform an automated extraction of the secret key from a white-box implementation faster and without specific knowledge of the design.

In order to close the gap the between the secure designs and the attacks, white-box cryptography competitions (or challenges) are organised throughout the years [TWC17, TWC19, TWC21]. Although all the submitted designs are broken quickly (Most of the designs lasts less then one day), excellent attacks and design ideas emerged from these challenges. The most important attack that surfaced is Algebraic Differential

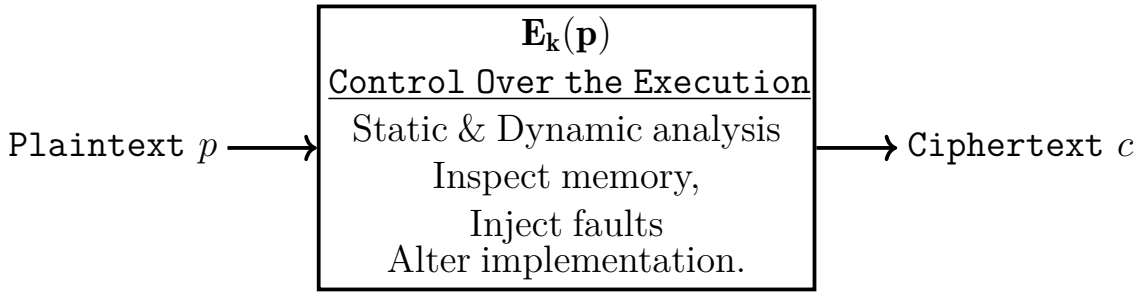


Figure 1.3: A brief summary of white-box model. The adversary not only access the input-output pair but also all the information within the implementation i.e. all the internal information about a cryptographic implementation.

Computation Analysis (ADCA) which easily breaks even protected implementations [BU18, GPRW19]. As a result new design methodologies and security notions are started to be introduced [BU18, GRW20].

1.2 Summary of Contribution and Road Map

As smart devices become more and more inevitable part of our lives, the protection of sensitive information becomes essential. The security of the embedded devices are leaned on cryptographic algorithms in order to use them securely. Unfortunately, we cannot rely on old cryptographic security assumptions as these devices are effortlessly accessible by the adversary and the technology (hardware and software) behind them becomes cheaper. Therefore we need better countermeasure designs that ensure security against advanced or combined attacks and can evolve as the attacks.

The last decade witnessed stunning series of attacks. Modern processors that exist nearly on all the IoT devices are shown to be vulnerable against hardware-level physical attacks such as Spectre [KHF+19] and Meltdown [LSG+18]. The designated secure modules [MSEH20] and hardware enclaves are shown to be vulnerable against such attacks [BMW+18]. As we explore the cause of these attacks, we uncover the vulnerabilities and push the security boundaries one step forward at a time [WWME20, WWSE21, BOM+19, CGG+19, SLM+19].

Adversaries equipped with such an advance (yet reachable) equipment can push the boundaries of attacks. For example, laser-assisted attack techniques give the adversary such a power that makes us question the well-defined security notions [KGM+20].

This work is motivated by the need for strong, reliable and provable countermeasures as the attacks are evolving each day. In the first part of this work, we boost the state

of art countermeasures to thwart more advanced attacks or combined attacks. In the second part we provide countermeasures for unprotected implementations that aim at the post-quantum era i.e. Post-quantum Cryptography.

1.2.1 Improving Side-channel Countermeasures

In Chapter 2, we introduce the details of SCA with countermeasures and ways of improving them. As the impact of SCA and FA cannot be ignored and cannot be stated out-of-scope, there exists several countermeasures to counteract these attacks individually.

Extending the Secure Multi-party Computation to Resist Fault Attacks.
An adversary can always implement active and passive attacks in a combined manner. One idea to eliminate combined attacks is to use different countermeasures to resist both attacks individually. However, the synergy between these countermeasures is not always given and this methodology can leak information [REB+08, LFZD14]. One countermeasure that we focus is Secure Multi-party Computation. SMC is shown to have fault resistant property, yet this property has never been analyzed in that context. Therefore our first research question is:

> *Can we provide a combined countermeasure built on secure multi-party computation that can be efficiently implemented while preserving the information-theoretic security?*

In Chapter 3 we deep dive into this problem and provide a combined countermeasure. At CHES 2011, Roche and Prouff applied secure multi-party computation to prevent side-channel attacks. While multiparty computation is known to be fault-resistant as well, the particular scheme used for side-channel protection does not currently offer this feature.
This chapter introduces a new secure multiparty circuit to prevent both fault injection attacks and side-channel analysis. The new scheme extends the Roche and Prouff scheme to make faults detectable. Arithmetic operations have been redesigned to propagate fault information until a new secrecy-preserving fault detection can be performed. A new recombination operation ensures randomization of the output in the case of a fault, ensuring that nothing can be learned from the faulty output. The security of the new scheme is proved in the ISW probing model [ISW03], using the reformulated t-SNI security notion [BBD+16]. Besides the new scheme and its security proof, we also present an extensive performance analysis, including a proof-of-concept, software-based AES

implementation featuring the masking technique to resist both *fault and side-channel attacks at the same time*. The performance analysis for different security levels are given for the ARM-M0+ MCU with its memory requirements. A comprehensive leakage analysis shows that a careful implementation of the scheme achieves the expected security level.
This work has been published by Okan Seker, Abraham Fernandez-Rubio, Thomas Eisenbarth and Rainer Steinwandt at IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES 2018) [SFRES18]. A reference microcontroller implementation is provided and published by the co-author Abraham Fernandez-Rubio [Rub17].

A Masking Scheme for White-box Designs.

In Chapter 4, we study a new protection mechanism for white-box designs. Protecting secrets purely in software is a great challenge, especially if a full system compromise is not simply declared out-of-scope of the security model. With fully homomorphic encryption still complex and computationally expensive [MOO+14] and secure enclaves being notoriously buggy at this time [VBPS17, MIE17, BMW+18], industry may opt for white-box cryptosystems (WBC) or even be required to do so by industry standards like EMVCo [Pay, BBF+19].
Due to their high utility, white-box cryptosystems (WBC) are deployed by the industry even though the security of these constructions is not well defined. A major breakthrough in generic cryptanalysis of WBC was Differential Computation Analysis (DCA), which requires minimal knowledge of the underlying white-box protection and also thwarts many obfuscation methods [BHMT16]. To avert DCA, classic masking countermeasures originally intended to protect against highly related side-channel attacks have been proposed for use in WBC. However, due to the controlled environment of WBCs, new algebraic attacks against classic masking schemes have quickly been found [GPRW19]. These algebraic DCA attacks break all classic masking countermeasures efficiently, as they are independent of the masking order. To sum up, although there exist informal ideas on how to create a secure white-box design that can resist both computational and algebraic attacks, formal and generic constructions with a security analysis are missing. Therefore our second question is:

> *Can we introduce a combined countermeasure based on basic masking in such a way that it achieves information-theoretic security against Differential Computation Analysis and Algebraic Differential Computation Analysis?*

In this chapter, we propose a novel generic masking scheme that can resist both DCA and algebraic DCA attacks. The proposed scheme extends the seminal work by Ishai et al. [ISW03] to also resist algebraic attacks. To prove the security of our scheme, we demonstrate the connection between two main security notions in white-box cryptography: *probing* and *prediction security*. Resistance of our masking scheme to DCA is proven for an arbitrary order of protection, using the strong non-interference notion [BBD+16]. Our masking scheme also resists algebraic attacks, which we show concretely for first and second-order algebraic protection. Moreover, we present an extensive performance analysis and quantify the overhead of our scheme, for a proof-of-concept protection of an AES implementation.
This work has been been published by Okan Seker, Thomas Eisenbarth and Maciej Liśkiewicz at the IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES 2021) [SEL21].

1.2.2 Attacks and Countermeasures for Post-Quantum Schemes

Many public-key protocols used in modern systems such as RSA or the Diffie-Hellman key exchange have shown to withstand attacks run on classical computers. On the other hand, it is long known that attackers equipped with quantum computers are provably able to break these systems [Sho99]. While the deployment of practical quantum computers seems to be unlikely within the next decade, the cryptographic community has already started to develop algorithms that withstand these quantum attackers. To standardize some of these algorithms and to foster their timely adoption, NIST has started their *Post-Quantum Cryptography Standardization* project in 2017, where 82 algorithms were submitted, 69 proceeded to the first round, 26 to the second round, and 15 in the third round [CCJ+16, NIS20a, NIS20b, AASA+20]. The standardization focuses on key encapsulation mechanisms (KEMs), from which public-key encryption schemes can be derived easily, and on signature schemes due to their widespread use in modern systems.
While the security of most of these algorithms relies on problems revolving around lattices or coding theory that are believed to be intractable for quantum computers, the signature scheme Picnic [CDG+17] relies on the hardness of SHA-3 and a low-complexity symmetric cipher called LowMC [ARS+15]. The underlying zero-knowledge protocol of Picnic follows the MPC-in-the-head paradigm and is called ZKB++ which is an optimised version of ZKBoo [GMO16]. We will use the ZKBoo protocol as an example both for our attacks on the general MPC-in-the-head paradigm as well as for our defence mechanisms.
In Chapter 5 we introduce the required background for Post-quantum Cryptography

and a general overview of side-channel attacks on the algorithms within the NIST PQC project. We focus on the side-channel analysis of the Picnic Signature Scheme, an alternate candidate in the ongoing project for post-quantum cryptography by the NIST. Since the seminal work by Ishai et al. [IKOS07], the MPCitH paradigm has been actively studied over the past decade. In particular, two closely related protocols ZKBoo [GMO16] and ZKB++ [CDG+17] brought MPCitH closer to practice, leading to the submission of Picnic1 to Round 1 of the NIST PQC Standardization Process. Katz, Kolesnikov, and Wang [KKW18] extended the paradigm to MPCitH-PP and corresponding version, Picnic2, was added during Round 2. Kales and Zaverucha [KZ20] further optimized Picnic2 from various implementation aspects and accordingly proposed Picnic3. Yet, despite such advantages with regard to the underlying assumptions, implementations of MPCitH-type signatures e.g. Picnic Signature Scheme, are not immune to side-channel attacks that threaten unprotected software.

In Chapter 6 we show that a direct implementation of MPCitH signing is susceptible to a probing side-channel attack, which is in practice realized by, e.g., differential power analysis (DPA). The attack proposed by Gellersen et al. [GSE21] is devastating – it allows a side-channel attacker to successfully recover the secret signing key, after observing no more than 30 signatures. To understand their attack, it is useful to review the underlying "commit-and-open" approach typically used by MPCitH proofs. The attacks target the three-round ZK proof system underlying some Picnic instances, known as ZKB++ [CDG+17] (an optimized variant of ZKBoo [GMO16]).

MPCitH proof systems like ZKB++ follow a typical *commit, challenge, response* structure (which, in fact, can be seen as a Σ-protocol [Dam10]), and prove knowledge of a *witness* $\mathbf{w}$ that satisfies a *statement* for a hard relation. For example, we can prove knowledge of a key (the witness) that relates a plaintext-ciphertext pair (the statement) for a block cipher (the relation). The ZKB++ proof system is built on a three-party MPC protocol Π_f for a function f implementing this hard relation (the block cipher in our example). To initialize the protocol, the prover P first additively secret-shares the witness such that $\mathbf{w} = \mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3$, and considers each share $\mathbf{w}_i$ as a private input to party P_i. Then P runs Π_f "in the head", i.e., it simulates a run of the protocol between the three parties, and produces for each P_i a commitment to its *view*, i.e., the input $\mathbf{w}_i$, output, communication, and random tape. These three commitments are sent to the verifier V, who replies with a challenge, for P to open two of them. Finally, V checks the opened values for consistency and for an accepting output. The (honest verifier) ZK property of the protocol is guaranteed by the 2-*privacy* of Π_f, i.e., it is possible to simulate up to 2 parties' views given their inputs, and the output of the protocol

execution. But once a side-channel adversary probes the unopened party's view, in particular their share of the witness, the adversary can recover the secret witness.

Fusing Strong Non-interference with MPC-in-the-Head Paradigm.

This analysis shows there is a need for provable security that can work with MPC-in-the-Head Paradigm. In Chapter 7 we investigate the protection mechanism's against side-channel attacks on MPC-in-the-Head Paradigm and ask the question:

> *Can we improve* ZKBoo *resp. MPC-in-the-Head paradigm in general in such a way that the unopened view is provably secure even if the opened views are reachable by an adversary?*

To thwart this attack, we proposed the *SNI-in-the-head* (SNIitH) approach, which naturally retains the privacy of MPC protocol, but also adds strong non-interference (SNI) [BBD+16] security, which is a strong provable security notion for ISW-style masking countermeasures (as defined by Ishai-Sahai-Wagner [ISW03]). In the SNIitH approach, the number of parties is generalized to N and the underlying MPC protocol is assumed to have $(N-1)$-privacy, so that the views of up to $N-1$ parties may be safely revealed. However, instead of opening $N-1$ parties as a typical MPCitH prover would, an SNIitH prover only reveals $N-t-1$ parties as a response, where the parameter t serves as the "buffer" for probing security. This way, the prover makes sure that at least one party's internal state remains completely hidden, even if the side-channel adversary observes up to t variables during the execution of MPC protocol Π_f. Relying on this idea, we generalized ZKB++, and gave a variant of Picnic that is provably secure against probing adversaries. However, as we shall see below, there are still several practical challenges which seem hard to overcome with SNIitH.

MPC-in-the-head based protocols have recently gained much popularity and are at the brink of seeing widespread usage. With widespread use come the spectres of implementation issues and implementation attacks such as side-channel attacks. As described above the implementations of protocols implementing the MPC-in-the-head paradigm are vulnerable to side-channel attacks. As a case study, we choose the ZKBoo-protocol of Giacomelli, Madsen, and Orlandi [GMO16] and show that even a single leaked value is sufficient to break the security of the protocol.

In order to remedy this situation, we extend and generalize the ZKBoo-protocol by making use of the notion of strong non-interference of Barthe et al. [BBD+16]. To apply this notion to ZKBoo, we construct novel versions of strongly non-interfering gadgets

that balance the randomness across the different branches evenly. Finally, we show that each circuit can be decomposed into branches using only these balanced strongly non-interfering gadgets. This allows us to construct a version of ZKBoo, called $(n+1)$-ZKBoo which is secure against side-channel attacks with limited overhead in both signature-size and running time. Furthermore, $(n+1)$-ZKBoo is scalable to the desired security against adversarial probes. We experimentally confirm that the attacks successful against ZKBoo no longer work on $(n+1)$-ZKBoo. Moreover, we present an extensive performance analysis and quantify the overhead of our scheme using a practical implementation.
This work has been published by Okan Seker, Sebastian Berndt, Luca Wilke, Thomas Eisenbarth at the ACM Conference on Computer and Communications Security (CCS 2020) [SBWE20].

Masking MPC-in-the-Head Paradigm with Preprocessing.

In Chapter 8, we push the security one step forward and consider security against MPC-in-the-head paradigm with preprocessing used in Picnic3 signature scheme. As shown in Chapter 6 known countermeasures are not sufficient when the MPC protocol uses a preprocessing phase, as in Picnic3. To the best of our knowledge, no previous work explored the possibility of masking the newer *MPC-in-the-head paradigm with preprocessing (MPCitH-PP)* of Katz, Kolesnikov, and Wang (KKW, [KKW18]), which produces much more compact proofs (and shorter signatures in turn). KKW is used in the latest version of Picnic, Picnic3 [KZ20], and in a similar signature scheme BBQ [dDOS19]. The preprocessing phase is independent of the witness, and is used by the parties to establish correlated random values, such as multiplication triples, that they can use during the MPC protocol to improve efficiency. Since it happens before the main, witness-dependent MPC protocol, the preprocessing phase is also called an *offline* phase, and the main part is called the *online* phase. We first observe that the preprocessing phase provides additional attack surface not present in MPCitH protocols without preprocessing in Chapter 6. In the KKW protocol, the prover first simulates many instances of the MPC protocol consisting of offline and online phases, and commits to each party's view in both phases. Then for each instance of the MPC protocol, the prover opens either the offline phase or the online phase, depending on the challenge sent from the verifier. To open the offline phase, the prover reveals the views of all N parties (which is secure, since preprocessing views are independent of the witness). For the online phase, the prover reveals the views of $N-1$ parties (which is secure by the $N-1$ privacy of the MPC protocol). In the former scenario, since the offline phase

contains the correlated random values used by all parties during the online phase, a probing adversary can break the privacy of the underlying MPC protocol by probing values from the corresponding online phase (which is computed by the prover but *not* revealed).
We note that the attack succeeds *for any number of opened views*, since MPCitH-PP inherently relies on revealing all views for the offline phase. This indicates that the SNIitH approach cannot mitigate this type of probing attack. We are motivated to design an alternative countermeasure addressing the following question:

> *Can we mask signature generation in signature schemes constructed with the MPC-in-the-head-with-preprocessing paradigm in a provably secure manner, without modifying the verification algorithm?*

We overcome these challenges by showing how to mask the underlying zero-knowledge proof system [KKW18] for any masking order, and by formally proving that our approach meets the standard security notions of non-interference for masking countermeasures. As a case study, we apply our masking technique to Picnic. We then implement different masked versions of Picnic signing providing first order protection for the ARM Cortex M4 platform, and quantify the overhead of these different masking approaches. We carefully analyze the side-channel risk of hashing operations, and give optimizations that reduce the CPU cost of protecting hashing in Picnic by a factor of five. The performance penalties of the masking countermeasures ranged from 1.8 to 5.5, depending on the degree of masking applied to hash function invocations.
Although our masked implementation focuses on Picnic3, which is instantiated with KKW and the LowMc circuit [ARS+15], our generic approach in Section 8.2 also applies to BBQ (KKW instantiated with the AES circuit) [dDOS19] and Baum and Nof's variant of KKW (instantiated over an arithmetic circuit for proving SIS instances) [BN20].
This work have been published by Diego F Aranha, Sebastian Berndt, Thomas Eisenbarth, Okan Seker, Akira Takahashi, Luca Wilke, Greg Zaverucha at the IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES 2021) [ABE+21] where Okan Seker, Akira Takahashi share the first-authorship and the implementation is provided by the co-authors.

Part I

Improving Side-channel Countermeasures

2

Introduction to Physical Embedded Security

In this chapter we focus on the background to understand and elaborate the side-channel attacks and countermeasures. We provide a detailed explanation of passive attacks, active and combined attacks. Each attack sections are followed by the corresponding countermeasure descriptions, security assessments and formal security notions. Moreover we link these attacks and countermeasures to the white-box model.
The modern Side-channel Analysis (SCA) and Fault Attacks (FA) and is a common threat to cryptosystems if the adversary has physical access to the implementation. A wide range of such attacks have been shown to circumvent security assumptions and reveal cryptographic keys, often with little effort, especially if no special precautions were taken during implementation. Physical attacks can be divided into classes; *active attacks* and *passive attacks*, through the thesis we use the terms Fault attacks and Side-channel attacks to describe the attacks respectively. Active attacks manipulate the target implementation and the resulting faulty output can then reveal information about the state and the key [BDL97, BECN+06]. On the other hand, passive attacks employ data side-channel sources such as power [KJJR11], sound [GST14], or electromagnetic emanation [GMO01] of a target implementation.

2.1 Side-channel Analysis

Side-Channel attacks assume the gray-box adversarial model as described in Section 1.1.2. The main idea is to observe the behavior e.g. power consumption of the device, of the cryptographic implementation without actively effecting/changing the behavior of the

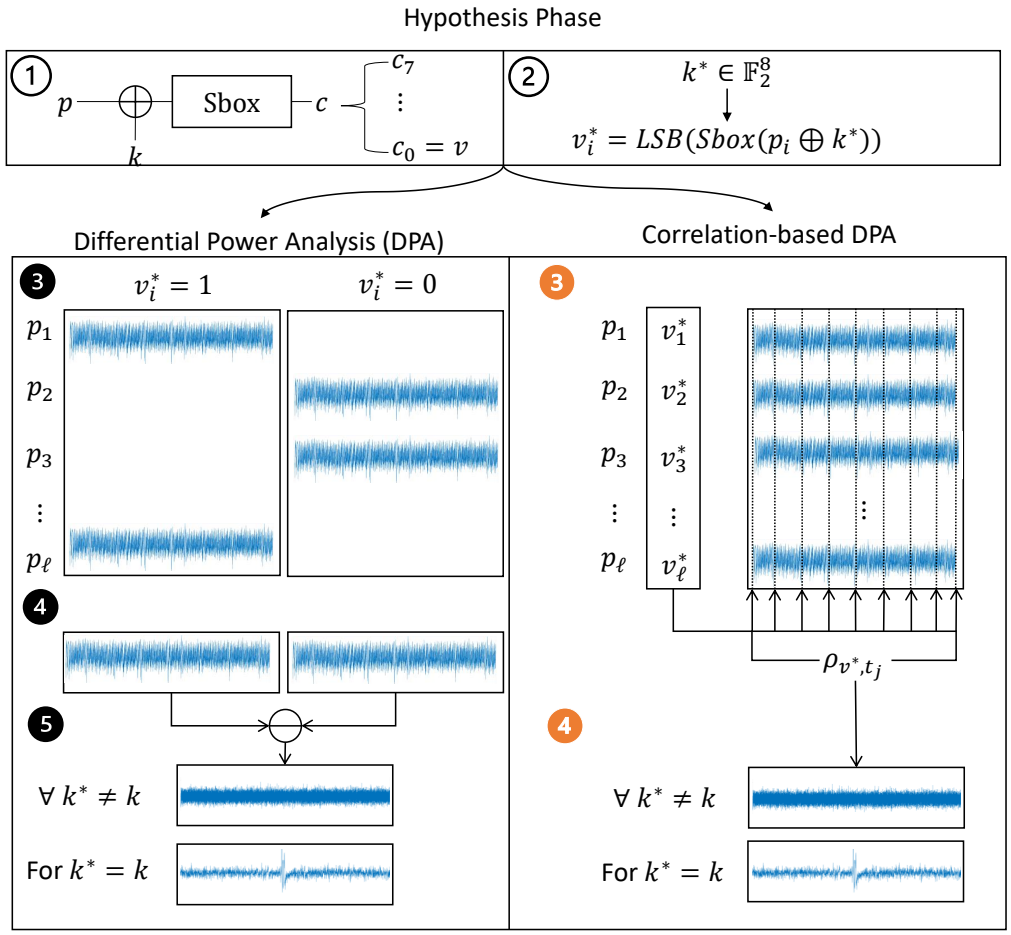


Figure 2.1: A brief summary of DPA and Correlation-based DPA. The attacks share a common hypothesis phase (1 and 2), however the exact attacks have small variations. DPA uses steps 3, 4 and 5 (left side), while CPA uses the steps 3 and 4 (right side).

device. This attack requires little knowledge about the implementation and can be done without much effort. As they are passive, the attacks are hard to detect. The primary motivation behind this attacks is side-channel information produced by the targeted device depends on the processed data i.e. depends on a secret key that is used and the input that is assumed to known by the adversary. Therefore, in an unprotected implementation side-channel information weaken the security of the device. The most common attacks are Simple Power Analysis (SPA) and Differential Power Analysis (DPA) using power analysis i.e. the instantaneous power consumption depending on the data [KJJ99]. Later, different type of analysis methods were introduced in the literature such as Template Attack (TA) [CRR02] and Correlation Power Analysis (CPA) [BCO04].

Moreover, recent trends combines machine learning with side-channel analysis to recover secret keys from even protected implementations [BPS+20].

We start with the description of DPA introduced by Kocher et al [KJJ99] to capture the essence of the attacks. In the DPA setting, the adversary is assumed to know the targeted algorithm. Thus an intermediate variable v which depends on a known input p (such as a *plaintext* or a *ciphertext*) and secret value k (such as a *secret key*) can be selected using a function f such that $v = f(p, k)$. Moreover the adversary chooses a function ϕ that models the side-channel information (a common model is a single bit [MOP07]). Then the adversary collects multiple side-channel traces denoted by $(T_1, \ldots, T_\ell)$ of the device with different inputs $(p_1, \ldots, p_\ell)$ and with the same secret key k. Remark that each side-channel trace T_i corresponds to a vector of length N i.e. $T_i = \{t_1^i, \ldots, t_N^i\}$ where t_j^i represents the measured information in the j^{th} instance (or j^{th} sample point) during the i^{th} iteration. For each key candidate from a key space $\mathcal{K}$[1] the adversary expresses the selection bit for all inputs:

$$v_1 = \phi(f(p_1, \hat{k})), \ldots, v_\ell = \phi(f(p_\ell, \hat{k})).$$

The traces are then divided into two groups ($\mathcal{T}_0$ and $\mathcal{T}_1$ resp.) with respect to the value v_i and each group is averaged:

$$\mathcal{T}_i = \{T_j \mid v_j = i\} \text{ and } \overline{\mathcal{T}_i} = \frac{\sum_{T_j \in \mathcal{T}_i} T_j}{|\mathcal{T}_i|}$$

Finally, the DPA trace is calculated as the difference of the two averaged trace $|\overline{\mathcal{T}_0} - \overline{\mathcal{T}_1}|$. The average traces is cancel each other out with a *random* set of inputs. However, when the predicted value v is correct, sample point that corresponds to the calculation of the value v reflect the power consumption correctly. Thus the average trace at this sample point is no more *random* set of inputs, but corresponds to specific values. In this instance, the average trace shows a clear difference as shown in Fig. 2.1, Step 5. This means that v classified into the sets correctly and the key guess $\hat{k}$ is indeed the *correct* key.

As an alternative, the adversary can employ a statistical tool like Pearson's correlation coefficient [CKN01] to deduce the best key candidate. In this version of the attack called as Correlation-based DPA (CPA), adversary calculates the correlation between $v = (v_1, \ldots, v_\ell)$ and $t_i = (t_i^1, \ldots, t_i^\ell)$ for each sample points $1 \leq i \leq N$ as in Eq. (2.1),

[1] $|\mathcal{K}|$ is small enough to process the analysis and generally it ranges from 2^1 to 2^8.

$$\rho_{v,t_i} = \frac{\sum_{j\in[1,\ell]}(v_j - \bar{v})(t_i^j - \bar{t_i})}{\sqrt{\sum_{j\in[1,\ell]}(v_j - \bar{v})^2 \sum_{j\in[1,\ell]}(t_i^j - \bar{t_i})^2}}. \tag{2.1}$$

Remark that the correlation measures the linear relation between two sets of data. Similar to DPA, the correlation between two *random* sets stays in a threshold (such as [-0.1,0.1]) as there is no linear relation at all. However for the correct key guess $\hat{k}$ the guessed values $v = (v_1, \ldots, v_\ell)$ correctly models the power consumption of the device. Therefore the correlation value i.e. Eq. (2.1) detect the relation between power consumption at the specific instance with v. Thus, the analysis results in an observable peak as shown in Fig. 2.1, Step 4.

2.1.1 Countermeasures

There have been a number of countermeasures in the literature in order to counteract SCA. Although they can be classified into two groups, hiding and masking, the main idea stays the same: make the side-channel information independent of the intermediate process of the device. The goal can be achieved by either making the side-channel information as stable as possible such that every operation consumes the roughly the same level of power or randomized it completely.
The hiding schemes are implemented to make the side-channel information randomized by executing the algorithm in a randomly-reorganized manner i.e. with shuffling or adding dummy operations randomly. Therefore the traces become time-domain desynchronized which makes the attack as described in Section 2.1 harder.
The masking is on the other hand is one of the most popular countermeasures against side-channel analysis. The method was proposed by Chari et al. [CJRR99b]. The main idea is to randomly split a sensitive variable x into $n+1$ shares, such that x can be recovered from $d+1$ ($n \geq d+1$) shares, while no information can be recovered from fewer than $d+1$ shares. The splitting is done in such a way that the following equation holds for a group operation $\perp$;

$$x_0 \perp x_1 \perp \cdots \perp x_n = x \tag{2.2}$$

In the literature $\perp$ is defined as addition $+$, xoration $\oplus$, etc. Most important examples are *Boolean masking* introduced by Ishai et al. [ISW03] which has been generalized by Rivain and Prouff [RP10], *Threshold Implementations* (TI) defined by Nikova et al. [NRS09], and *polynomial masking* as defined in [RP12] based on Shamir's secret sharing [Sha79].

Earlier masking schemes were considered secure under specific security models such as [OMPR05, CB08], however, they would still leak detectable information under the presence of glitches in the hardware [MME10, MPO05]. Due to these facts, glitch resistance masking schemes were introduced. TI were defined by Nikova et al. [NRS09] and then generalized by [BGN+14, RBN+15]. Gross et al. [GMK16] introduced *domain-oriented masking*. While the scheme had the same level of security as TI, it has lower implementation cost, since it has lower randomness cost.

Boolean Masking. The seminal work by Ishai, Sahai, and Wagner in 2003 [ISW03] introduced the concept of the private circuits and the idea of Boolean masking. The main idea of the scheme is to develop a theoretical foundation against side-channel attacks and private circuits protect the circuits if an adversary is able to access a bounded number of wires.
The Boolean masking is carried out in two steps. First, the data transformation is processed by representing each input x by $n+1$ values as described before. In the scheme the field operation defined as xoration $\oplus$ thus in Eq. (2.2) is realized as:

$$x_0 \oplus x_1 \oplus \cdots \oplus x_n = x, \tag{2.3}$$

where each x_i is an element from a field $\mathbb{K}$ and each set of n shares are distributed uniformly and independently. This achieved by randomly selecting the first n share, $x_0, , \ldots, x_{n-1}$ and calculating the last share as $x_n = x \oplus x_0 \oplus \cdots \oplus x_{n-1}$. Secondly, each operation is transformed to masked operations which we called *gadgets* and we use `typewriter` letters to distinguish the masked operations from regular (unmasked) operations. For example, XOR represents a field operation while `XOR` represents a secure-masked xor operation.
In the following we describe the basic gadgets. Let x and y be two elements from a field and consider an n^{th}-order masking scheme, i.e. x and y have been split into $(n+1)$ shares such that $\bigoplus_{i=0}^{n} x_i = x$ and $\bigoplus_{i=0}^{n} y_i = y$.

- `XOR` gadgets that calculates a masked representation of $z = x \oplus y$ such that $\bigoplus_{i=0}^{n} z_i = z$, can be summarized as follows:

$$z_i = x_i \oplus y_i \text{ for } 0 \leq i \leq n \text{ where } z = x \oplus y.$$

- `Affine` gadgets that calculates an affine function A of x in a secure manner such that of $z = ax \oplus b = A(x)$ where $a, b \in \mathbb{K}$ and $\bigoplus_{i=0}^{n} z_i = z$, can be summarized as

Algorithm 1 n^{th}-order secure multiplication over $\mathbb{F}_2$.

Input: Shares of x as $(x_i)_{0\leq i\leq n}$ and shares of y as $(y_i)_{0\leq i\leq n}$.
Output: Shares of $x \cdot y$ as $(z_i)_{0\leq i\leq n}$.

1: **for** $i = 0$ **to** n
2: **for** $j = i + 1$ **to** n
3: $r_{i,j} \leftarrow rand(0, 1)$ // Random sampling
4: $r_{j,i} \leftarrow (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$
5: **for** $i = 0$ **to** n
6: $z_i \leftarrow x_i y_i$
7: **for** $j = 0$ **to** n, $j \neq i$
8: $z_i \leftarrow z_i \oplus r_{i,j}$
9: **return** $(z_i)_{0\leq i\leq n}$

follows:

$$z_i = ax_i \oplus \delta_{0,i} b \text{ for } 0 \leq i \leq n \text{ where } z = ax \oplus b \text{ and } \delta_{0,i} = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases}$$

- `NOT` gadgets can simply be represented as follows;

$$\texttt{NOT}(x_0, \ldots, x_n) = (NOT(x_0), x_1, \ldots, x_n).$$

- `AND` gadgets suggested by Ishai et al. [ISW03] can be divided into three steps as below and a summary can be found in Algorithm 1. It require $(n+1)^2$ AND gates and $2n(n+1)$ XOR gates with $n(n+1)/2$ random bits. The gadget
  - For all $0 \leq i < j \leq n$ sample a random bit $r_{i,j}$,
  - For all $0 \leq i < j \leq n$ calculate $r_{j,i} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$,
  - For all $0 \leq i \leq n$ calculate $z_i = x_i y_i \oplus \bigoplus_{j\neq i} r_{i,j}$.

2.1.1.1 Secure Multiparty Computation:

Secure Multi-party Computation (SMC) with secret sharing schemes (denoted by (n, d)-SMC) allow us to split a sensitive variable into shares in such a way that neither the shares nor the computations on them reveal any critical information. Relying on this fact, Roche and Prouff proposed SMC as *multiparty circuit* (MPC) to counteract higher-order side-channel analyses, even in the presence of *glitches*, and showed how to apply it

to AES [RP12]. Moradi et al. [MM13] provided a first implementation of this scheme in hardware, as well as a practical side-channel analysis of their implementation. Similarly, Grosso et al. [GSF14] examined the performance of existing masking schemes in software for low-power microcontrollers. Both works concluded that the scheme comes with a significant overhead, even when compared to other side-channel protection schemes. The latter work proposed the usage of packed secret sharing to make the scheme more efficient for higher protection orders. They expanded SMC into a (n, d)-multiparty circuit using a sequence of sub-circuits.

Shamir's Secret Sharing. The second masking scheme that we would like to focus is polynomial masking which is based on based on Shamir's secret sharing [Sha79]. This novel idea is employed by Roche and Prouff [RP11, RP12] and Goubin and Martinelli [GM11] (which is shown to be flawed in [CPR13]) using secure multi-party computations (SMC) defined by Ben-Or et al. [Ben88] and Gennaro et al. [GRR98]. The SMC proposed by Ben-Or et al. [Ben88] is both t-private and t-resilient, i.e. it guarantees that some subset of t parties can neither learn nor modify results. Roche and Prouff adopted the t-private property to achieve side-channel protection through glitch-free SMC [RP11, RP12]. It is known that the t-resilient property can be used to thwart fault attacks [RP12, GSF14].

The scheme was introduced by Shamir in 1979 [Sha79] and it allows players to split a secret using a polynomial. In the protocol, the trusted dealer generates a random polynomial, $F(\mathsf{x}) = x + f_1\mathsf{x} + \ldots + f_d\mathsf{x}^d$ to share a secret $x \in \mathcal{K}$ with coefficients $f_i \in \mathcal{K}$ for all $1 \leq i \leq d$. Here, we use x as the secret and x as the variable of the function F. The secret value is shared by evaluating $F(\mathsf{x})$ for $n+1$ distinct and nonzero public points $(\alpha_0, \ldots, \alpha_n)$. The shared representation of x is shown as $\mathcal{F}_x = \{F(\alpha_0), \ldots, F(\alpha_n)\} = \{F_0, \ldots, F_n\}$. A visual representation of the scheme can be found in Fig. 2.2.

The secret reconstruction of polynomial masking requires polynomial interpolation and requires at the shares of least $d + 1$ players. Without loss of generality let us assume that the first $d' \geq d + 1$ shares are combined to reconstruct the polynomial. Evaluating the formula for $\mathsf{x} = 0$ simply reveals the secret value x:

$$F(\mathsf{x}) = \sum_{j=1}^{d'} = P_j(\mathsf{x}), \text{ where } P_j(\mathsf{x}) = F_j \prod_{\substack{k=1 \\ k \neq j}}^{d'} \frac{\mathsf{x} - \alpha_k}{\alpha_j - \alpha_k}. \tag{2.4}$$

Remark that, reconstruction with at least $d + 1$ players generates a unique polynomial.

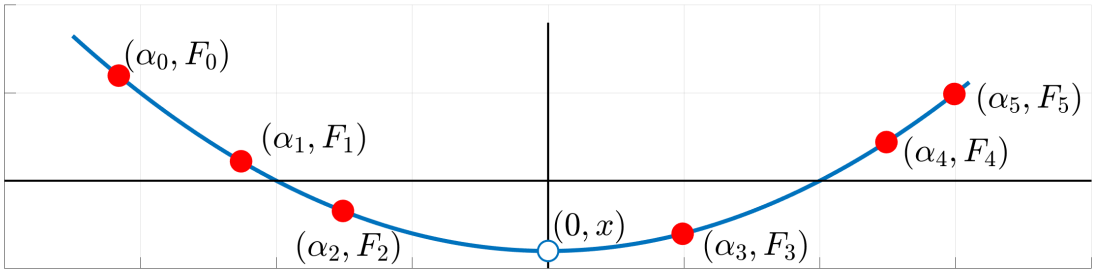


Figure 2.2: A visual representation of polynomial masking where $n = 5$ and $d = 2$. Observe that any set of shares with ≤ 2 elements is distributed uniformly and independently and in order to reconstruct the value x at least three shares need to be combined.

Therefore, one can define the connection between the coefficients $(C = (f_1, \ldots, f_n))$[2] and the shares $(S = (F_0, \ldots, F_n))$ using the Vandermonde matrix (V):

$$\underbrace{\begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^n \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^n \end{pmatrix}}_{V} \cdot \underbrace{\begin{pmatrix} x \\ f_1 \\ \vdots \\ f_n \end{pmatrix}}_{C} = \underbrace{\begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_n \end{pmatrix}}_{S} \tag{2.5}$$

Using Eq. (2.5) we can formulate the connection as $C = V^{-1}S$ and can reconstruct all coefficients of $F(\mathsf{x})$. Observe that, V is invertible for $\alpha_i \neq \alpha_j$.
As in Boolean masking, the operations are needed to be adjusted to masked values. Let x and y be two values and consider an $(n, d)^{th}$-order masking scheme, i.e. x and y have been split into $(n + 1)$ shares $\{F_0, \ldots, F_n\}$ and $\{G_0, \ldots, G_n\}$ such that $F(0) = x$ and $G(0) = y$.

- **Affine transformation of a secret (`Affine`):** An affine transformation $A(x) = a\mathsf{x} + b$ with $a \neq 0$ can be computed on the secret value by applying $\mathtt{A}$ to secret shares locally:

$$\mathcal{F}_{\mathtt{A}(x)} = \{A(F_0), \ldots, A(F_n)\}$$

- **Addition of two secrets (`Add`):** Two secret values $x + y$ can be added by pairwisely adding shares.

[2] Since F is a degree d polynomial $f_{d+1} = f_{d+2} = \cdots = f_n = 0$

$$\mathcal{F}_{x+y} = \{(F_0 + G_0), \ldots, (F_n + G_n)\}$$

- **Multiplication of two secrets (`Mult`):** The multiplication of two secret values xy requires communication between players. An efficient algorithm was proposed by Gennaro et al. [GRR98], simplifying the original SMC multiplication proposed by Ben-Or et al. [Ben88]:

  1. Each player computes $H_i = F_i \cdot G_i$.
  2. Each player generates a degree d polynomial $\mathcal{Q}_i(x)$ such that $\mathcal{Q}_i(0) = H(\alpha_i)$, and sends the value $\mathcal{Q}_i(\alpha_j)$ to j^{th} player.
  3. Each player computes its secret share by $\mathbf{Q}_i = \sum_{j=0}^{n-1} \lambda_j \mathcal{Q}_{j,i}$ where $(\lambda_0, \lambda_1, \ldots, \lambda_n)$ corresponds to the first row of the inverse Vandermonde matrix.

  The shares calculated in step 1 cannot be used as a valid secret sharing because of two main problems: (1) the polynomial is a degree $2d$ polynomial. (2) It is not a random polynomial [Ben88]. In step 2 and 3, degree reduction and randomization are done in order to generate a *proper* (n, d)-secret sharing of xy.

- **Efficient squaring operation (`Sqr`$_k$):** Efficient squaring operations can be used to eliminate costly multiplications in GF(2^m) [RP12]. Squaring can be defined as $\eta_k(y) = y^{2^k}$ and requires two conditions on public points α_i :

  1. $\alpha_i \neq 0$ for $i = 0, \ldots, n$.
  2. For every α_i there exists α_j such that $\alpha_i^2 = \alpha_j$.

  Each player calculates the operation on its share locally by $\eta_k(F_i)$. The family of shares $(\eta_k(F_i))_{i=0,\ldots n}$ is a valid secret share of x^{2^k}. However, communication between players is needed to do the reordering of the secret shares.

2.1.2 Probing attacks and Its Connection to Real-world Adversaries

Until now what we have assumed is an adversary observes a physical leakage defined as noisy leakage model [CJRR99b]. This model assumes an intermediate variable x leaks information as $x+\chi$ where χ sampled from a Gaussian distribution and adversary obtains $x + \chi$. In this section we introduce probing model by Ishai et al. [ISW03] a theoretical model of probing adversaries i.e. a model where an adversary that can access a bounded number of intermediate variables. We would like to give a better understanding of probing attacks due to its importance in the analysis of masked implementations.

Briefly, a probing adversary may invoke a cryptographic implementation multiple times with chosen inputs. Before each call, the adversary can choose a set of up to t wires of the circuit and observe the values on these wires during the invocation. After c calls, an attacker can then combine the $c \times t$ observations in arbitrary ways to extract information about sensitive variables. This model is closely related to concrete physical attacks: For example, in [KGM+20], eight simultaneous probes are given as an upper limit achievable by modern commercially-available probe stations, and we therefore assume that t is at most sixteen. While the t-probing model is a clean theoretical model, Duc et al. [DDF19] showed that security in this model implies security in the more practical SCA-inspired noisy leakage model [PR13]. To protect against multi-probe attacks, generally higher-order masking is applied, where the number of independent shares is increased. Higher-order SCA is expensive, as the number of measurements needed grows exponentially with the masking order, effectively limiting the attack order or the number of *simultaneous* probes an adversary may use (which we assume is below sixteen). Kranchenfels et al. [KGM+20] describe a new attack technique, laser logic state imaging, that can potentially have an unlimited number of probes, however this attack is quite new and may not be widely applicable, but in any case must be mitigated with countermeasures below the software level (at the package, device or circuit level).

In this thesis, we use the noisy leakage model introduced by Chari et al. [CJRR99b] and extended by Prouff and Rivain [PR13]. The model enables an adversary to obtain each intermediate value perturbed with a noisy leakage function. Furthermore, as stated above we use the connection between probing and the noisy leakage model given by Duc et al. [DDF19]. Therefore a probing adversary reflects the capabilities of a real world adversary such as DPA and security against probing adversaries as defined in [ISW03] implies security in the noisy leakage model.

Note that a single probe will only provide a few key bits, but the attack scales linearly in the key size. Side-channel countermeasures depend on the quality of the side-channel, its signal-to-noise ratio. If the implementation is noisy or has weak side-channels, security against $t = 2$ can be sufficient. Currently, protection against $t = 4$ probes is often an upper bound on security in practical systems [Nik20].

2.1.3 Security Notions

In the previous section we focused on some secure solutions in the literature. One of the main challenges of proposing such a construction is using the proper security notion. In this section we focus on the formal security notions used in the literature.

The ISW Probing Model. Ishai et al. [ISW03] introduced how to build secure circuits against an adversary that can probe a portion of intermediate variables of the circuit.

The most basic security notion for a masking countermeasure is the *t-privacy* of a gadget G [ISW03]. Remark that, in the setting of the probing model an adversary may invoke the (randomized) construction multiple times and adaptively choose the inputs. Prior to each invocation, the adversary may fix an arbitrary set of t wires of the circuit values which can be observed during that invocation.

The notion requires the existence of a *simulator* [ISW03]. The simulator attempts to simulate the view of the adversary using only black-box access (i.e., without having access to any internal wires) to the masked. The presence of a simulator (or simulatability) implies that t probes are indeed independent of the processed data and therefore t probes will not provide any information to the adversary.

Informally, this means that the information provided by t probes of outputs or intermediate variables can also be obtained by probing t input variables, as long as the inputs are an encoding of x.

While the idea behind the notion is relatively simple, it is unfortunately not composable as the output of a t-private gadget is not necessarily a truly uniform encoding. The composition of two t-private gadgets is thus not necessarily t-private [BBD+15a].

Example 2.1. *Assume that we have two secret (x, y) shares as (x_0, x_1) and (y_0, y_1) such that $x = x_0 \oplus x_1$ and $y = y_0 \oplus y_1$. Let's consider the following gadget;*

$$G((x_0, x_1), (y_0, y_1)) = (x_0 \oplus y_1, x_1 \oplus y_0) = (z_0, z_1).$$

Clearly a single probe (a single variable) can be simulatable by at most one share of each input sets e.g. z_i can be simulatable by x_i and y_{i+1}. Now consider the following copy gadget an a secret a shared as (a_0, a_1) such that $a = a_0 \oplus a_1$;

$$C(a_0, a_1) = ((a_0, a_1)(a_0, a_1)).$$

Again, a single variable can be simulatable by a share of input set. Now lets consider the composition $G \circ C$:

$$G[C(a_0, a_1)] = G[(a_0, a_1), (a_0, a_1)] = (a_0 \oplus a_1, a_1 \oplus a_0) = (z_0, z_1).$$

Clearly this composition is not secure anymore, since e.g. z_0 can only be simulatable using both shares a_0 and a_1.

Non-interference. In order to remove the requirement that the inputs have a certain distribution, the notion of *non-interference* was introduced [BBD+16]. Non-interference gets rid of the dependency of the uniformly encoded inputs, but a more subtle issue still prevents a composability result. To give an intuitive overview of this problem, consider a gadget G with two shared-sensitive inputs $(x_0, x_1, \ldots, x_n)$ and $(y_0, y_1, \ldots, y_n)$ and output a shared-sensitive output $(z_0, z_1, \ldots, z_n)$. Non-interference now implies that for any $I_y \subsetneq [n]$, the values $\{z_i\}_{i \in I_z}$ can be simulated from $\{x_i\}_{i \in I_x}$ and from $\{y_i\}_{i \in I_y}$ for two sets I_x, I_y of cardinality at most $|I_z|$. Now, if G is used in another circuit, it might be the case that x and y are correlated (or even identical). Then $\{x_i\}_{i \in I_x} \cup \{y_i\}_{i \in I_y}$ might reveal information about x. See e.g., [BBD+16] for a more detailed explanation. Hence, an even stronger notion, called *strong non-interference* was introduced [BBD+16], that guarantees a clear separation between input variables and output variables.
In the following, *perfectly simulatable* means that there is a probabilistic algorithm that on input $x_{|I}$ generates t intermediate variables and $|\mathcal{O}|$ output variables with the same probability distribution as the gadget.

Definition 2.1 (t-NI, t-SNI). Let G be a gadget with inputs in $\mathbb{F}^{n+1}$ and $t \leq n$. Suppose that for any set of t_1 intermediate variables and any subset of O of output indices with $t_1 + |O| \leq t$, there exists a subset of indices I such that the output distribution of the t_1 intermediate variables and the output variables $z_{|O}$ is perfectly simulatable from I. Then

(i) if $|I| \leq t_1 + |O|$ we say G is *t-non-interfering (t-NI)*, and

(ii) if $|I| \leq t_1$ we say G is *t-strong-non-interfering (t-SNI).*

Note that *linear* operations which can be performed share-wise (such as addition or multiplication by a constant) are trivially t-NI, as each computation on share i can be simulated from the input share x_i.
Note that in SNI the size of the input set I depends only on the intermediate variables. For a given set Int of t_1 intermediate variables and a subset O of output indices with $t_1 + |O| \leq t$, we say that the output variables $O' \subseteq O$ that are simulatable without any knowledge about I are (Int, O, t)*-input-ignorant.* Hence, if no intermediate variable was probed, the output of the circuit is independent of its input (from observing at at most t positions) and thus all subsets of at most t outputs are input-ignorant. We will occasionally talk about the concrete distribution of these input-ignorant variables. If all (Int, O, t)-input-ignorant output variables of a t-SNI gadget G are distributed according to a distribution D, we say that G is *t-strong-non-interfering (t-SNI) with output-distribution D.*

Finally, the SNI notion guarantees that the composition of two t-SNI gadgets is t-SNI again. For the sake of completeness, we repeat the corresponding proposition from [BBD+16].

Lemma 2.1 (Proposition 4 [BBD+16]). *Let C be a circuit built from gadgets $G_1, \ldots, G_r$ such that all G_i are t-NI, all encodings are used at most once as input of a gadget call other than* RefreshM*. Then C is t-NI. Moreover, C is t-SNI if it is t-NI and all encodings corresponding to the outputs of C are refreshed through* RefreshM *before output.*

The commonly used term *probing-security* can either mean privacy [BBC+19] or non-interference [CGPZ16]. Classically, the non-interference notions only deal with gadgets where all of the inputs and outputs are sensitive. To also handle public, non-sensitive values, the notion of *NI with public output* (t-NIo) was proposed in [BBE+18]. As mentioned in [BBE+18, Lemma 1], if a gadget G is t-NI secure it is also t-NIo secure for any public outputs.
The notion of t-SNI security is known to provide scalable protection against a broad class of side-channel attacks up to t-th order under few additional assumptions [DDF14], where the needed number of observations grows exponentially in t. It has been widely adopted e.g. for automation of applying and checking side-channel resistance in hardware and software designs [BBC+19] and can be viewed as a reliable and fairly efficient method to achieve a desired degree of side-channel resistance. Similarly, SNI can also be used to ensure and verify protection of hardware circuits, where special care needs to be taken to account for *glitches*, either by extending the model or by carefully placing registers to interrupt unintended asynchronous propagation of signals [FGP+18].
Masking schemes are commonly used to protect against SCA, such as the ones achieving SNI security, are also MPC protocols. In the case of masking, the parties are just different parts of the same circuit (one might call it *MPC-in-the-circuit*), forcing an attacker to consider more than t parts of the circuit in parallel to infer about secret intermediate values.

Formal Verification. The verification of such masked implementations is a manual and error-prone task. The tool assisted formal verification has been adopted by the academia and industry. The most famous tool is `maskVerif` developed by Barthe et al. [BBC+19]. The tool is language-based and provides an automatic and formal security verification of higher-order masking implementations based on the various notions such as NI, SNI and probing. NI and SNI notions. Briefly speaking, it checks every possible

attack combination within the implementation and either provides a security proof for the specified order or gives a list of potential attack targets in the implementation. The bottleneck of the tool is the order of the masking and the complexity of the implementation. It has been used in the literature to provide assurance of masked implementations [BBD+15b, GSDM+19, SEL21].
Another work by Bloem et al. [BGI+18] provide a formal verification using the circuit's netlist. This approach results in the direct localization of the vulnerabilities and provides concrete security assessment. More recent works such as SILVER [KSM20] uses symbolic and exhaustive analysis of statistical independence of joint distributions between every set of probes and every subset of input shares. This tool is used to analyze the tools that generated masked implementations [KMMS21].

2.1.4 Leakage Assessment

Main aim of the assessment is to provide a quantitative and statistically sound answer to the question if a cryptographic implementation is vulnerable to physical attacks or not. Remark that it is infeasible to test and verify security against all possible attack vectors. Therefore leakage assessment focuses on a specific question; are side-channel information corresponding to a cryptographic device data dependent or not. If the device fails the test, i.e if the device shows data dependent behavior, we cannot conclude the actual vulnerability with respect to key recovery attacks. But this result proves the existence of such a vulnerability. On the other hand if the device pass the test, the assessment experimentally provides a desired security level with a confidence level. These analysis methods are extensively used in academia and industry. One example is test vector leakage assessment (TVLA) by Goodwill et al. [GGJR+11], which is based on Welch's t-test. Another version employs χ^2-test as the statistical tool [MRSS18]. More recent works shows that deep-learning techniques can improve the efficiency of the analysis [MWM21].
In this thesis we employ TVLA by Goodwill et al. [GGJR+11]. The analysis detects leakage at a given order and has two different versions: the non-specific and the specific method. The first version is defined as Fixed-vs-Random (FvR) and it aims to detect all possible sources of first-order leakage. During the trace collection phase, a set of side-channel traces is collected by processing either a fixed input or a random input under the same conditions. An example image of the collection phase can be found in Table 2.1.
If the t-test only gives a very small value, this indicates that the run of the algorithm on a fixed input is indistinguishable from a run of the algorithm on a random input. Hence,

	plaintext		ciphertext
000	00000000000000000000000000000000	→	7DF76B0C1AB899B33E42F047B91B546F
001	99DB065D7FE92B34FE7FB00A799F370C	→	3FB07D4DABF39BD297CFFE0977B0F17C
002	28BE3BB98D2D8807C6421B05B04F7F2C	→	A2109EFD661C7084960925A6865CFBF7
003	00000000000000000000000000000000	→	7DF76B0C1AB899B33E42F047B91B546F
004	00000000000000000000000000000000	→	7DF76B0C1AB899B33E42F047B91B546F
005	0AF4EE9B536C77DFFD435C355F8F51A0	→	FF22FB6E4FC14452CF0B094E7575D4F3
006	00000000000000000000000000000000	→	7DF76B0C1AB899B33E42F047B91B546F
007	11F416B019A6767F32106CE446F6937A	→	C3F574B6BA665E42A04EA505C8F71458
008	00000000000000000000000000000000	→	7DF76B0C1AB899B33E42F047B91B546F
009	961E6187F93AEAC55660C99C0E5F998E	→	8B9A136A74D167760F7CD93076129117
010	DF1CE29BE0FF201D727613494026A3E6	→	25D77EC44F76B39560CA12D634CEC522
⋮	⋮	⋮	⋮

Table 2.1: A visual representation of the collection phase. The device is set to process fixed input or a random input in a random patter.

a small value in the fixed-vs-random scenario implies the *absence* of sensitive leakage. The test assumes no other information and it can be used to detect leakage through the entire algorithm.

The second test is defined as Random-vs-Random (RvR), and employs only traces with a random input, and uses a function of inputs to sort the traces. The main advantage of the RvR method is that it can identify specific sources of exploitable leakage and thus shows the feasibility of an actual attack.

After collecting and sorting the traces, the means (μ_0, μ_1) and standard deviations (σ_0, σ_1) for the two sets are calculated. Welch's t-test is computed as;

$$t = \frac{\mu_0 - \mu_1}{\sqrt{(\sigma_0^2/n_0) + (\sigma_1^2/n_1)}}, \tag{2.6}$$

where n_0 and n_1 denote the number of traces for the sets, respectively. Remark that the value t is defined as the t-value in the statistics and different from t-probes. The t-test indicates whether the two distributions are *indistinguishable* for a first-order side-channel analysis.

The threshold value for t value is determined as for general purposes 4.5 [SM15] in the literature. For more specific targets (depending on the length of the traces) different threshold values are suggested by [DZD+18]. For specific targets we use the value 5.7 for traces of length more than 10^4, and the value 6.1 for traces of length more than 10^6. This threshold rejects the null-hypothesis of non-leakage with $> 99.99\%$ probability.

Observe that Eq. (2.6) analyze the indistinguishably of the means of traces corresponding

the first-order side-channel attacks. Using the similar idea one can define higher-order t-test can be defined analyzing the other moments of the distribution of the sample points corresponding higher-order t-tests such as variance and skewness [SM15].

Moreover, one can employ multivariate t-test to analyze the masked implementation. A multivariate t-test combines multiple points in the trace thus catches the shares processed at the different instances. De Cnudde et al. [CBR+15] uses this method to analyse the higher-order masked AES S-box implementation and show that even any combination of two sample points leaks no information.

2.2 Fault Attacks

Next, we focus on active attacks or Fault Attacks (FA). Unlike the side-channel attacks where the adversary can only observe the side-channels, this type of attack is implemented by an adversary who can interact and/or manipulate the implementation and can observe unexpected behaviors such as faulty outputs, abort signal etc. [BDL97]. The faults can be injected e.g. by overheating, under/over-powering or optical beams. These unexpected behaviors cause a faulty computation which results in information related with the secret key. Prominent examples are on RSA [BDL97] and AES [Muk09] where single faulty ciphertext to successfully recover the secret key.

Before delving into some fault attack examples, we introduce the different fault models in the literature. The models range from bit level faults to data path faults or control flow faults. Briefly, bit level faults can be summarized as *reset* (change a wire value to zero), *set* (change a wire value to one) or *toggle* (flipping the wire value) attacks [IPSW06]. More general faults, such as data path targeted faults can be classified according to the distribution of the injected faults. An attacker can sample the injected faults from a uniform distribution or from a biased distribution, where one set of faults is significantly more probable then the set or remaining faults [SMG16]. Moreover, an extreme case of this idea is used by Reparaz et al. [RMB+17] and it gives the attacker full control of the faults and is known as known-value faults. Depending on the security models, the distribution of the faults is combined with the number of injections or number of wires to inject the fault. For example, known-value faults can be used with limited number of wires to inject the faults or a uniform fault model can be used to injects faults to all wires [RMB+17]. A fault model is an important part of the attack description and it summarizes what does it to mean by *manipulating* the implementation. It is a mathematical description of the characteristic of the injected fault which includes the mathematical representation, duration etc. In the following, we list the most common

aspects of a fault model as introduced in [KSV13].

- **Number of faulted bits** is the first characteristics we need to define the fault model. This can be single bit faults, word-size faults and variable size faults.

- **Fault Type** defines the mathematical effect of the fault on the implementation. Stuck-at-zero/one (fixing value of a bit to zero or one resp.) and bit flip (toggling the value of the bit) are three most used bit level faults. For the larger size faults in addition to stuck-at and flip faults, random faults (adding a random value with a given distribution) can also be defined.

- **Fault Duration** impacts the lifespan of a fault and there exists three categories: transient, permanent and destructive. While the transient faults last only a limited time and after a certain duration the correct value is presented again. The permanent faults last until the value is rewritten. Finally, the destructive faults lasts indefinitely by damaging the physical layer of an implementation. Stuck-at faults can be listed as an example for this type of fault attacks.

- The last item is the **Controllability over the Fault Timing**. These can be divided into two sub-classes as spatial and temporal fault control. This defines the adversaries's control over the timing and the place over the fault point.

The most remarkable fault attacks in the literature are Differential Fault Analysis (DFA) [BS97], Safe-Error Attack (SEA) [YJ00], Fault Sensitivity Analysis (FSA) [LSG+10], and Statistical Ineffective Fault Attack (SIFA) [DEK+18].

Differential Fault Analysis. During DFA an adversary injects a fault which changes a variable inside the algorithm $\mathbf{E}$, ideally close to the end of the encryption and collects the correct and faulty ciphertext corresponding to the same plaintext i.e. pairs (p, c) and (p, c'):

$$\mathbf{E}_k(p) = c \text{ and } \overset{\lightning}{\mathbf{E}}_k(p) = c'.$$

Using this pair and a key guess, an adversary can compute the partial decryption until the point where the fault is injected and check if the effect of the fault injection is correct or not.

Safe-Error Attack. This fault attack uses the fact that a fault has no effect on the output if the key bit has a specific value. Unlike DFA, this attack only concerns if the non-faulted output is equal to the faulted output. If an adversary inject a stuck-at-zero fault to a key bit and if there is no difference in the behavior of the device e.g. the output is correct then the value of the targeted key bit is revealed. Therefore, this attack can eliminate countermeasures that ensure the correctness of the output. Using this idea the adversary can proceed with the same methodology.

Fault Sensitivity Analysis. As in DFA this attack requires an adversary to collect faulty and non-faulty ciphertext pairs, however the value of the faulted ciphertext is not important. The core idea is that by increasing the intensity of the fault (e.g by disturbs the power supply or manipulating clock) one can find the distinguishable characteristic, e.g. the first appearance of a faulty output.

2.2.1 Countermeasures

A common technique to prevent fault induction is error detection through adding redundancy. Often, error detection requires the duplication of computation in space or time [BECN+06], especially for symmetric ciphers, to achieve the desired error detection ratio. For asymmetric ciphers, lower overheads are often possible [Sha99,Gir06]. Another technique proposed by Genkin et al. [GIP+14] uses algebraic manipulation detection (AMD) codes to protect sensitive variables. The idea is to compute a proof that the output is correct. However, this approach requires generating a MAC tag for each gate. Other branches of fault prevention are *infective* countermeasures, which aim at randomizing the secret state if an error occurs. Lomné et al. [LRT12] introduced an infective countermeasure using multiplicative random masking. Gierlichs et al. [GST12] presented the idea of dummy rounds. However, these countermeasures were broken by Bastellini et al. [BG13]. The second scheme was improved by Tupsamudre et al. [TBM14]. The updated scheme has been analyzed in [BG16], showing that getting the countermeasure right and efficient is difficult. The countermeasures are classified as follows:

- **Redundant Operations:** this is the simplest and naive way of providing a countermeasure. The redundancy can be supplied to the implementation in two ways: spatial and temporal. In the first one the circuit is repeated using the same circuit (redundancy in time) or using the same circuit in parallel (redundancy in space). As a result, if the implementation is repeated k times, a total of $k-1$

faults can be detected just by comparing the outputs [KW02]. On the other, hand in the temporal redundancy the implementation is protected by error correction and/or detection codes on intermediate values [KKT04].

- **Infective Countermeasures:** As described in the previous sections, the essence of the fault attacks relies on interpretation of the faulty outputs. The infective countermeasure aim at this connection and randomize the connection between faults and corresponding outputs. When a fault is injected, the implementation does not try to detect but propagate the faults and randomize the output [LRT12].
- **Sensors and Shields:** The main idea of this type of countermeasure is to reduce the efficiency of the fault attacks or preventing the attacks completely [SA02]. In case of a detection an abort signal is created to prevent the output to be produced.

2.2.2 Security Notions

As given in Section 2.1.3, the security notions for side-channel analysis is a well-defined and already explored area. Recent years also gave us the information-theoretic foundations against fault attacks. The notions are inspired from non-interference and defined as k-Non-Accumulative (k-NA) and k-Strong Non-Accumulative k-SNA.

Definition 2.2 (k-NA, k-SNA [DN20]). Let G be a gadget and suppose there exists k_1 errors on each input and k_2 errors on the intermediate variables, with $k_1 + k_2 \leq k$.

(i) if the gadget either aborts or gives an output with at most $k_1 + k_2$ errors, we say G is *t-non-accumulative (k-NA)*, and

(ii) if the gadget either aborts or gives an output with at most k_2 errors, we say G is *t-strong non-accumulative (k-SNA).*

The main idea of this notion accumulation is to show that an injected fault affects only one of the output share of the gadget. Therefore it is ensured that faults does not spread and accumulate on a limited number of shares which allows error detection mechanisms to to detect faults. The number of faults k is defined as the maximum number of circuit wires that can be faulted and still return the correct output. However the composability needs to be considered in this case. An adversary can always inject faults on output, therefore the notion cannot guarantee that all the outputs are correct. Therefore, the notion deals with faults on inputs shares or intermediate values such that the fault spread to other share. This case is dealt with the abort mechanisms within the gadget. Thus the notion requires the error detecting methods and provides a composability result as in NI and SNI.

Formal Verification. Despite the importance of the fault attacks, there was a gap between fault detection and formal verification. The tool by [AWMN20] provides the analysis tool of an implementation of fault-injection countermeasures at the gate level, called VerFI. The tool can be seen as a fault simulator and provide an vulnerability assessment for the design. The tool checks if a fault model is detectable or not with a given set of inputs (selected by the user). Therefore it does not provide a complete proof of security but an analysis of the fault detection capabilities of the implementation.
A recent work by Richter-Brockmann et al. [RSS+21] introduces the tool FIVER: a formal verification approach for fault attacks that provides a security proof. The case studies show that the tool provide an efficient way of security evaluation depending on the number of gates in the design.

2.3 Combined Attacks

As the attacks become more and more apparent, combined attacks becomes a real threat. They can be defined as a combination of certain side-channel model and a fault model independently. As shown in [AVFM07] a passive and active combined attacks (PACA) is a real threat and can be implemented efficiently.
To achieve both fault and side-channel resistance, two countermeasures can be combined. However, while the interactions between the countermeasures have not been studied in much depth, combining ad-hoc methods can have adverse effects [REB+08, LFZD14]. Furthermore, overheads are huge and become larger for combined methods. Nevertheless, resistance against both attacks is important, since attacks can be combined to have greater effects on partially protected implementations [RLK11, LRT12]. So far, the amount of research in this area is scarce. Ishai et al. extended private circuits [ISW03] by adding redundancy via encodings [IPSW06]. Therefore, they were able to generate a circuit which resists both SCA and tampering attacks. Depending on the fault model, they defined two different encodings. Therefore they managed to detect faults by means of invalid encodings. Schneider et al. [SMG16] proposed a combined side-channel and fault countermeasure using TI and error detecting codes [MS77]. While their proposal is fairly efficient, the fault coverage depends on the fault distribution. The scheme does not by itself ensure the randomness of the output. The SCA resistance of scheme relies on security of a TI. Another countermeasure was introduced by De Cnudde et al. [DCN16] which enhanced [IPSW06] with TI. The idea is the same which forwards the valid inputs unless a faulty encoding is detected. A recent countermeasure introduced by Reparaz et al. [RMB+17] builds on doing computations on shared values and MAC tags. The side-

channel resistance of the scheme relies on the secret sharing while fault resistance of the scheme relies on the MAC-tag shares. Using a similar idea De Meyer et al. [MAN+19] combined masking with information-theoretic MAC tags and infective computation.

2.3.1 Security Notions

As a natural approach the Section 2.1.3 and Section 2.2.2 can be combined to generate a security notion for combined security. To things are needed to be full filled in the combination case: the correctness of the output (concerning injected faults) and the privacy of the gadgets (the combination of injected faults and probed variables.)
The main idea is to seen faulted values as a probing tool. As given by [Cla07], fault attacks can be considered as a probing tool. Therefore the faulted values give an extra share to the simulator. Moreover the notion requires an abort signal to guarantee the correctness of the output. Remark that this signal should be simulated by the simulator. The following notion captures the security notion for d-probes and k-faults in such a way that d' probes and k' faults can be combined with $d' + k' \leq d$ and $k' \leq k$. Moreover it provides composability.

Definition 2.3 ((d, k)-NINA, (d, k)-SNINA [DN20]). Let G be a gadget and suppose there exists k_1 errors on each input, k_2 errors on the intermediate variables, with $k_1 + k_2 \leq k$ and there exists d_1 intermediate probes, d_2 output probes such that $d_1 + d_2 + k_1 + k_2 \leq d$,

(i) if the gadget either aborts or gives an output with at most $k_1 + k_2$ errors and probes can be simulated by an input set of shares I such that $|I| \leq d_1 + d_2 + k_1 + k_2$, we say G is *t-non-interfering non-accumulative (k-NINA)*,

(ii) if the gadget either aborts or gives an output with at most k_2 errors and and probes can be simulated by an input set of shares I such that $|I| \leq d_1 + k_1 + k_2$, we say G is *t-strong non-interfering non-accumulative (k-SNINA).*

As given above the faults and probes act as same. In the next notion the adversary does not learn anything by faulting the gadget i.e. the simulator does not get additional shares by injecting a fault.

Definition 2.4 ((d, k)-ININA, (d, k)-SININA [DN20]). Let G be a gadget and suppose there exists k_1 errors on each input and every set of k_2 errors on the intermediate variables, with $k_1 + k_2 \leq k$ and there exists d_1 intermediate probes and d_2 output probes such that $d_1 + d_2 \leq d$,

(i) if the gadget either aborts or gives an output with at most k_1+k_2 errors and probes can be simulated by an input set of shares I such that $|I| \leq d_1 + d_2 + k_1 + k_2$, we say G is *t-Independent non-interfering non-accumulative (k-ININA)*,

(ii) if the gadget either aborts or gives an output with at most k_2 errors and and probes can be simulated by an input set of shares I such that $|I| \leq d_1 + k_1 + k_2$, we say G is *t-strong independent non-interfering non-accumulative (k-SININA)*.

2.4 Physical Attacks on White-box Designs

Cryptanalysis of WBCs usually requires a time-consuming reverse engineering step to surpass included obfuscation layers [GPRW19]. To overcome this, Differential Computation Analysis (DCA) or in short *computational analysis* of white-box cryptosystems has been proposed by Bos et al. [BHMT16]. The attack is inspired by physical gray-box attacks, mainly *side-channel attacks* and perform a statistical analysis of observable intermediate states of a cryptographic implementation. Following this work, further generic computational analysis techniques have been proposed for white-box implementations, such as Zero Difference Enumeration [BBIJ17], Collision Attacks, and Mutual Information Analysis [RW19]. Alpirez-Bock et al. [BBMT18] analyzed the ineffectiveness of internal encodings and explained why DCA works so well in the white-box setting. Even fault attacks [BDL97, BECN+06] have been shown to be an effective method for state and key recovery attacks on white-box implementations [BHMT16, BBB+19]. Biryukov et al. [BU18] introduced two new types of fault attacks to reveal the structure of a white-box implementation, an important step of overcoming obfuscation in WBC.

Let's deep dive into DCA, as it is regarded as one of the most efficient attacks against white-box implementations, since it does not require full knowledge of the white-box design and thus avoids the time-consuming reverse engineering process [BHMT16]. It utilizes internal states of the software execution (such as memory accesses) to generate software traces. The first step of DCA consists of collecting software traces consisting of memory addresses, intermediate values, or values written/read by the implementation and this step corresponds to only distinction between DCA and DPA/CPA. The second step consists of a statistical analysis of the software traces collected during the first step i.e. difference of means or Pearson's correlation as outlined in Fig. 2.1

The second most important attacks on WBC is the Algebraic attack which has been introduced during the WhibOx contest of Ches 2017 [TWC17]. Although the majority of the implementations in the contest were broken in less than one day, even the strongest

design (by means of the surviving time: 28 days) was broken by algebraic analysis [BU18, GPRW19]. Algebraic attacks try to find a set of circuit nodes whose linear (or higher) order of combination equals to a predictable vector. Observe that if an implementation is protected by a classical masking scheme (Eq. (2.3)), there exists a set of circuit nodes (corresponding to the secret shares) such that a linear combination (i.e. a first-order combination) is always equal to a predictable secret value. This means that classical masking schemes are inherently vulnerable to first-order algebraic attacks *independently of the masking order* [BU18,GPRW19]. Like computational attacks, algebraic attacks do not require complex reverse engineering and are thus a generic threat that any white-box implementation needs to address.

2.4.1 Masking on White-box Designs and Its Challenges

In order to counteract DCA, the natural approach is to apply masking. The dedicated masked white-box implementations have been introduced in the literature. As an example for this methodology, consider the dedicated masked white-box implementation introduced in [LKK18] which was broken in [RW19]. Classical masking schemes are also not sufficient to procure a secure implementation as the strength of the masking is related to the noise within the implementation and a DCA adversary can produce noise free software traces [BRVW19].
Remark that the security of masking schemes against SCA require noisy observations [CJRR99a]. Higher order variants of DCA have been shown to be effective when applied to masked white-box implementations due to the adversary's ability to observe shares without noise [BRVW19]. To deal with this problem, artificial noise sources such as control flow obfuscation [BBIJ17], shuffling [BRVW19], and input and output encodings [BBMT18] have been analyzed in the literature. The artificial noise introduced by these methods increases the complexity of computational attacks dramatically. It has been shown in [BRVW19] that the complexity of attacks increases with the order of the masking and the order of the obfuscation layers. Therefore, the gray-box adversarial model is a valid approach to analyze the security of masking schemes of white-box implementations against computational attacks. Due to the artificial noise sources, it becomes infeasible for an attacker to combine the required number of shares to recover the sensitive information.
Furthermore, algebraic attacks are able to break masked WBC independently of *the masking orders if the masking is linear* as in Eq. (2.3). Thus, all current masking proposals resisting DCA are vulnerable to algebraic attacks. Only the scheme defined by [BU18] resists first-order algebraic attacks due to its *non-linear* structure, but as we

will show, does not resist computational attacks.

Another challenge for the secure constructions is the source of randomness source. Masking schemes rely on the availability of good randomness, which is usually provided by secure RNGs, e.g. in the form of a secure and efficient Pseudorandom Generator [IKL+13, CGZ19]. Similarly, randomness generation for white-box implementations has been analyzed in the literature. Due to the adversarial ability to control the execution environment in the white-box model, the attacker can simply disable any external randomness sources. Therefore, white-box implementations have to rely on internal randomness sources in combination with additional obfuscation countermeasures [BBIJ17, BU18, BRVW19].

The effectiveness of computational attacks comes from its universality and its ability to avoid reverse-engineering, which can be extremely costly [GPRW19]. By combining masking with an obfuscation layer, the adversary is thus again forced to do a time-consuming reverse engineering step to bypass the obfuscation, while the masking prevents obfuscation-oblivious attacks.

Throughout this thesis we assume a reliable randomness source is provided as part of the implementation, in other words, randomness can be provided via pseudorandom values derived from the input and protected by obfuscation layers, as done in [BBIJ17, BU18, RW19]. Therefore, the attacks on randomness sources and the adversaries' ability to disable randomness are out-of-scope in this work. For a full white-box implementation, other techniques (fault protection, randomness generation, obscurity layers) need to be added [BU18, BRVW19] in addition to a secure masking scheme.

3

Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks

3.1 Motivation

This chapter examines the fault-resistance of the glitch-free secure multiparty circuits proposed by Roche and Prouff [RP11, RP12] and proposes a new combined protection scheme for both side-channel and fault attacks with its security proof in the ISW probing model [ISW03].

We start with analyzing the fault behavior of the operations, namely affine transformation, squaring of a secret share, addition of two secret shares, and multiplication of two secret shares. It is shown that, while most parts of the glitch-free Secure Multi-party Computation (SMC) can be naturally extended to detect faults, the multiplication operation makes faults undetectable and the circuits become vulnerable to fault attacks. We propose a new multiplication circuit that properly maintains fault information and thus allows for the composition of glitch-free fault-detecting SMCs in which errors are propagated by the algebraic operations. Thus, the fault information will be detectable until the end of the circuit. Therefore, we eliminate the cost of fault detection. We introduce a new recombination operation which randomizes the output, if an error occurred anywhere in the circuit, ensuring that the attacker cannot learn anything from

the output.

We are able to construct arbitrary fault-resistant circuits using the basic arithmetic operations and a new recombination-fault detection operations. As a result, the attacker gets random outputs at the end of the cryptographic operations. Our scheme differs from previous proposals, as it does not have the same requirement of $n \geq 3d + 1$ to detect d *cheaters* in an (n, d)-secret sharing scheme. Instead, our scheme can detect up to ε errors, where $n > 2d + \varepsilon$ with very high probability. In fact, the detection probability is 1 after the first operation on faulty shares, but can slightly decrease, depending on the number of subsequent additions and multiplications. Even in the worst case, if only one of the inputs of the operations is faulty, the faulty output can always be detectable. Moreover, we provide a secrecy-preserving fault detection operation to increase the fault detection capabilities of the users as an option. This operation provides a trade-off between performance and security. It can be used securely (i.e., without leaking sensitive information) and therefore, perfect fault detection can be achieved.

To be able to prove the security against probing attacks, we follow the t-SNI security notions and give the formal proof of our schemes. First, we reformulate the t-SNI security notions to cover arbitrary (n, d)-secret sharing schemes. Then, we show that each operation defined in this work satisfies t-probing security. Since, the new multiplication scheme is an extension of the one used in [RP12], the first formal security proof of the multiplication scheme [RP12] is provided within this work. Moreover, we introduce a refresh masking operation for polynomial masking to construct complete t-SNI algorithms.

To analyze the fault detection properties of our scheme we use *Coverage* [SMG16] and define a new security notion *Propagation*, which states the probability of detecting faults using output shares if the input shares are faulty. We give a theoretical analysis of the proposed scheme using the new security notion. Moreover, we examine the fault resistance of the scheme by a simulation written in `SAGE`.

Implementation and leakage analysis of the original scheme was performed in hardware [MM13] and Grosso et al. [GSF14] analyze its performance with several methods for speeding up polynomial masking. To the best of our knowledge, however, there are no software implementations of the scheme tested under a comprehensive leakage assessment. Thus, we propose a practical C implementation tested on a popular ultra-low power architecture, the ARM Cortex M0+ core. Our analysis goes beyond the implementation itself by demonstrating its level of side-channel resistance and measuring its performance. The implementation provides multiple masking schemes easily portable to higher orders. However, by precomputing and inserting different public shares and corresponding

Vandermonde matrices in the code, other orders of masking can be employed without modification of the operations and functions. To show the side-channel resistance of our implementation, we address a full leakage analysis including higher order moments on the SMC multiplication. High assurance of the mask quality is ensured through utilizing a built-in true random number generator available on the MCU. These tests enable us to see the relation between processed sensitive variables and side-channel leakage, the results show how they are statistically independent.

The code has also more advanced constant-time features: we present different types of field multiplication, some of which rely on input-dependent table accesses and thus give better performance. However, we also present true constant-time multiplication, which is slower, but is constant-time even on systems featuring caches. All of these schemes execute in constant time regardless of the selected version of field operations. To that end, the code features a *fully constant execution flow with constant memory accesses* and is available as open source.

https://github.com/vernamlab/Robust-AES

Notation. In the protocol, the trusted dealer generates a random polynomial, $F(\mathsf{x}) = x + f_1\mathsf{x} + \ldots + f_d\mathsf{x}^d$ to share a secret x. The secret value is shared by evaluating $F(\mathsf{x})$ for n distinct and nonzero public points $(\alpha_0, \ldots, \alpha_{n-1})$.[1] The shared representation of x is shown as $F_x = (F(\alpha_0), \ldots, F(\alpha_{n-1})) = (F_0, \ldots, F_{n-1})$ and throughout the work, we denote them as *secret shares* or *secret states*. As seen in the Eq. (2.5) in a valid secret sharing scheme $f_{d+1} = \ldots = f_{n-1} = 0$. This fact is used in the following sections to detect faults.

We denote the secret values as x and y from a fixed finite field $\mathbb{F}$ and the nonzero public points as $(\alpha_0, \ldots, \alpha_{n-1})$. To generate corresponding (n, d)-secret sharing schemes, the trusted dealer generates two random polynomials $F(\mathsf{x}) = x \oplus f_1\mathsf{x} \oplus \ldots \oplus f_d\mathsf{x}^d \in \mathbb{F}[\mathsf{x}]$ and $G(\mathsf{x}) = y \oplus g_1\mathsf{x} \oplus \ldots \oplus g_d\mathsf{x}^d \in \mathbb{F}[\mathsf{x}]$. Therefore, the sets $(F_0, \ldots, F_{n-1})$ and $(G_0, \ldots, G_{n-1})$ represent the sets of the secret states of x and y respectively. Moreover, a valid secret sharing means when we perform a polynomial interpolation to the set $(F_0, \ldots, F_{n-1})$, the $\deg(F(x))$ will be less then or equal to d. Similarly, an invalid secret sharing means $\deg(F(x)) > d$.

[1] In this chapter we adapt the SMC literature and denote a secret sharing with n player.

3.2 SMC as a Fault Injection Countermeasure

In this section, we describe our fault model and extend the description of Shamir's Secret Sharing as outlined in Section 2.1.1

In our model, we address data targeted faults on the secret state only. As studied in most of the works, fault injections on the control flow are excluded from the model and need to be prevented by other means. The attack model is similar to known-value faults defined in [RMB+17] or the extreme case of a biased fault model in [SMG16]. It can be described as *blindly-chosen*, *non-adaptive*, and *additive* faults. That is, the attacker can pre-calculate a set of faults and induce single or multiple faults from this set to the secret states. The attacker can target any instance of the operation, can choose the specific elements from the secret states, and can inject multiple faults in one clock cycle. More precisely, the logical effect of a fault σ to a share F_i will be $F_i \oplus \sigma$ and the faulty state denoted by F_i'. Therefore, the *additive fault model* describes a wide class of errors that can be observed in practice, such as flipping one bit of F_i. Using this model, we eliminate cases like limitation of Hamming weight or selecting faults from a distribution. Faults on randomness are also allowed due to the *additive* fault property: Adding a blindly-chosen fault to an unknown random value results in an unknown random value.

Since the computations continue with the faulty state F_i', the degree of the polynomial generated by the faulty state (denoted by $F'(\mathsf{x})$) is greater than d with a high probability, so we can detect the faults by checking the degree of the secret sharing polynomial. Therefore the effect of multiple fault injections in one clock cycle will be the same in our analysis. The advantage of the additive model is that we can clearly define the relation between the secret state and the secret sharing polynomial using the Vandermonde representation. To examine this relation, we use the matrix representation of Eq. (2.5) as in Section 2.1.1: $V^{-1}S = C$.

$$V^{-1} \cdot \underbrace{\begin{pmatrix} F_0 \\ \vdots \\ F_{i-1} \\ F_i \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Secret states.}} = \underbrace{\begin{pmatrix} x \\ \vdots \\ f_d \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\substack{\text{The coefficients} \\ \text{of } F(\mathsf{x}).}} \xrightarrow{\text{Fault Injection}} V^{-1} \cdot \underbrace{\begin{pmatrix} F_0 \\ \vdots \\ F_{i-1} \\ F_i' \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Faulty states.}} = \underbrace{\begin{pmatrix} x' \\ \vdots \\ f_d' \\ f_{d+1}' \\ \vdots \\ f_{n-1}' \end{pmatrix}}_{\substack{\text{The coefficients} \\ \text{of } F'(x).}} \tag{3.1}$$

According to the additive fault model, the relation between $F'(\mathsf{x})$ and $F(\mathsf{x})$ can be summarized as $F'(\mathsf{x}) = F(\mathsf{x}) \oplus \Delta(\mathsf{x})$, where $\Delta(\mathsf{x})$ is the polynomial generated by faults

i.e. interpolated by the points, $(0,\ldots,0,\sigma,0,\ldots,0)$ as seen in Eq. (3.2).

$$V^{-1}\cdot\begin{pmatrix}F_0\\ \vdots\\ F_{i-1}\\ F_i'\\ \vdots\\ F_{n-1}\end{pmatrix}=V^{-1}\cdot\left[\begin{pmatrix}F_0\\ \vdots\\ F_{i-1}\\ F_i\\ \vdots\\ F_{n-1}\end{pmatrix}\oplus\begin{pmatrix}0\\ \vdots\\ 0\\ \sigma\\ \vdots\\ 0\end{pmatrix}\right]=\begin{pmatrix}x\\ \vdots\\ f_d\\ 0\\ \vdots\\ 0\end{pmatrix}\oplus\begin{pmatrix}\sigma_0\\ \vdots\\ \sigma_d\\ f'_{d+1}\\ \vdots\\ f'_{n-1}\end{pmatrix}\tag{3.2}$$

For an (n,d)-secret sharing scheme, faults are undetectable if and only if the degree of $\Delta(\mathsf{x})$ is smaller than or equal to d, i.e., the coefficients of the terms of degree $d{+}1,\ldots,n{-}1$ are zero. We refer to these terms as *error detection terms*.

3.2.1 Fault Detection and Propogation

Another important feature of secure multiparty computation schemes is that they can be used to detect faults. Depending on the number of faulty shares, errors are *detectable*, *undetectable*, or *correctable*. To be able to *correct* faults on d shares, previously proposed schemes require at least $3d+1$ shares [Ben88]. Furthermore, *robust* multiplication requires cheater detection at the input and the output of every multiplication, which is a very costly operation [GRR98].
Fault injection countermeasures are less concerned with the correction of errors. The main goal is usually to *detect* faults and to ensure nothing can be learned from a faulty output. Therefore, our aim is to preserve faults and detect them only once the output is produced. To be able to detect the faults without conducting additional operations, we need to observe the propagation of the faulty state for each SMC component. In this section, we discuss the preservation of the faulty state and show the vulnerabilities of the computations. Remark that, in the following analysis, we assume that at least one of the input polynomials is faulty, i.e. $\deg(F(\mathsf{x}))>d$ or $\deg(G(\mathsf{x}))>d$.
Let x and y be two values and consider an $(n,d)^{th}$-order masking scheme, i.e. x and y have been split into (n) shares $\{F_0,\ldots,F_{n-1}\}$ and $\{G_0,\ldots,G_{n-1}\}$ such that $F(0)=x$ and $G(0)=y$.

- **Affine transformation of a secret (`Affine`):** An affine transformation $\mathsf{L}(\mathsf{x})=a\mathsf{x}\oplus b$ with $a\neq 0$ can be computed on the secret value by applying L to secret shares locally; also each player P_i computes its component by $\mathsf{L}(F_i)$. If $\deg(F(\mathsf{x}))>d$, then clearly $\deg(\mathsf{L}(F(\mathsf{x})))>d$ as well. $\mathsf{L}(\mathsf{x})$ changes the faults only in magnitude while the localization of the faults is preserved. Moreover, the behavior of the

faulty state is the same if the faults are injected during the computation of the affine transformation.

- **Addition of two secrets (`Add`):** Two secret values $x \oplus y$ can be added by pairwisely adding shares. Each player P_i computes $F_i \oplus G_i$. According to our fault model, if only one polynomial is faulty, i.e. has a degree greater than d, then the degree of the resulting polynomial will be also greater than d and therefore the faults are propagated. However, an attacker can inject faults to both polynomials in different or in the same shares. In these cases, there is a probability that faults can become undetectable. The error detection terms can be zero if corresponding coefficients of $F(\mathsf{x})$ and $G(\mathsf{x})$ are equal, that is:

$$f_{d+1} \oplus g_{d+1} = \ldots = f_{n-1} \oplus g_{n-1} = 0.$$

  As we assumed, $\deg(F(\mathsf{x})) > d$ and $\deg(G(\mathsf{x})) > d$, $f_i \neq 0$ and $g_j \neq 0$ for at least one $i, j \in \{d+1, \ldots, n-1\}$. So, $Pr[(f_i \oplus g_i = 0)_{d+1 \leq i < n}] \approx (1/|\mathbb{F}|)^{d+\varepsilon}$. [2]

- **Efficient squaring operation ($\mathtt{Sqr}_k$):** Efficient squaring operations can be used to eliminate costly multiplications in $\mathrm{GF}(2^m)$ [RP12]. Squaring can be defined as $\eta_k(y) = y^{2^k}$ and requires two conditions on public points α_i :

  1. $\alpha_i \neq 0$ for $i = 0, \ldots, n-1$.
  2. For every α_i there exists α_j such that $\alpha_i^2 = \alpha_j$.

  Each player calculates the operation on its share locally by $\eta_k(F_i)$. The family of shares $(\eta_k(F_i))_{i=0,\ldots n-1}$ is a valid secret share of x^{2^k}. However, communication between players is needed to do the reordering of the secret shares. Faulty shares are preserved as in the affine transformation.

- **Multiplication of two secrets (`Mult`):** The multiplication of two secret values $x \cdot y$ requires communication between players. An efficient algorithm was proposed by Gennaro et al. [GRR98], simplifying the original SMC multiplication proposed by Ben-Or et al. [Ben88]:

  1. Each player P_i computes $H_i = F_i \cdot G_i$.
  2. Each player P_i generates a degree d polynomial $\mathcal{Q}_i(\mathsf{x})$ such that $\mathcal{Q}_i(0) = H(\alpha_i)$, and sends the value $\mathcal{Q}_i(\alpha_j)$ to player P_j. In this step, faults in $F(\mathsf{x})$ and $G(\mathsf{x})$ spread to all shares and they become undetectable, since $\mathcal{Q}_i(\mathsf{x})$ is a degree d polynomial.

[2] The exact probability can be calculated as $(1/|\mathbb{F}^*|) \cdot (1/|\mathbb{F}|)^{d+\varepsilon}$.

3. Each player P_i computes its secret share by $\mathbf{Q}_i = \sum_{j=0}^{n} \lambda_j^0 \mathcal{Q}_{j,i}$ and gets a *valid* (n, d)-secret sharing of the faulty secret value.

The shares calculated in step 1 cannot be used as a valid secret sharing because of two main problems: (1) the polynomial is a degree $2d$ polynomial. (2) It is not a random polynomial [Ben88]. In step 2 and 3, degree reduction and randomization are done in order to generate a *proper* (n, d)-secret sharing of xy. As a result, the output shares are always be a valid secret sharing and an adversary can inject faults without detection.

Remark 3.1. *Castagnos et al. [CRZ13] suggested an improvement of the SMC multiplication by using the connection between Reed-Solomon codes [Ber68] and Shamir's secret sharing scheme. Although Reed-solomon codes provide extensions and generalization of the sharing [MS], the improved secure multiplications have the same undetectable fault problem. The improvement focuses on the randomization part of the multiplication scheme (or encoding procedure as stated in [CRZ13]). The output is produced by the Re-encoding algorithm ([CRZ13], Algorithm 3). The output of the algorithm is always a valid secret state. Therefore, a faulty input will always result in a valid secret sharing and faults become undetectable.*

3.3 Error Preserving Multiparty Computation

The error-preserving multiparty computation scheme below differs significantly from other proposals, such as robust SMC. Unlike, e. g., [GRR98, GIP+14], detecting errors after each operation is not convenient in many cryptographic implementations, as it can reveal critical information. The basic ideas of our scheme are as follows:

- **Error Detection Only:** Our scheme does neither try to correct errors, nor detect where the error occurred. As in most application scenarios, the scheme only aims at detecting the errors and ensures that the attacker cannot learn anything from a faulty output.
- **Fault Detection Without Leaking Information:** The scheme aims to eliminate the leakages that can occur during the detection and keep the fault detection probability as one.
- **Error-Preserving Computation:** Once an errors occurs, it will spread through the state and remain part of the state. The advantage of this is that error detection can be performed once an output is produced.

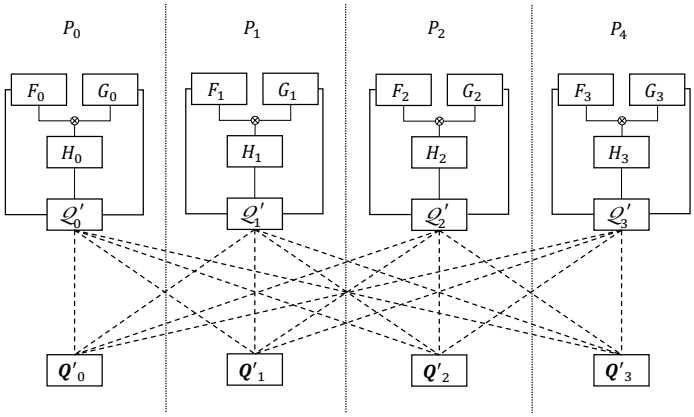


Figure 3.1: A detailed visualization of (4,1)-SMC Multiplication. Dashed lines show the communications between players (P_i). The shares of error detection terms of $H(x)$, $F(\mathsf{x})$, and $G(\mathsf{x})$ are forwarded to shares of xy during the randomization step. Thus, each $\mathbf{Q}'_i$ contains a share of error detection terms of $F(\mathsf{x})$, $G(\mathsf{x})$, or $H(\mathsf{x})$.

- **Infective Computation:** If an error occurs, it is important to ensures the output does not reveal information to the attacker. We show that the randomization property of the secret sharing together with the redundant error detection coefficients ensure random outputs of faulty parts of the state if an error occurs.

Most of these goals can be achieved with the SMC described in [RP12] in a straightforward manner. However, the *multiplication* is difficult to construct in such a way that error detection is not performed once for each multiplication on each input and output. Instead, we propose a new multiplication engine that, in addition to the shared inputs and outputs, also uses additional shared error detection coefficients. The main advantage of error detection coefficients is that they add redundancy while only introducing minor overhead. In summary, all circuits can be represented by a classic SMC addition, our updated SMC multiplication, and a new recombination step. SMC squaring and affine transformation can still be used as before, as they do not influence the fault propagation.

3.3.1 Error Preserving Multiplication (`EPMult`)

Multiplication is the most critical SMC operation. Even without error detection, multiplication is the reason why $n > 2d$ is required, since the product of two degree-d polynomials is of degree-$2d$, To achieve error detection, more shares are needed. In fact, we show that to detect ε errors, a total of $n > 2d+\varepsilon$ shares are needed. A representation of the (4,1)-error preserving multiplication can be seen in Fig. 3.1.
In the new scheme, the error propagation is achieved by using the *error detection terms*

Algorithm 2 Error Preserving Multiplication (`EPMult`)

Input: Shares of f_0 as $(F_i)_{0\leq i<n}$ and shares of g_0 as $(G_i)_{0\leq i<n}$.
Output: Shares of $f_0 g_0$ as $(\mathbf{Q}_i)_{0\leq i<n}$.

1: **for** $i = 0$ **to** $n-1$
2: $\quad H_i \leftarrow F_i G_i$
3: **for** $i = 0$ **to** $n-1$
4: $\quad (r_{i,1}, \ldots, r_{i,d}) \leftarrow \mathbb{F}_{2^m}$ // Coefficients of the random polynomial.
5: $\quad$ **for** $j = 0$ **to** $n-1$
6: $\quad\quad \mathcal{Q}_{i,j} \leftarrow H_i$ // referred to as $\mathcal{Q}^0_{j,i}$.
7: $\quad\quad$ **for** $k = 1$ **to** d // Evaluate the polynomial.
8: $\quad\quad\quad \mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus r_{i,k}\alpha_j^k$ // referred to as $\mathcal{Q}^k_{j,i}$.
9: $\quad\quad \mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus \mathsf{E}_{i,j}$ // Add a share of an error detection term.
10: $\quad\quad \mathbf{Q}_j \leftarrow \mathbf{Q}_j \oplus \lambda_i^0 \mathcal{Q}_{i,j}$ // referred to as $\mathbf{Q}_{j,i}$.
11: **return** $(\mathbf{Q}_0, \ldots, \mathbf{Q}_{n-1})$

of input polynomials and the intermediate polynomial $H(x)$. A step-by-step description of our new multiplication scheme that can resist ε faults can be introduced as follows:

1. Each player P_i locally computes $H_i = F_i \cdot G_i$.

2. Each player P_i generates a degree d polynomial $\mathcal{Q}_i(x)$ such that $\mathcal{Q}_i(0) = H_i$ and evaluates the polynomial for the public points $\mathcal{Q}_i(\alpha_j)$ (denoted by $\mathcal{Q}_{i,j}$). The main difference in our scheme occurs in this step: P_i also calculates a share of error detection coefficients (denoted by $\mathsf{E}_{i,j}$) of H_i or $G_i \oplus F_i$ with the corresponding Vandermonde element. The resulting $\mathcal{Q}'_{i,j} = \mathcal{Q}_{i,j} \oplus \mathsf{E}_{i,j}$ is sent to player P_j for $j = 0, \ldots, n-1$ and $j \neq i$. $\mathsf{E}_{i,j}$ is defined as follows:

$$\mathsf{E}_{i,j} = \begin{cases} \frac{\lambda_i^{n-j}}{\lambda_i^0} H_i & \text{if } 0 \leq j < \varepsilon \\ \frac{\lambda_i^{n-j}}{\lambda_i^0}(F_i \oplus G_i) & \text{if } \varepsilon \leq j < \varepsilon + d \\ 0 & \text{if } \varepsilon + d \leq j < n \end{cases}. \tag{3.3}$$

3. In the third step, each player calculates its share by $\mathbf{Q}'_i = \sum_{j=0}^{n-1} \lambda_i^0 \mathcal{Q}'_{j,i}$.

Remark 3.2. *The 11^{th} line in the Algorithm 2 corresponds to the error propagation. Without it, the `EPMult` corresponds to the SMC-Multiplication defined by Gennaro et al. [GRR98] and applied by Roche and Prouff [RP12]. The advantage of $\mathsf{E}_{i,j}$ is that it ensures the propagation of error detection terms as faults to the output by using only the local information. Therefore, the first $\varepsilon + d$ players implicitly get an error detection*

coefficient of $H(x)$ or $F(x) \oplus G(x)$. For example, player P_i $(0 \leq i < \varepsilon)$ calculates its share $\boldsymbol{Q}'_i$ by:

$$\begin{aligned}
\boldsymbol{Q}'_i &= \sum_{j=0}^{n-1} \lambda_i^0 \mathcal{Q}'_{j,i} = \sum_{j=0}^{n-1} \lambda_i^0 (\mathcal{Q}_{j,i} \oplus \mathsf{E}_{j,i}) \\
&= \left[\lambda_0^0 \mathcal{Q}_{0,i} \oplus \lambda_0^{n-i} H_0\right] \oplus \ldots \oplus \left[\lambda_{n-1}^0 \mathcal{Q}_{n-1,i} \oplus \lambda_{n-1}^{n-i} H_{n-1}\right] \\
&= \underbrace{\left[\lambda_0^0 \mathcal{Q}_{0,i} \oplus \ldots \oplus \lambda_{n-1}^0 \mathcal{Q}_{n-1,i}\right]}_{=\boldsymbol{Q}_i \text{ in Section 3.2.1 SMC mult.}} \oplus \underbrace{\left[\lambda_0^{n-i} H_0 \oplus \ldots \oplus \lambda_n^{n-i} H_{n-1}\right]}_{=h_{n-i-1} \text{ by Eq. (3.1)}} \\
&= \boldsymbol{Q}_i \oplus h_{n-i-1}.
\end{aligned}$$

The propagation of the faults within the output shares can be summarized as follows:

$$\boldsymbol{Q}'_i = \begin{cases} \boldsymbol{Q}_i \oplus h_{n-i-1} & \text{if } 0 \leq i < \varepsilon \\ \boldsymbol{Q}_i \oplus g_{n-i-1} \oplus f_{n-i-1} & \text{if } \varepsilon \leq i < \varepsilon + d \\ \boldsymbol{Q}_i & \text{if } \varepsilon + d \leq i < n \end{cases},$$

where h_i, g_i and f_i represent i^{th} degree the coefficients of H, G and F, respectively. Remark that, $(h_i)_{2d<i<n} = 0$ and $(f_i \oplus g_i)_{d<i<n} = 0$ for valid secret sharing schemes.

3.3.2 Fault Detection Operation (`FDect`)

In order to maintain the error detection probability as one, faults need to be detected before each multiplication and addition. Detection can be performed according to the Vandermonde representation. However, this operation can leak sensitive information. Therefore, we add a randomization step to provide the confidentiality of the secret value.

1. ***Randomization:*** This step adds a random degree-d polynomial to the shared secret, thereby masking the secret value, but not deleting fault information.
   a) A random degree d polynomial $\mathcal{R}(\mathsf{x})$ is generated, the value $\mathcal{R}(\alpha_i)$ is sent (denoted by $\mathcal{R}_i$) to player P_i.
   b) Each player calculates the new share by simply adding the random share to their own share.
   $$F_{\mathcal{R}_i} = F_i \oplus \mathcal{R}_i \text{ for } i = 0, \ldots n-1.$$
2. ***Detection:*** Error detection coefficients are reconstructed in the natural way as given in Eq. (3.1):
   $$f_j = \sum_{i=0}^{n-1} \lambda_i^j F_{\mathcal{R}_i} \text{ for } j = d+1, \ldots, n-1.$$

Algorithm 3 Fault Detection Operation

Input: Shares of f_0 as $(F_i)_{0\leq i<n}$.
Output: Fault Decision.
1: $(r_0, \ldots, r_d) \leftarrow \mathbb{F}_{2^m}$ // Coefficients of the random polynomial.
2: **for** $i = 0$ **to** $n-1$ // ***Randomization.***
3: $\quad F_{\mathcal{R}_i} \leftarrow F_i$
4: $\quad$ **for** $k = 0$ **to** d // Evaluate the polynomial.
5: $\quad\quad \mathcal{R}_i \leftarrow \mathcal{R}_i \oplus r_k \alpha_j^k$ // Referred to as $\mathcal{R}_i^k$
6: $\quad F_{\mathcal{R}_i} \leftarrow F_{\mathcal{R}_i} \oplus \mathcal{R}_i$
7: Fault Detection using the set of secret shares: $(F_{\mathcal{R}_i})_{0\leq i<n}$// ***Detection.***

The randomized share of a player (denoted by $F_{\mathcal{R}_i}$) corresponds to a secret sharing of a random value. Therefore, the reconstruction cannot leak information about the real secret value. Moreover, $F_{\mathcal{R}}(\mathsf{x}) = F(\mathsf{x}) \oplus \mathcal{R}(\mathsf{x})$, and we know that $\deg(\mathcal{R}_i(\mathsf{x})) \leq d$. Therefore, if $F(\mathsf{x})$ is faulty, i.e. $\deg(F(\mathsf{x})) > d$, then $\deg(F_{\mathcal{R}}(\mathsf{x})) > d$, and fault detection can be achieved easily in the *detection* step by reconstructing and checking the error detection terms. The algorithm can be found in Algorithm 3.

Remark 3.3. *While in-circuit fault detection is an option, our aim is not to give any error message or stop the execution. The scheme outputs the faulty ciphertext even if the fault is detected. The degree of the polynomial is used as a fault flag. The output can then be randomized using this flag.*

3.3.3 Recombination Operation (`ReComb`) and Infective Computation

In order to avoid costly fault detection operations, we first propose a recombination operation which detects the faults when the output is produced. We explain the *infectiousness* of the faults while introducing a recombination algorithm.

The recombination operation is composed of two main steps, ***re-sharing*** and ***reconstruction***. The main idea is to share the secret variable while adding randomized error detection terms. The first part can be seen as a modified version of `EPMult` using a different $\mathsf{E}_{i,j}$. The details can be found in Algorithm 4. The inputs of the operation are secret shares F_i for $0 \leq i < n$ and a random vector $(\mathsf{r}_0, \ldots, \mathsf{r}_{\varepsilon+d-1})$ where $\mathsf{r}_i \in \mathrm{GF}(2^8) \setminus 0$. The outputs are the secret value f_0 and a fault decision.

1. ***Re-Sharing***: Players share the secret value as in the second part of the `EPMult`,

Algorithm 4 Recombination Operation

Input: Shares of f_0 as $(F_i)_{0\leq i<n}$ and non-zero random values $(\mathsf{r}_0, \ldots, \mathsf{r}_{\varepsilon+d-1})$.
Output: f_0 and Fault Decision.

1: **for** $i = 0$ **to** $n-1$ // Re-Sharing
2: $(r_{i,1}, \ldots, r_{i,d}) \leftarrow \mathbb{F}_{2^m}$ // Coefficients of the random polynomial.
3: **for** $j = 0$ **to** $n-1$
4: $\mathcal{Q}_{i,j} \leftarrow F_i$ // referred by $\mathcal{Q}^0_{j,i}$.
5: **for** $k = 1$ **to** d // Evaluate the polynomial.
6: $\mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus r_k \alpha_j^k$ // referred to as $\mathcal{Q}^k_{j,i}$.
7: $\mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus \mathsf{E}^R_{i,j}$ // Add a share of an error detection term.
8: $\mathbf{Q}_j \leftarrow \mathbf{Q}_j \oplus \lambda_i^0 \mathcal{Q}_{i,j}$ // referred to as $\mathbf{Q}_{j,i}$.
9: Reconstruction using the set of secret shares: $(\mathbf{Q}_i)_{0\leq i<n}$// ***Reconstruction.***

the only difference is that we update $\mathsf{E}_{i,j}$ as follows:

$$\mathsf{E}^R_{i,j} = \begin{cases} \mathsf{r}_j \frac{\lambda_i^{n-(j-1)}}{\lambda_i^0} F_i & \text{if } 0 \leq j < d+\varepsilon \\ 0 & \text{if } d+\varepsilon \leq j \leq n-1 \end{cases}.$$

The adversary is still able to get information from the output, so we ensure the randomization of the secret value by using fresh random values.

a) Each player P_i generates a degree d polynomial $\mathcal{Q}_i(\mathsf{x})$ such that $\mathcal{Q}_i(0) = F_i$ and evaluates the polynomial for the public shares $\mathcal{Q}_i(\alpha_i)$ (denoted by $\mathcal{Q}_{i,j}$) and sends the value $\mathcal{Q}'_i(x) = \mathcal{Q}_i(x) \oplus \mathsf{E}^R_{i,j}$ to player P_j.

b) Each player calculates its new share $\mathbf{Q}'_i$ by $\sum_{j=0}^{n-1} \lambda_i^0 \mathcal{Q}'_{j,i}$.

Remark 3.4. *As in the* `EPMult`*, the first $\varepsilon+d$ players implicitly get the randomized error detection coefficient of $F(x)$.*

$$\boldsymbol{Q}'_i = \begin{cases} \boldsymbol{Q}_i \oplus \mathsf{r}_i f_{n-i-1} & \text{if } 0 \leq j < \varepsilon + d \\ \boldsymbol{Q}_i & \text{if } \varepsilon + d \leq j < n \end{cases}.$$

Reconstruction. In the last step, the secret value and error detection coefficients are reconstructed using Equation (3.2):

$$f_j = \sum_{i=0}^{n-1} \lambda_i^j \mathbf{Q}'_i \text{ for } j = d+1, \ldots, n-1 \text{ and } 0.$$

Clearly, if F is faulty, then at least one of the error detection terms is non-zero. In the second step, the secret value is randomized by these terms, and, therefore, the output is randomized in case of fault injections. Thus, infective computation is achieved. The details and security features of the operation can be found in Section 3.4.

3.4 Security Analysis

Previously proposed schemes that combined countermeasures against side-channel analysis and fault injection have different security claims depending on their used adversarial models. The SCA security of works like [IPSW06] are based on the ISW transformation [ISW03], while others [SMG16, DCN16] are based on Threshold implementations. Both the ISW transformation and TI can achieve t^{th}-order security. However, it was recently shown Moos et al. [MMSS18] that TI and derived schemes can have security issues due to the insufficient refreshing in higher order variants and thus require special care during implementation. There are bigger differences in the fault resistance properties of the proposed schemes. For example, in [IPSW06] security against *reset attacks* and *set, reset* and *toggle attacks* are formally proven. In [SMG16] authors defined a notion called *Coverage* to quantify the fault coverage of their scheme. They analyzed the fault resistance of the scheme using this notion. Similarly, in [RMB+17] the authors examined the conditions where faults are undetectable. In this section, we discuss the security features of our combined countermeasure under specific attack models. We formally prove the t-probing security of individual operations and analyze the side-channel resistance of a combination of operations. Then a similar discussion in [SMG16, RMB+17] will be done to explain the fault resistance of our scheme. First we define a new notion called *Propagation*, which states the probability of detecting faults using output shares if the input shares are faulty. Then we examine the conditions where faults are undetectable for each individual operation and for a combination of operations.

3.4.1 Side-Channel Resistance

As stated in Section 2.1, t-SNI security notion becomes the standard way of proving the security against probing attacks. However, we cannot use the security notion directly. The notion is specialized for Boolean masking where $(n-1)$-tuples of n intermediate variables are uniformly distributed. On the other hand, an (n, d)-secret sharing corresponds to n intermediate variables such that every d-tuple of them is uniformly distributed and independent of any sensitive variable instead of $(n-1)$-tuple.

Therefore, we extend the definition to cover (n, d)-SMC and we focus on the modified version of the security notion defined as follows:

Definition 3.1 (t-SNI$_d^n$ Security). Let G be a gadget which takes as input n shares $(x_i)_{0\leq i<n}$ and as outputs n shares $(y_i)_{0\leq i<n}$. The gadget G is said to be t-SNI$_d^n$ secure if for any set of t probed intermediate variables and any subset $\mathcal{O} \subset [0, n)$ of output indices, such that $t + |\mathcal{O}| < d + 1$, there exists a subset $I \subset [0, n)$ of input indices which satisfies $|I| \leq t$, such that the t intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.

Clearly, the original definition corresponds to t-SNI$_{n-1}^n$ in our notation. We show that it is possible to construct t probes as well as a set of output shares $\mathcal{O}$ such that $t + |\mathcal{O}| < d + 1$, using a subset of input shares with at most t elements. The elements in the subset will be uniformly distributed and be independent from the shared secret value. It should be noted that, the set of probed variables and the output shares can be perfectly simulated by d-tuple of random variables. Therefore, the modified security notion will be equivalent to the original definition and the *perfect t-probing security* defined by Carlet et al. [CPRR15] as well as Lemma 1 in [RP10].

Theorem 3.1 (t-SNI$_d^n$ of `EPMult`). *Let $(F_i)_{0\leq i<n}$ and $(G_i)_{0\leq i<n}$ be the input shares of the Error Preserving Multiplication operation, and let $(\boldsymbol{Q}_i)_{0\leq i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < d + 1$, there exist two subsets I and J of indices with $|I| \leq t_1$ and $|J| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $\boldsymbol{Q}_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$ and $G_{|J}$.*

Proof. In the first part of the proof, we construct the sets of the input share indices I and J depending on the intermediate variables that are probed. We denote U as the intersection of I and J. We divide the probes into 2 groups.

- **Group 1:** If F_i or G_i is probed, add i to I or J, respectively. If H_i, $\mathcal{Q}_{i,j}^0$ or $\mathsf{E}_{i,j}$ is probed, add i to I and J.
- **Group 2:** If $r_{i,j}$ or $\mathcal{Q}_{i,j}^k$ where $k \in \{1, \ldots, d\}$ is probed, add i to I and J.

According to our selection, we add at most one index to I and J for each probe and, therefore, $|I| \leq t_1$ and $|J| \leq t_1$.

1. The simulations of the probed variables in **Group 1** are straightforward. Since $i \in I$ ($i \in J$ resp.), we can perfectly simulate F_i (G_i resp.), because F_i and

G_i are known values. Similarly, we can simulate H_i and $\mathcal{Q}^0_{i,j}$, since $i \in I$ and $i \in J$. Finally, since the elements λ_i of the inverse Vandermonde matrix are public variables, we can simulate $\mathsf{E}_{i,j}$ as defined in Eq. (3.3).

2. If $\mathcal{Q}^k_{i,j}$ is probed, we need to consider two cases:
   - If $r_{i,k}$ is also probed, we leave $r_{i,k}$ as in the real circuit, therefore, we can simulate $\mathcal{Q}^k_{i,j}$ as $H_i \oplus r_{i,k}\alpha^k_j$ where α_j is a public value.
   - If $r_{i,k}$ is not probed, it does not enter into the computations of $\mathcal{Q}^k_{i,j}$, therefore, we can perfectly simulate $\mathcal{Q}^k_{i,j}$ with a random value.

3. If $\mathbf{Q}_{j,i}$ is probed, we need to consider two cases as in the previous step:
   - If all the values $r_{i,k}$ for $1 \leq k \leq d$ are probed, we can perfectly simulate the values $\mathcal{Q}^k_{i,j}$, and hence $\mathbf{Q}_{j,i}$ can be simulated. Remark that, the λ^0_i is an element of the inverse Vandermonde matrix so it is a public value.
   - If at least $r_{i,k}$ for $1 \leq k \leq d$ is not probed, that means $r_{i,k}$ does not enter into the computation of $\mathbf{Q}_{j,i}$, therefore $\mathbf{Q}_{j,i}$ can be simulated by a random value.

Now we explain how to simulate output shares $\mathbf{Q}_i$ for all $i \in \mathcal{O}$ where $\mathcal{O}$ is an arbitrary subset of $[1, n]$ with t_2 elements such that $t_1 + t_2 < d + 1$. Clearly, using t_1 probes, we can observe at most t_1 intermediate variables of $\mathbf{Q}_i$, where $\mathbf{Q}_i$ can be written as: $\mathbf{Q}_i = \sum_{j=0}^{n-1} \lambda^0_i \mathcal{Q}_{j,i}$. Since $t_1 + t_2 < d + 1$, at least one intermediate variable of $\mathbf{Q}_i$ is not probed. Therefore, we can simulate $\mathcal{Q}_{j,i}$ with $j \notin U$ by generating a random degree d polynomial and evaluating it for α_i. Hence, we can simulate $\mathbf{Q}_i$ for each $i \in \mathcal{O}$. □

We show that any set of t_1 intermediate variables and any subset of t_2 output shares can be perfectly simulated by at most d independent and uniformly distributed variables. Therefore, we can say that `EPMult` is secure in the d-probing model which is followed by security in the noisy leakage and bounded moment leakage models [DDF14, BDF+17]. Next, we give the security notion of SMC-addition, affine transformation, and efficient squaring. These operations perform sharewise computations. Therefore, they can be computed using affine gadgets as defined in [BBD+16]. As given in Section 2.1.3 composition of t-NI and t-SNI gadgets are also t-SNI with the help of a mask refreshing operation. For our work we denote this gadget by `RefreshM` and define in Algorithm 5. `RefreshM` ensures the independence of the inputs of the `EPMult` operation and we can implement an arbitrary function with t-SNI^n_d security. The following theorem provides the security of our `RefreshM` operation.

Algorithm 5 Mask Refreshing (`RefreshM`)

Input: Shares of f_0 as $(F_i)_{0\leq i<n}$.
Output: Shares of f_0 as $(C_i)_{0\leq i<n}$.
1: $(r_1, \ldots, r_d) \leftarrow \mathbb{F}_{2^m}$ // Coefficients of the random polynomial.
2: **for** $j = 0$ **to** $n-1$
3: **for** $k = 1$ **to** d // Evaluate the polynomial.
4: $\mathcal{Q}_j \leftarrow \mathcal{Q}_j \oplus r_k \alpha_j^k$ // referred to as $\mathcal{Q}_j^k$.
5: $C_j \leftarrow F_j \oplus \mathcal{Q}_j$
6: **return** $(C_0, \ldots, C_{n-1})$

Theorem 3.2 (t-SNI$_d^n$ of `RefreshM`). *Let $(F_i)_{0\leq i<n}$ be the input shares of `RefreshM` and let $(C_i)_{0\leq i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < d+1$, there exist a subset I of indices with $|I| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $C_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$.*

Proof. The proof is relatively straight forward. For every probed variable F_i, C_i, or $\mathcal{Q}_|$ add i to I. Clearly, we add at most one index to I and, therefore, $|I| \leq t_1$.

- If r_k or $\mathcal{Q}_j^k$ probed, we let the variables as in the circuit and perfectly simulate them.
- if C_i is probed, we can perfectly simulate the variable by $F_i \oplus \mathcal{Q}_i$, by letting $\mathcal{Q}_i$ as in the real circuit.

Therefore, we are able to simulate all the probed variables. Now, we consider the simulation of output variables. We need to show that C_i for $i \in \mathcal{O}$ can be simulated from $F_{|I}$. If $i \in I$, we can simulate C_i as explained above. We now examine the simulation of output variables C_i, where $i \notin I$. That means, $\mathcal{Q}_j$ is not probed and is not involved in the computation of C_i. Hence, we can perfectly simulate C_i by a random value. □

Furthermore, the security analysis of the fault detection and recombination operations can be found below. Before going into the proofs of fault detection operation and recombination operation, we need to clarify that the *detection* parts are excluded from the proofs to make definitions compatible. As given in Eq. (3.1), the detection mechanism requires all shares. However, during fault detection we already randomize the sensitive variable.

Theorem 3.3 (t-SNI$_d^n$ of Fault Detection Operation). *Let $(F_i)_{0\leq i<n}$ be the input shares of Fault Detection Operation and let $(F_{\mathcal{R}_i})_{0\leq i<n}$ be the output shares. For any*

set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < d + 1$, there exists a subset I of indices with $|I| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $F_{\mathcal{R}_{|\mathcal{O}}}$ can be perfectly simulated from $F_{|I}$.

Proof. The proof is very similar to the proof of Theorem 3.2. For every probed variable F_i, $F_{\mathcal{R}_i}$, or $\mathcal{R}_i^d$ add i to I. Clearly, we add at most one index to I and, therefore, $|I| \leq t_1$.

1. If $\mathcal{R}_i^k$ is probed, we can perfectly simulate it with a random value, since it does not depend on any variable.

2. If $F_{\mathcal{R}}$ is probed, we can simulate it as $F_i \oplus \mathcal{R}_i$. Note that, $\mathcal{R}_i$ can be simulated as in the first step.

Therefore, we are able to simulate all the probed variables. Now, we consider the simulation of output variables. We need to show that $F_{\mathcal{R}_i}$ for $i \in \mathcal{O}$ can be simulated from $F_{|I}$. If $i \in I$, we can simulate $F_{\mathcal{R}_i}$ as explained above. We now examine the simulation of output variables $F_{\mathcal{R}_i}$, where $i \notin I$. That means, $\mathcal{R}_i^d$ is not probed and is not involved in computation of $F_{\mathcal{R}_i}$. Hence, we can perfectly simulate $F_{\mathcal{R}_i}$ by a random value. □

Theorem 3.4 (t-SNI$_d^n$ of Recombination Operation). *Let $(F_i)_{0\leq i<n}$ be the input shares of the Recombination Operation and let $(\boldsymbol{Q}_i)_{0\leq i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < d + 1$, there exists a subset I of indices with $|I| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $\boldsymbol{Q}_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$.*

Proof. As stated in Section 3.3.3, the recombination operation can be seen as a modified version of `EPMult`, therefore, the proof is built on the same structure as in the proof of Theorem 7.2 in Section 3.4.
In the first part of the proof, we construct the sets of input share indices I depending on the intermediate variables that are probed. If F_i, $r_{i,j}$, $F_{j,i}^k$ or $E_{i,j}^R$ is probed, add i to I

- **Group 1:** If F_i or $\mathcal{F}_{i,j}^0$ or $\mathsf{E}_{i,j}^R$ is probed, add i to I.
- **Group 2:** If $\mathsf{E}_{i,j}^R$ or $\mathsf{r_j}$ is probed, add i to I.
- **Group 3:** If $r_{i,j}$ or $\mathcal{Q}_{i,j}^k$ where $k \in \{1, \ldots, d\}$ is probed, add i to I and J.

According to our selection, we add at most one index to I and J for each probe and, therefore, $|I| \leq t_1$ and $|J| \leq t_1$.

1. The simulations of the probed variables in group 1 are straightforward. Since $i \in I$, we can perfectly simulate F_i. Similarly, $\mathcal{Q}^0_{i,j}$, since $i \in I$.

2. Since the elements λ_i of the inverse Vandermonde matrix are public variables, we can simulate $\mathsf{E}^R_{i,j}$ as defined in Eq. (3.3). In fact we let r_j for $j \in \{0, \ldots, \varepsilon + d + 1\}$ as in the real circuit.

3. If $\mathcal{Q}^k_{i,j}$ is probed, we need to consider two cases:
   - If $r_{i,k}$ is also probed, we let $r_{i,k}$ as in the real circuit, therefore, we can simulate $\mathcal{Q}^k_{i,j}$ as $F_i \oplus r_{i,k}\alpha^k_j$ where α_j is a public value.
   - If $r_{i,k}$ is not probed, it does not enter into the computations of $\mathcal{Q}^k_{i,j}$, therefore, we can perfectly simulate $\mathcal{Q}^k_{i,j}$ with a random value.

4. If $\mathbf{Q}_{j,i}$ is probed, we need to consider two cases as in the previous step:
   - If all the values $r_{i,k}$ for $1 \leq k \leq d$ are probed, we can perfectly simulate the values $\mathcal{Q}^k_{i,j}$, and hence $\mathbf{Q}_{j,i}$ can be simulated. Note that, λ^0_i is an element of the inverse Vandermonde matrix so it is a public value.
   - If at least $r_{i,k}$ for $1 \leq k \leq d$ is not probed, that means $r_{i,k}$ does not enter into the computation of $\mathbf{Q}_{j,i}$, therefore, $\mathbf{Q}_{j,i}$ can be simulated by a random value.

Now we explain how to simulate output shares $\mathbf{Q}_i$ for all $i \in \mathcal{O}$ where $\mathcal{O}$ is an arbitrary subset of $[1, n]$ with t_2 elements such that $t_1 + t_2 < d + 1$. Clearly, using t_1 probes, we can observe at most t_1 intermediate variables of $\mathbf{Q}_i$, where $\mathbf{Q}_i$ can be written as: $\mathbf{Q}_i = \sum_{j=0}^{n-1} \lambda^0_i \mathcal{Q}_{j,i}$. Since $t_1 + t_2 < d + 1$, at least one intermediate variable of $\mathbf{Q}_i$ is not probed. Therefore, we can simulate $\mathcal{Q}_{j,i}$ with $j \notin U$ by generating a random degree d polynomial and evaluating it for α_i. Hence, we can simulate $\mathbf{Q}_i$ for each $i \in \mathcal{O}$. □

3.4.2 Fault Resistance

First, we discuss the resistance capabilities of the systems using fault detection operations. As given in Section 3.2.1, affine transformation and efficient squaring operations can be listed as fault preserving, while addition and multiplication are fault preserving with high probability. Therefore, fault detection operations should be used before all multiplication and addition operations to ensure perfect fault detection. However, depending on the circuit and the number ε, this number can be decreased.

Therefore, an optimal point between performance and security can be achieved. The advantage of using fault detection operations is that it can be carried out without leaking sensitive information.
Next, we discuss the fault resistance features of the proposed scheme without using the fault detection operation. The fault detection mechanism relies on the degree of the polynomial generated by the secret state. We illustrate the methodology with the following example:

Example 3.1. *Assume* $(4,1)$*-secret sharing* (F_0, F_1, F_2, F_3) *is used to share a secret. Clearly, the secret sharing polynomial* $\deg(F(\mathsf{x})) \leq 1$ *and the following equation holds:*

$$V^{-1} \cdot \underbrace{\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{pmatrix}}_{\textit{Secret states.}} = \underbrace{\begin{pmatrix} x \\ f_1 \\ 0 \\ 0 \end{pmatrix}}_{\textit{The coefficients}}.$$

Assume the fault, denoted by σ*, is injected to the last share. After the injection, the secret sharing polynomial* $F'(\mathsf{x})$ *becomes a polynomial generated by the points* $(F_0, F_1, F_2, F_3 \oplus \sigma)$*. Remark that, as stated by the additive fault model, the relation between polynomials can be seen as* $F'(\mathsf{x}) = F(\mathsf{x}) \oplus \Delta(\mathsf{x})$ *and can be illustrated as follows:*

$$V^{-1} \cdot \begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3' \end{pmatrix} = V^{-1} \cdot \left[\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ \sigma \end{pmatrix} \right] = \begin{pmatrix} x \\ f_1 \\ 0 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{pmatrix}.$$

Error polynomial $\Delta(\mathsf{x})$ *has one nonzero point and three zero points. In other words* $\Delta(\mathsf{x})$ *is generated by the points* $(0,0,0,\sigma)$*. Clearly these points belong to an at least degree-3 polynomial. Hence, we can detect the fault by looking at the degree of the polynomial resulting* $F'(\mathsf{x})$*. Clearly, the only way of generating undetectable faults is arranging* $\Delta(\mathsf{x})$ *as a degree-1 polynomial.*

Using this motivation, we introduce the following lemma which constitutes a basis of our error detection method.

Lemma 3.5. *Let* $(F_i)_{0 \leq i < n} \in \mathbb{F}$ *represent an* (n,d)*-secret sharing of* $f_0 \in \mathbb{F}$ *with* $n = d + \varepsilon + 1$ *and* $\Delta(\mathsf{x})$ *represents the polynomial generated by the faults. If* k *faults are*

injected to secret states, generating an error polynomial degree greater than d is,

$$Pr[\deg(\Delta(x)) > d] = \begin{cases} 1 & k \leq d+\varepsilon \\ 1 - \frac{|\mathbb{F}|^{k-\varepsilon}-1}{|\mathbb{F}|^{k}-1} & k > d+\varepsilon \end{cases}.$$

Proof. Let $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{F}$ be public evaluation points and $F(\mathsf{x}) = x \oplus f_1\mathsf{x} \oplus \ldots \oplus f_d\mathsf{x}^d \in \mathbb{F}[\mathsf{x}]$ be the secret sharing polynomial. Without loss of generality, assume there exist k faults in the first k secret shares and let us denote the corresponding error polynomial by $\Delta(\mathsf{x}) = \delta_0 \oplus \delta_1\mathsf{x} \oplus \ldots \oplus \delta_{d+1}\mathsf{x}^{d+1} \oplus \ldots \oplus \delta_{d+\varepsilon}\mathsf{x}^{d+\varepsilon} \in \mathbb{F}[\mathsf{x}]$. From Equation (3.1), the relation between faulty shares and coefficients of $F(\mathsf{x})$ and $\Delta(\mathsf{x})$ can be seen as follows:

$$V^{-1}\underbrace{\begin{pmatrix} F_0 \oplus \sigma_0 \\ \vdots \\ F_{k-1} \oplus \sigma_{k-1} \\ F_k \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Faulty state}} = V^{-1}\underbrace{\begin{pmatrix} F_0 \\ \vdots \\ F_{k-1} \\ F_k \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Points of } F(x)} \oplus V^{-1}\underbrace{\begin{pmatrix} \sigma_0 \\ \vdots \\ \sigma_{k-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\text{Points of } \Delta(x)} = \begin{pmatrix} x \\ \vdots \\ f_d \\ 0 \\ \vdots \\ 0 \end{pmatrix} \oplus \begin{pmatrix} \delta_0 \\ \vdots \\ \delta_d \\ \delta_{d+1} \\ \vdots \\ \delta_{n-1} \end{pmatrix}$$

where $V = (\alpha_j^i)$ is the $n \times n$ Vandermonde matrix. As seen in the above equation $\Delta(\mathsf{x})$, is the polynomial generated by the points $(\sigma_0, \ldots, \sigma_{k-1}, 0, \ldots, 0)$, i.e. it has $n-k$ zero points and, therefore, it generates at least a degree $n-k$ polynomial.

Assume that $k \leq d+\varepsilon$. Since $n = d+\varepsilon+1$, it is clear that $n-k \geq d+1$. Therefore, $\deg(\Delta(\mathsf{x})) > d$ with probability 1 and faults are always detectable.

Assume $k > d+\varepsilon$. Then we need to focus on the Vandermonde representation of the secret shares. Using $V^{-1}_{i,j} = \lambda_j^i$, we can form a system of linear equations for error detection terms of $\Delta(x)$:

$$\begin{aligned} \sigma_0\lambda_0^{n-1} \oplus \ldots \oplus \sigma_{k-1}\lambda_{k-1}^{n-1} &= \delta_{n-1} \\ \sigma_0\lambda_0^{n-2} \oplus \ldots \oplus \sigma_{k-1}\lambda_{k-1}^{n-2} &= \delta_{n-2} \\ &\vdots \\ \sigma_0\lambda_0^{n-\varepsilon} \oplus \ldots \oplus \sigma_{k-1}\lambda_{k-1}^{n-\varepsilon} &= \delta_{d-\varepsilon} \end{aligned}$$

To arrange $\Delta(x)$ as degree d polynomial, the above equations should be solved for $\delta_i = 0$ for $i = (d+1, d+2, \ldots, d+\varepsilon)$. The parameters of this homogeneous system of linear equations are k unknowns and ε equations. Since $k > \varepsilon$, the number of non-trivial

solutions for this system is $|\mathbb{F}|^{k-\varepsilon} - 1$. Therefore, $Pr[\deg(\Delta(x)) \leq d] = \frac{|\mathbb{F}|^{k-\varepsilon}-1}{|\mathbb{F}|^{k}-1}$. Hence,

$$Pr[\deg(\Delta(x)) > d] = 1 - \frac{|\mathbb{F}|^{k-\varepsilon} - 1}{|\mathbb{F}|^{k} - 1}.$$

which proves the lemma. □

Next, we can state the following theorem, to clarify our fault detection properties. Remark that the following theorem was already proven in [YO13, Sec. 4.1].

Theorem 3.6. *Let $F'(\mathsf{x})$ be the faulty secret sharing polynomial. If $\deg(F'(\mathsf{x})) > d$, the faults can be detectable.*

Using this theorem and the notation given by Schneider et al. [SMG16], we can define the fault coverage of our scheme. Let $F'(x)$ be the faulty secret sharing polynomial, then the probability of a set of faults to be undetectable is defined as our fault coverage:

$$Coverage_{\varepsilon} = 1 - Pr[\deg(F'(\mathsf{x})) \leq d].$$

Assume the number of injected faults to the system is k, in the first multiplication, faults are propagated with probability 1 if $k \leq \varepsilon$ as given in Lemma 3.5. However, we cannot use Lemma 3.5 as the fault coverage for a set of operations, since faults can be injected in different instances or one fault can spread to a large number of shares. As a result, faults become unstable and potentially undetectable. In a sequence of operations, faults can become undetectable after an SMC addition or multiplication. In the following, we perform the security analysis and derive the probability of undetectable faults in SMC multiplication.

Corollary 3.7 (*Propagation*$_{\varepsilon}$(EPMult)). *Let $(F_i)_{0\leq i<n} \in \mathbb{F}$ and $(G_i)_{0\leq i<n} \in \mathbb{F}$ be the input shares of the Error Preserving Multiplication operation and let $(\boldsymbol{Q}_i)_{0\leq i<n}$ be the output shares. And let us denote the sets of faulty indices as k_F and k_G respectively, with $k = |k_F \cup k_G|$. If the fault is detectable using both of the set of input shares $(F_i)_{0\leq i<n}$ and $(G_i)_{0\leq i<n}$, the faults can be detectable using output shares $(\boldsymbol{Q}_i)_{0\leq i<n}$ with the following probability:*

$$Propagation_{\varepsilon}(\texttt{EPMult}) = \begin{cases} 1 - \frac{1}{q^{d+k}} & k \leq \varepsilon \\ 1 - \frac{1}{q^{d}}\left(\frac{1}{(q+1)^{e}} + \frac{1}{q^{k}}\right) & k > \varepsilon \end{cases}, \text{ where } |\mathbb{F}^*| \text{ is denoted by } q.$$

Proof. Assume there exists 2 sets of faults within the input shares $(F_i)_{0\leq i<n}$ and $(G_i)_{0\leq i<n}$, such that if i^{th} share F_i (resp. G_i) is faulty, then $\sigma_{F_i} \neq 0$ (resp. $\sigma_{G_i} \neq 0$) otherwise $\sigma_{F_i} = 0$ (resp. $\sigma_{G_i} = 0$). And clearly, $k_F = \{i|\sigma_{F_i} \neq 0\}$ and $k_G = \{i|\sigma_{G_i} \neq 0\}$. The polynomial $H(x)$ is generated by the shares $F_i \cdot G_i$ for $0 \leq i < n$ and the faults in H_i can be calculated as follows:

$$H_i = [F_i \oplus \sigma_{i_F}] \cdot [G_i \oplus \sigma_{i_G}] = F_i G_i \oplus \underbrace{F_i\sigma_{i_G} \oplus G_i\sigma_{i_F} \oplus \sigma_{i_F}\sigma_{i_G}}_{\text{The fault in } H_i \text{ denoted by } \sigma_{H_i}.}$$

Remark that $\mathsf{E}_{i,j}$ is used to propagate the faults. The faults in $(\mathbf{Q}_i)_{0\leq i<n}$ become undetectable if and only if the following equations, which correspond to the partial sums in step 3, hold:

$$\sum_{j=0}^{n-1} \lambda_i^0 \mathsf{E}_{i,j} = 0 \text{ for } 0 \leq i < \varepsilon + d. \tag{3.4}$$

From the definition of $\mathsf{E}_{i,j}$ we know that the variables correspond to shares error detection terms of $F(\mathsf{x}) \oplus G(\mathsf{x})$ or $H(\mathsf{x})$. Hence, faults become undetectable if and only if error detection terms are zero, i.e. the following equations hold:

1. $f_{d+1} \oplus g_{d+1} = \ldots = f_{2d} \oplus g_{2d} = 0$. Remark that we assumed the fault is detectable using both of the sets of input shares $(F_i)_{0\leq i<n}$ and $(G_i)_{0\leq i<n}$, therefore, at least one $f_i \neq 0$ and $g_i \neq 0$ where $i \in \{d+1, \ldots, 2d\}$. Therefore,

$$Pr[(f_i \oplus g_i = 0)_{d<i\leq 2d}] = \frac{1}{q}\left(\frac{1}{q+1}\right)^{d-1} \approx \frac{1}{q^d}.$$

2. $h_{2d+1} = \ldots = h_{2d+\varepsilon+1} = 0$. In other words $\deg(H(\mathsf{x}) \leq 2d)$. And $\deg(H(\mathsf{x}) \leq 2d)$ if and only if

   a) The polynomial generated by σ_{H_i}'s is at most a degree $2d$ polynomial or

   b) $\sigma_{H_i} = 0$ for $i = 0, \ldots, n-1$.

   From Lemma 3.5, the probability of condition (a) is:

$$Pr[\deg(H(x)) \leq 2d] = \begin{cases} 0 & k \leq \varepsilon \\ \frac{(q+1)^{k-\varepsilon}-1}{(q+1)^k-1} \approx \frac{1}{(q+1)^\varepsilon} & k > \varepsilon \end{cases}.$$

   And the probability of condition (b) is:

$$Pr[(\sigma_{H_i} = 0)_{0\leq i<n}] \approx \frac{1}{q^k}.$$

Using the conditions listed above we can calculate the probability of the Equation (3.4) to be hold is:

$$Pr\left[(\sum_{j=0}^{n-1}\lambda_i^0\mathsf{E}_{i,j}=0)_{0\leq i<\varepsilon+d}\right]=\begin{cases}\frac{1}{q^{d+k}} & k\leq\varepsilon\\ \frac{1}{q^d}\left(\frac{1}{(q+1)^\varepsilon}+\frac{1}{q^k}\right) & k>\varepsilon\end{cases}.$$

Therefore,

$$Propagation_\varepsilon(\texttt{EPMult})=\begin{cases}1-\frac{1}{q^{d+k}} & k\leq\varepsilon\\ 1-\frac{1}{q^d}\left(\frac{1}{(q+1)^\varepsilon}+\frac{1}{q^k}\right) & k>\varepsilon\end{cases}.$$

□

Using propagation probabilities of individual operations, we can analyze the fault resistance of a composition of operations. The following theorem formally analyzes the fault resistance properties of a combination of operations. The main idea of the theorem is to examine the propagation of fault indices and calculate the individual $Propagation_\varepsilon$.

Theorem 3.8. *Let $A_1,\ldots,A_t$ be a sequence of operations and let us denote the sets of faulty indices of A_j as k_{F^j} and k_{G^j} respectively, with $k_j=|k_{F^j}\cup k_{G^j}|$. Without loss of generality let's assume that the fault is detectable using the inputs of A_i where $1\leq i\leq t$. Then, $Propogation_\varepsilon$ can be calculated as follows:*

$$Propogation_\varepsilon(A_1,\ldots,A_t)=\prod_{j=i}^{t}Propogation_\varepsilon^{k_j}(A_j).$$

Proof. First, let us categorize operations into to two sets depending on the number of inputs as follows: $\mathcal{A}_1=\{\texttt{Affine},\texttt{Sqr},\texttt{RefreshM},\texttt{FDect}\}$[3] and $\mathcal{A}_2=\{\texttt{EPMult},\texttt{Add}\}$. Since a fault is detectable using the inputs of A_i, we know that the degree of the secret sharing polynomial is greater than d and $k_i>0$ from Theorem 3.6. First, we analyze propagation of the number of faulty indices in three cases:

- **Case 1:** $k_{i+1}=k_i$ if $A_{i+1}\in\mathcal{A}_1$ and $Propagation_\varepsilon(A\in\mathcal{A}_1)=1$.
  - $A_{i+1}\in\{\texttt{Affine},\texttt{Sqr}\}$, then the magnitude of faults is changed however the number of faulty indices is preserved, as explained in Section 3.2.1.
  - $A_{i+1}\in\{\texttt{RefreshM},\texttt{FDect}\}$. Since `RefreshM` and `FDect` can be seen as additions with a valid (i.e degree-d) secret sharing, the number of faulty indices are preserved with their magnitudes.

[3] We excluded the `ReComb` operation from the analysis, since it can only be the last operation. The case where a fault is detectable using the inputs of `ReComb` is already explained in Section 3.3.3.

- **Case 2:** $0 \leq k_{i+1} \leq k_i$ if $A_{i+1} = \texttt{Add}$, The number of fault indices can be decreased depending on the magnitudes and the indices of the faults. As explained in Section 3.2.1 $Propagation_\varepsilon(\texttt{Add}) = (1/q)^{d+\varepsilon}$.
- **Case 3:** $0 \leq k_{i+1} \leq \varepsilon + d$ if $A_{i+1} = \texttt{EPMult}$, then the number of faulty indices changes depending of the Equation (3.4) in Corollary 3.7.

Using these discussions we analyzed the propagation of the number of faulty indices of each operation. Depending on the operation and number of fault indices we can calculate $Propogation_\varepsilon$ of operations individually. Hence, the following equation holds:

$$Propogation_\varepsilon(A_1, \ldots, A_t) = \prod_{j=i}^{t} Propogation_\varepsilon^{k_j}(A_j). \tag{3.5}$$

□

Remark 3.5. *In Theorem 3.8 we assumed that faults are injected before the i^{th} operation. The attacker can inject additional faults into the scheme, which can change the number of faulty indices. However, the propagation of faulty indices of individual operations works as analyzed in Theorem 3.8 and the propagation of faults can be calculated using the Equation (3.5) for a composition of individual operations in the presence of faults.*

The infective computation property of our scheme is based on the $Propogation_\varepsilon$. As given in Section 3.3.3, the infective property of our scheme is provided by $\mathsf{E}^R_{i,j}$. If a fault is detectable using input shares $(F_i)_{0\leq i<n}$, then $f_i \leq 0$ for at least one $i \in \{d{+}1, \ldots, n{-}1\}$ from Theorem 3.6. As a result, the output shares are randomized by at least one nonzero coefficient and a random value. Therefore, the infective computation is achieved.
The analyzed fault model considers faults on intermediate states only. However, it has been shown that faulting the control flow, e.g. the number of rounds of a cipher, is also sufficient for key recovery [CT05, DMN+12]. We consider such attacks out of the scope of this work. Indeed, such attacks might not be a concern for fully unrolled circuits, but other implementation styles would require additional protection of the control flow to prevent such attacks.

3.4.3 Resistance Against Combined Attacks

After describing the side-channel and fault resistance of our scheme, a natural question arises: what will be the security properties if an attacker is able to mount SCA and FA together? In this section we focus on two attacks described in [CFGR10]. Due to

Table 3.1: Number of operations in Gennaro et al. [GRR98] and `EPMult` in Section 3.3.1, where Mul., Add. and Rand. represents the field multiplication, field addition, and randomness requirements respectively.

	Gennaro et al. [GRR98]			`EPMult`			Overhead
	step 1	step 2	step 3	step 1	step 2	step 3	
Mul.	n	n^2d	n^2	n	$n^2d + n(\varepsilon + d)$	n^2	$n(\varepsilon + d)$
Add.	-	n^2d	$(n-1)n$	-	$n^2d + n(\varepsilon + 2d)$	$(n-1)n$	$n(\varepsilon + 2d)$
Rand.	-	nd	-	-	nd	-	-

the infective properties of our scheme, the attacker will not be able to collect useful faulty ciphertexts. Secondly, even if the attacker successfully chooses ε faults in such a way that the shared values are fixed to a predefined value, the attacker should probe $d + 1$ variables to recover the secret, which is not possible in our model. Therefore, the combined attacks as defined in [CFGR10] are naturally eliminated by the scheme. Another advantage of the scheme is that our fault model is defined as *blindly-chosen* and *non-adaptive*. Therefore, the attacker cannot observe the secret states and forge a fault to inject. The model inherently creates a timing limitation which eliminates *rushing adversary* [RMB+17]. As a result, *Propagation* probabilities are not affected by probing d variables. Moreover, fault injections targeting randomness sources to disable masking could be a serious threat to the system. In our model, the faults on randomness would change the randomness in a way the attacker cannot control, thereby keeping the side-channel protection intact.

3.4.4 Performance Analysis

Next we analyze the performance of our scheme in terms of basic operations such as field multiplications, field additions and randomness requirements, and compare the perormance to related work. Table 3.1 compares the SMC multiplication of Gennaro et al. [GRR98] to the `EPMult` defined in Section 3.3.1 in terms of field additions (XOR), multiplications, and required fresh randomness. As shown in Table 3.1, performance overhead is only introduced in the second step, while calculating the $\mathsf{E}_{i,j}$. The additional costs of adding $\mathsf{E}_{i,j}$ are $n(\varepsilon+d)$ field multiplications and $n(\varepsilon+2d)$ field additions. Except this overhead, both schemes have identical cost: each player generates a random degree-d polynomial and sends the corresponding values to the other players, requiring nd random values and n^2 polynomial evaluations, where each evaluation costs d field multiplications and d additions.

An overview of the computational cost for the SMC operations described in Section 3.2.1

Table 3.2: Number of field multiplications, additions, and randomness requirements for the SMC operations.

	`EPMult`	$\texttt{Sqr}_k$	`Add`	`Affine`	`RefreshM`
Field Mul.	$n^2(d+1)+n(\varepsilon+d+1)$	nk	-	n	nd
Field Add	$n^2(d+1)+n(\varepsilon+2d-1)$	-	n	n	nd
Randomness	nd	-	-	-	d

Table 3.3: Number of field multiplications, additions, and randomness requirements for the Recombination Operation and Fault Detection Operation

	Recombination		Fault Detection	
	Re-Sharing	Reconstruction	Randomization	Detection
Mul.	$n^2(d+1)+n(2\varepsilon+2d+1)$	$n(\varepsilon+d+1)$	$n(d+1)$	$n(\varepsilon+d)$
Add.	$n^2(d+1)+n(\varepsilon+d-1)$	$(n-1)(\varepsilon+d+1)$	$n(d+2)$	$(n-1)(\varepsilon+d)$
Rand.	$\varepsilon+d+nd$	-	$d+1$	-

and in Section 3.3.1 is provided in Table 3.2. The table lists the required number of field multiplications, additions, and the randomness requirements for every secure operation of an *(n,d)* masking scheme. Besides the arithmetic operations, the scheme requires the recombination and fault detection operations. The costs for both are listed in Table 3.3. A single recombination is more costly than an error preserving multiplication; however, only one recombination operation per secret value is needed, keeping the contribution to the overall cost small. The total overhead of the fault detection operation depends on the sequence of operations and security level of the implementation. In the first step, $(d+1)$ random values are needed to generate a random polynomial. The evaluation of the polynomial for all players costs n^2d field multiplications and n^2d field additions. Thus, the cost is small compared to the multiplication, allowing for frequent checks of the state.

We compare our results with other side-channel countermeasures and one other combined side-channel fault countermeasure in Table 3.2. We consider the total number of field multiplications and additions of the side-channel protected scheme by Roche and Prouff [RP11], the scheme by Rivain and Prouff [RP10] and combined side-channel fault countermeasure (CAPA) by Reparaz et al. [RMB+17]. Remark that, our work and [RP11] operate on polynomial masking while [RMB+17] and [RP10] use Boolean masking, hence $d = n-1$. The only comparable numbers corresponds to a combined countermeasure are given in Table 1 in [RMB+17]. For CAPA, the number of operations shown in Table 3.2 include the computations required for public values, output calculations and MAC check functionalities [RMB+17]. While some

Table 3.4: Performance Comparison of Secure Operations in terms o field operations; field additions are shown in normal font while the number Field Multiplications are in bold font.

	SMC Multiplication	SMC Square	SMC Addition
Our Work	$n^2(d+1)+n(\varepsilon+d+1)$	n	-
	$\mathbf{n^2(d+1)+n(\varepsilon+2d-1)}$	**-**	$\boldsymbol{n}$
Roche-Prouff [RP11]	$n^2(d+1)+n$	n	-
	$\mathbf{n^2(d+1)-n}$	**-**	$\boldsymbol{n}$
CAPA [RMB+17]	$8n$	$2n$	-
	$\mathbf{11n+4n(n-1)+1}$	$\mathbf{2n^2+3n+1}$	$\mathbf{2n}$
Rivain-Prouff [RP10]	n^2	n	-
	$\mathbf{2n(n-1)}$	**-**	$\boldsymbol{n}$

operations like addition and square transformation cost more in [RMB+17], the secure multiplication is the bottleneck of our scheme when applied for higher order masking.

3.5 Side-Channel and Fault Resistant AES Implementation

The AES block cipher consists of multiple rounds of operations on its state. The iterations include three linear layers: `MixColumns`, `ShiftRows`, and `AddRoundKey` and one non-linear layer named `SubBytes`. In order to protect these functions from leaking information about the data they are processing, they must be designed to work on shares of the secret variables. These operations are known as *secure* or *SMC* addition(`Add`), multiplication(`EPMult`), squaring($\texttt{Sqr}_k$), and affine transformation(`Affine`); furthermore, the secure operations are composed of simpler field addition, multiplication, and squaring. The details of the simpler operations can be found in the appendices. Also, even though it is not an operation itself, a reliable source of randomness is fundamental. Thus, our implementation is built bottom-up, the field operations and randomness represent the building blocks, and more complex functions are layered on top of them

3.5.1 SMC Operations

The linear layers can be implemented in a straightforward manner with computations done locally. The MPC implementation of `SubBytes` consists of squarings and multiplications [RP12]. As explained in Section 3.2.1, faults injected during this part remain undetected in the previous scheme [RP12], which makes the `SubBytes` vulnerable. The `SubBytes` layer consists of two main stages.

- The power function $x \rightarrow x^{254}$ over GF(2^8), denoted by $Exp254(x)$, can be

Algorithm 6 $Exp254((F_i)_{0\leq i<n})$

Input: Shares of f_0 as $(F_i)_{0\leq i<n}$.
Output: Shares of f_0^{254} as $(Y_i)_{0\leq i<n}$.
1: $(Z_i)_{0\leq i<n} = \texttt{Sqr}_1((F_i)_{0\leq i<n})$ // $z \leftarrow f_0^2$
2: $(Z_i)_{0\leq i<n} = \texttt{RefreshM}((Z_i)_{0\leq i<n})$
3: $(Y_i)_{0\leq i<n} = \texttt{EPMult}((Z_i)_{0\leq i<n}, (F_i)_{0\leq i<n})$ // $y \leftarrow f_0^3$
4: $(W_i)_{0\leq i<n} = \texttt{Sqr}_2((Y_i)_{0\leq i<n})$ // $w \leftarrow f_0^{12}$
5: $(W_i)_{0\leq i<n} = \texttt{RefreshM}((W_i)_{0\leq i<n})$
6: $(Y_i)_{0\leq i<n} = \texttt{EPMult}((Y_i)_{0\leq i<n}, (W_i)_{0\leq i<n})$ // $y \leftarrow f_0^{15}$
7: $(Y_i)_{0\leq i<n} = \texttt{Sqr}_4((Y_i)_{0\leq i<n})$ // $y \leftarrow f_0^{240}$
8: $(Y_i)_{0\leq i<n} = \texttt{EPMult}((Y_i)_{0\leq i<n}, (W_i)_{0\leq i<n})$ // $y \leftarrow f_0^{252}$
9: $(Y_i)_{0\leq i<n} = \texttt{EPMult}((Y_i)_{0\leq i<n}, (Z_i)_{0\leq i<n})$ // $y \leftarrow f_0^{254}$
10: **return** $(Y_0, \ldots, Y_{n-1})$

calculated using the Algorithm 6. Using Theorem 7.2 and Theorem 3.2, we can prove the t-SNI$_d^n$ security of the $Exp254(x)$ operation, as already proven in [BBD+16].

Theorem 3.9 (t-SNI$_d^n$ of *Exp254*). *Let $(F_i)_{0\leq i<n}$ be the input shares of $Exp254$, and let $(Y_i)_{0\leq i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 \leq d$, there exist two subsets I and J of indices with $|I| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $Y_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$.*

- The second part of the `SubBytes` operation is the GF(2)-affine transformation and it is denoted by $\tau(y)$ [RP12]:

$$\begin{aligned}\tau_A(y) = 0x63 \quad &\oplus \quad (0x05 \cdot y) \oplus (0x09 \cdot y^2) \oplus (0xf9 \cdot y^4) \oplus (0x25 \cdot y^8) \\ &\oplus \quad (0xf4 \cdot y^{16}) \oplus (0x01 \cdot y^{32}) \oplus (0xb5 \cdot y^{64}) \oplus (0x8f \cdot y^{128}).\end{aligned}$$

Using the `EPMult`, we are able to compute the output of `SubBytes` securely while the probability of generating undetectable faults is 2^{-12} in the worst case for a $(4,1)$-MPC where all the shares are faulty. To further break down the SMC design into its fundamental components, Table 3.5 shows the total number of SMC operations in one round of AES-128.
Based on these results, we provide the performance analysis and cost of different (n,d)-SMC schemes. The analysis is performed by using the total number of field multiplications, additions, and randomness requirements for one round of AES-128 as seen in Table 3.5 and Table 3.2. Results are shown in Table 3.6.

Table 3.5: The number of SMC operations in one round of AES.

	$Exp254$	$\tau(y)$	`MixColumns`	`AddRoundKey`	`ShiftRows`
`EPMult`	16×4	-	-	-	-
$\texttt{Sqr}_1$	16×7	16×7	-	-	-
`Add`	-	16×7	12	16×1	-
`Affine`	-	16×8	16	-	-
`RefreshM`	16×2	-	-	-	-

Table 3.6: Total number of operations for different (n, d)-scenarios for one round of AES-128.

	$\varepsilon = 0$		$\varepsilon = 1$		$\varepsilon = 2$	$\varepsilon = 3$
	(3,1) [GRR98]	(3,1)	(4,1)	(6,2)	(5,1)	(6,1)
Field Mul.	2448	2640	4288	10656	6320	8736
Field Add	1428	2196	3696	10152	5580	7848
Randomness	192	192	256	768	320	384

Next, we analyze first-order side-channel resistant AES-128 implementations. Using (4,1)-SMC, we are able to extend the first-order side-channel implementation of Roche and Prouff [RP12] to a combined first-order side-channel and fault resistant implementation. The extension increases the number of field multiplications by 62%, additions by 68%, and randomness requirements by 33%. Since the error detection coefficients are used for error propagation, our scheme is more efficient than simple duplication. Moreover, we can increase the side-channel resistance of the system to second order by using (6,2)-SMC. The cost of this implementation requires 148% more field multiplications and also 164% more additions, since it heavily depends on n and ε. On the other hand, the randomness requirement increases by 200%, because the cost of it is proportional to n and d. Also, as Table 3.6 shows, (4,1), (5,1), and (6,1)-SMCs have the same side-channel resistance and have the first, second, and third order fault resistance, respectively. The number of field multiplications and additions is nearly proportional to half of the fault resistance order. Therefore, we can conclude that increasing the order of fault resistance costs less than the increase of the side-channel resistance.

3.5.2 Software Implementation

Up to this point, we have only discussed the theoretical performance results; the next section describes the performance results in terms of execution time of the whole encryption and its building blocks om ARM Cortex M0+ (the details of our setup can

Table 3.7: AES-128 encryption execution time, code and RW-data size depending on the GF(2^8) operations variations. I:Instruction Only, M:Mixed, L:LUT, E:Exp-Log.

	(3,1)			(5,2)			unmasked
GF(2^8) mult.	I	M	E	I	M	E	-
GF(2^8) sqr.	I	L	L	I	L	L	-
Encryption [GRR98]	1.45M	1.11M	0.52M	8.04M	6.48M	2.90M	-
Our scheme	1.75M	1.37M	0.64M	9.21M	7.47M	3.4M	-
Code Size (kB)	3.4	3.3	3.3	3.6	3.4	3.5	7.2
RW-data (B)	12	524	780	32	544	800	12
RO-data (B)	224	224	224	224	224	224	870

Table 3.8: Execution time for GF(2^8) and SMC operations in μs with the CPU running at 4 MHz. I:Instruction Only, M:Mixed, E: Exp-Log

	(3,1)			(4,1)		(5,1)		(5,2)			(6,2)	
	I	M	E	I	M	I	M	I	M	E	I	M
GF(2^8) mult.	54.5	44.5	17.5	54.5	44.5	54.5	44.5	54.5	44.5	17.5	54.5	44.5
GF(2^8) sqr.	13.8	1.5	1.5	13.8	1.5	13.8	1.5	13.8	1.5	1.5	13.8	1.5
`getrn()`	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8
SMC add.	15.2	15.2	15.2	18.2	18.2	21.2	21.2	21.2	21.2	21.2	24.2	24.2
Mult [GRR98]	1.2k	1.0k	0.48k	2.1k	1.7k	3.2k	2.7k	9.1k	7.5k	3.4k	13k	10.8k
`EPMult`	1.4k	1.1k	0.54k	2.4k	2k	3.8k	3.2k	9.8k	8.1k	3.7k	13.5k	11.2k

be found in Chapter B). The overall encryption execution timings for the (3,1) and (5,2) schemes are shown in Table 3.7. As a reference, we consider the 32-bit C implementation of AES in OpenSSL 1.0.1g, compiled for the ARM Cortex-M0+ and run at core clock frequency of 4 MHz. The execution time for this unmasked encryption is 481.5 μs. Table 3.7 shows its corresponding code and data size. Even though full unrolling is disabled, the code and data size is significantly larger, however, the execution time is 1090X faster than the fastest masked encryption in Table 3.7.

SMC multiplication is the bottleneck of the algorithm and in turn it relies on the field multiplication. The execution time can also be reduced by running at higher frequencies but the performance would remain the same, however, power consumption would increase. Table 3.7 details the amount of code and RW-data according to selected combinations of field operations, noted that other combinations are also possible to produce different code and data sizes.

Table 3.8 summarizes the execution timings corresponding to the different versions of field operations and the SMC multiplication. Based on Table 3.5, these building block operations represent the key elements to boost the performance of the masking scheme, that is the reason to look for faster methods to perform field arithmetic.

Performance Analysis. The only comparable implementation was presented in [GSF14] and features, according to its Figure 2, an approximate number is 4.5 million cycles for a (5,2) scheme. Our fastest second order implementation takes 11.6 million cycles which is nearly 2.6X slower. The comparison is based on the graphs in Figure 2 of [GSF14]. We surmise that part of the performance degradation is due to different platform features and the fact that our implementation is of constant time and performs on-the-fly mask generation. It also suggests that significant performance gains can be achieved through further optimizations of our proof-of-concept implementation.

Hardware Implementation. The proposed scheme is also well-suited for hardware implementation, due to its glitch-resistance. A proof-of-concept implementation of the Roche and Prouff scheme was analyzed by Moradi and Mischke [MM13]. Their reference implementation introduces a rather high overhead, in area but also in lost performance. Our scheme will increase this overhead due to the fault resistance, as quantified in Section 3.4.4, mainly due to the increased number of shares. It should be noted that the reference implementation in [MM13] has parallel hardware, but still performs serialized processing of all shares. However, parallel processing of shares can be secure [BDF+17] and would provide a significant performance boost over the fully serialized implementation in [MM13].

3.5.3 Side-channel Analysis

Using the software traces we demonstrate a simple leakage detection test by the test vector leakage assessment (TVLA) by Goodwill et al. [GGJR+11] as described in Section 2.1

To demonstrate the effectiveness of our implementation, an initial t-test was performed with a switched-off masking on a small set of samples and later with the masking enabled on a much larger set of measurements. To disable the masking scheme, during the sharing of the input operands, the highest-degree coefficient was hardcoded to `0x1`.

Fig. 3.2 shows the intense leakage spread around three different points in time. They correspond to the initial three field multiplications that are done within the SMC multiplication. The peaks are generated because there is an immediate relationship between the operands and their corresponding shares. The shares for the fixed operands are always the same and thus consume an approximately equal amount of power on every execution so, when compared to the power consumption of random operands, a huge difference is revealed after just 12,000 traces.

After demonstrating the effectiveness of the t-test, results corresponding to (3,1)-`EPMult`

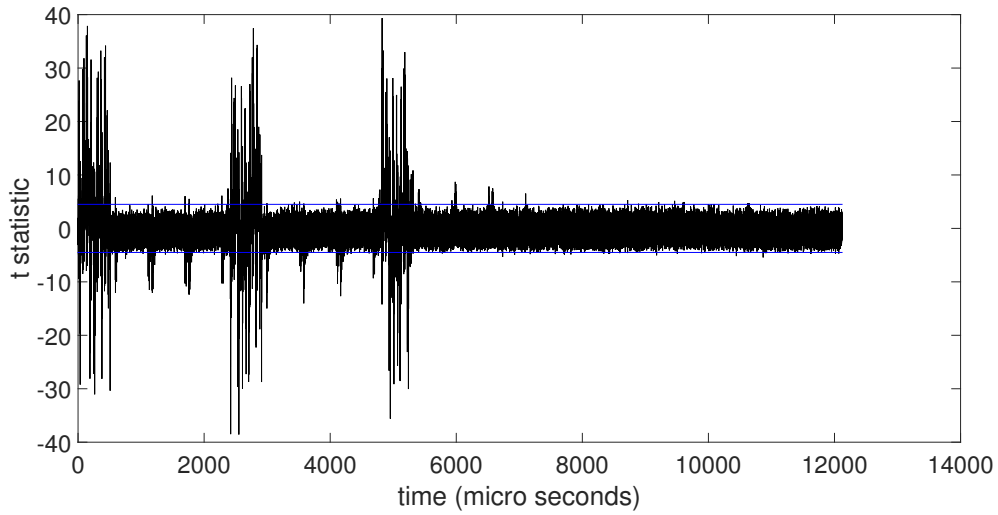


Figure 3.2: Leakage analysis with disabled masking after 12,000 traces.

and (5,2)-`EPMult` are displayed in Fig. 3.3. While (3,1)-`EPMult` executed with a 4 MHz clock, (5,2)-`EPMult`'s clock was switched to 16 MHz, due to its execution length with 4 MHz clock would turn the trace collection impractical.

For all subplots in Fig. 3.3 are showing analysis results for 1st through 5th order. As the figure shows, the level of leakage is contained within the acceptable boundaries. Also, Fig. 3.4 shows the t growth over the number of samples for the first-order t-test.

Multivariate t Test. SMC hardware implementations process their shares in parallel, therefore, the power consumption reflects the processing demand of all of them simultaneously. In our single-threaded software implementation, the operations on every share or pair of shares are performed sequentially. As a result, the power consumption at certain intervals may only be related to a single share or pair of shares being processed [SM15]. The multivariate t-test combines a sample from a particular point in time to other samples at different intervals of time. The objective is to identify if there is a relationship between the processing of the sets of shares that occur at different points in time.

Fig. 3.5 shows the results of the multivariate analysis on relevant sections of the (3,1)-`EPMult`. Each of the plots belongs to the combination of the points in the section where the first pair of shares is processed and those of the sections where the remaining pairs are processed. Although the Exp-Log field multiplication uses table look-ups, the result in Fig. 3.5 does not show any evidence of leakage derived from the memory accesses.

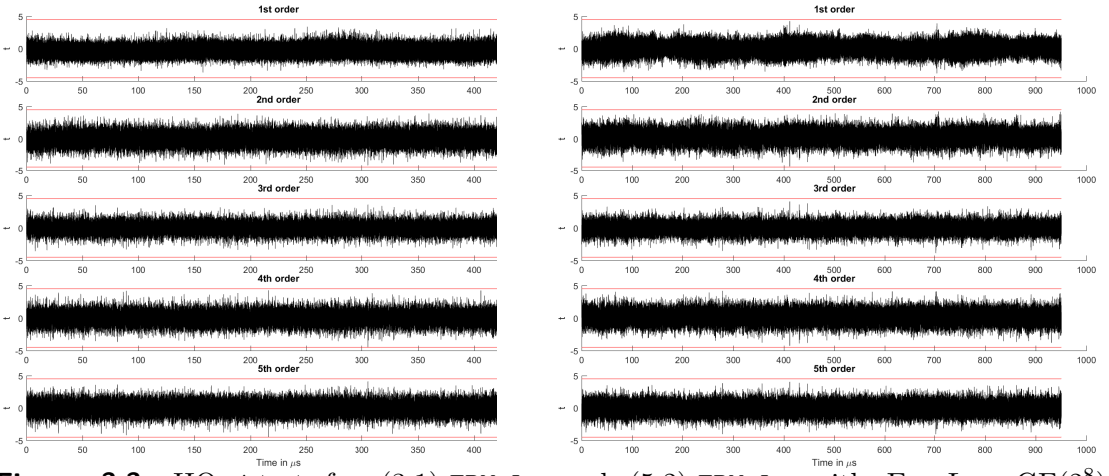


Figure 3.3: HO t-test for (3,1)-`EPMult` and (5,2)-`EPMult` with Exp-Log GF(2^8) multiplication using 250.000 traces.

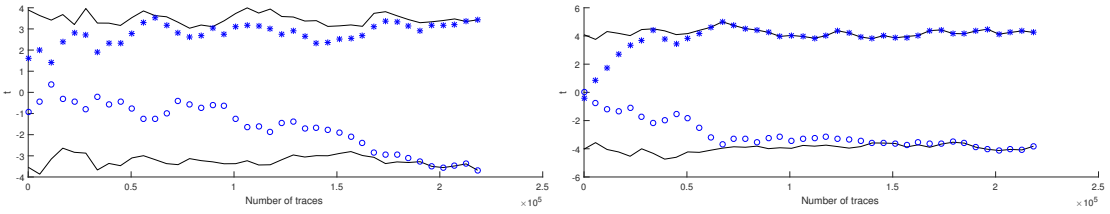


Figure 3.4: First order t growth for (3,1)-`EPMult` and (5,2)-`EPMult` with Exp-Log GF(2^8) multiplication. The black lines show the evolution of the maximum and minimum first-order t values over the number of traces. The stars mark how the index of the last maximum value grew over the number of traces. The circles mark the last minimum values.

The multivariate analysis is a useful tool to reveal potential sources of interdependent side-channel leakage that otherwise would be hidden from the regular t-test. However, the time execution and memory constraints are significant factors to constrain the extension of the analysis to certain sections. Remark that for all of the multivariate analysis subplots, the horizontal axis does not represent time since the analysis itself requires the combination of traces at different points.

3.5.4 Fault Analysis

Next, we present experimental results of fault injection on the proposed scheme on the simulation in `SAGE`. As given in Section 3.4, faults can be undetectable in a sequence of operations. As the only non-linear operation of AES, we focused on `SubBytes` operation. We start with the $Exp254$ operation to our analyses. First, we do the theoretical analyses

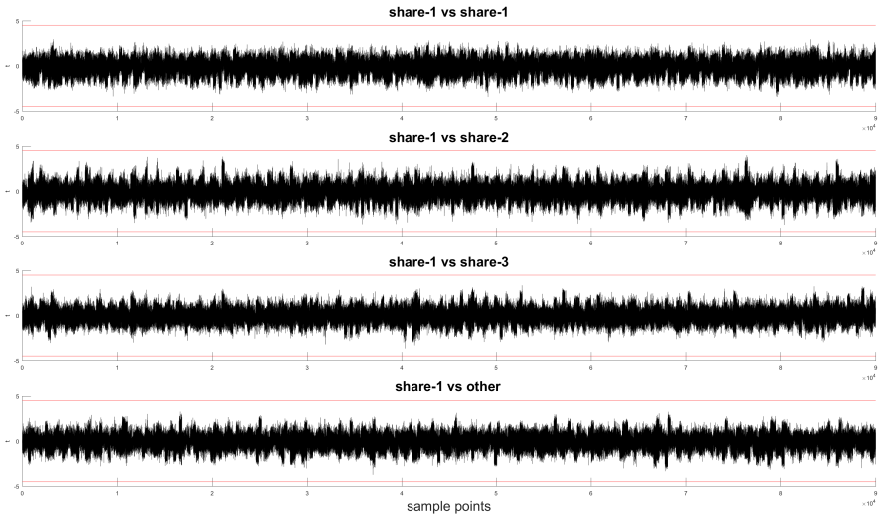


Figure 3.5: Multivariate t-test for sections of the (3,1)-`EPMult` based on Exp-Log GF(2^8) multiplication. Share-1 vs Share-1 shows the multivariate t-test results of all combinations of points during the first GF(2^8) multiplication. Share-1 vs Share-2 shows the results of the multivariate t-test for the combination of points from the first field multiplication to the second one. Share-1 vs Share-3 corresponds to the multivariate t-test analog to the previous case. Share-1 vs Other shows the results of multivariate t-test of the points during the first field multiplication combined with all the points of a section close to the end of `EPMult`.

on `EPMult` and $Exp254$. In Table 3.9 and in Table 3.10, one can see the probabilities of generating undetectable faults for `EPMult` and $Exp254$, respectively. For these analyses, we randomly select the instances withing the gadgets and add a random fault to these instances. For `EPmult`, the faults are detectable using $(F_i)_{(0\leq i<n)}$ and $(G_i)_{(0\leq i<n)}$ in Algorithm 2. And for $Exp254$, the faults are detectable using $(F_i)_{(0\leq i<n)}$ in Algorithm 6. Note that, k is defined as the number of faulty shares as in Corollary 3.7. As seen both tables, the probabilities slightly changes, depending on the conditions listed in Corollary 3.7. Also, even if we used a sequence of operations, the generating undetectable faults for $Exp254$ mostly depend on the last multiplication (line 9 in Algorithm 6). Notice that, the faults in the initial input shares spread to first $\varepsilon + d$ shares of both inputs of `EPMult` operations within $Exp254$. Therefore, the number of faulty shares after the first `EPMult` is calculated as $max(n, 2(\varepsilon + d))$.

In the second part, we verify the theoretical analyses with the experimental results. We look the fault detection capabilities of $Exp254$ and $\tau_a \circ Exp254$. The experimental setup

Table 3.9: Probabilities of generating undetectable faults for `EPMult`.

	$k=1$	$k=2$	$k=3$	$k=4$
(4,1)	1.54×10^{-5}	1.54×10^{-5}	1.53×10^{-5}	1.53×10^{-5}
(5,1)	1.54×10^{-5}	6.03×10^{-8}	6.01×10^{-8}	5.98×10^{-8}
(6,1)	1.54×10^{-5}	6.03×10^{-8}	2.37×10^{-10}	2.35×10^{-10}
(6,2)	6.03×10^{-8}	6.03×10^{-8}	6.01×10^{-8}	6.01×10^{-8}

Table 3.10: Probabilities of generating undetectable faults for $Exp254$.

	$k=1$	$k=2$	$k=3$	$k=4$
(4,1)	1.53×10^{-5}	1.53×10^{-5}	1.53×10^{-5}	1.53×10^{-5}
(5,1)	5.98×10^{-8}	5.98×10^{-8}	5.98×10^{-8}	5.98×10^{-8}
(6,1)	2.34×10^{-10}	2.34×10^{-10}	2.34×10^{-10}	2.34×10^{-10}
(6,2)	6.03×10^{-8}	6.01×10^{-8}	6.01×10^{-8}	6.01×10^{-8}

can be summarized as follows:

1. Select a secret variable $x \in \mathrm{GF}(2^8)$ and create an (n,d)-sharing of x as $(F_i)_{(0\leq i<n)}$.
2. Select k faults $\sigma_i \in \mathrm{GF}(2^8) \setminus \{0\}$ and inject the faults to the first k shares of x.
3. Process the detection on the $Exp254((F_i)_{0\leq i<n})$ and the $\tau_a \circ Exp254((F_i)_{0\leq i<n})$.

For example in the $(4,1)$ case, even if faults spread to all shares, the probability of generating undetectable errors at most 2^{-12}, as expected. In each multiplication, faults are spread to $k' \leq \varepsilon + d$ shares and these shares become input for another multiplication. Therefore, $Propogation_\varepsilon$ changes for each multiplication. As seen in Table 3.11, if we increase n, the probability of undetectable faults decreases with respect to the conditions in Corollary 3.7. In these experiments, we maximize the attackers capabilities to simulate the real-world settings, and efficiently analyze the fault model and detection capabilities of our scheme. Remark that, for an (n,d)-scheme with k faults, the total number of secret shares is $(2^8)^{d+1}$ and the total number of all possible faults is $(2^8)^k$.
The number of undetectable faults are same in $Exp254$ and $\tau_a \circ Exp254$. Therefore, we can conclude that τ_a does not produce undetectable faults. And if the fault is detectable using the output of $Exp254$, the attacker should inject another fault to τ_A to generate undetectable faults. Moreover, in some experiments, all faults become detectable even if the propagation probability is not 1. Although we perform the experiments with maximized number of faults, randomness is added in the nature of multiplication. Therefore, the numbers are not exact values, but rather upper bounds.

Table 3.11: Probabilities of generating undetectable faults for $Exp254$ and $\tau_A \circ Exp254$ using `SAGE` simulation.

	# Secret Shares / # **Fault Injections**			
	$k=1$	$k=2$	$k=3$	$k=4$
	2^{16} / $\mathbf{2^8}$	2^8 / $\mathbf{2^{16}}$	2^8 / $\mathbf{2^{16}}$	2^8 / $\mathbf{2^{16}}$
(4,1)	1.45×10^{-5}	1.52×10^{-5}	3.04×10^{-7}	2.88×10^{-5}
(5,1)	2.40×10^{-7}	6.01×10^{-8}	6.01×10^{-8}	1.20×10^{-7}
(6,1)	0	0	0	0
(6,2)	0	0	1.20×10^{-7}	1.26×10^{-5}

3.6 Conclusion

Fault and side-channel attacks have become a real threat to cryptographic systems if the adversary can observe and interact with the physical implementation. In this chapter, we propose a new secure multiparty computation to achieve both fault and side-channel resistance. It is shown that the proposed schemes can be used to perform addition, affine transformation, multiplication, and squaring while resisting both well-defined fault and side-channel adversaries. One advantage of the proposed scheme is a reduced overhead, as only an extra operation within the error preserving multiplication is needed.

We define a new multiplication engine in such a way that, once a fault occurs, information about the error remains as a part of the shares. The error propagates through the algebraic operations with high probability. It will be detectable even after further computations on the shares. This gives implementers the choice to perform error detection regularly (for higher detection rates at a higher overhead) or only implicitly during recombination at the end of the circuit. The error detection method is based on the degree of the secret sharing polynomial, which increases when errors occur. After the initial increase, additional levels of logic operations can result in a loss of degree for faulty states, leaving a small probability of undetected errors. A secrecy-preserving fault detection operation is defined to perform the detection. Also the idea of forwarding faults allows us to delay any error detection as late as the final recombination step. We introduce a recombination gate which is used for both fault detection and reconstruction of the secret. Hence, fault detection can be carried out when the output is produced. Moreover, the recombination gate features another desired property: Infective Computation. If an error occurs, our scheme ensures that attackers cannot learn anything since the output is random.

Security properties of our scheme are given using ISW probing model and a formal

analysis of fault resistance. Every gadget, including fault detection operation and recombination operation is proven to be secure in ISW probing model using the reformulated t-SNI security notion. Also, the first formal security proof of the multiplication scheme [RP12] is proven within this work, since the previous scheme can be seen as a subset of our scheme. Fault detection of our scheme is examined using notion of *Propagation*. The error-detection capacities of each operations are formally given by analysing the undetectable faults for each operation.
We propose a practical C implementation AES-128, tested on a popular ultra-low power architecture, the ARM Cortex M0+ core. We also measure its performance and demonstrate its level of side-channel resistance by addressing a full leakage analysis including higher order moments on the SMC multiplication. Also, to show the fault resistance capabilities of the proposed scheme, we perform the experiments on the `SubBytes` operation which can be considered as the most to vulnerable part of AES. The implementation provides multiple masking schemes with different types of field operations and is easily portable to higher orders. Different masking orders with different field operations executed in constant time. The code provides a *fully constant execution flow with constant memory accesses*.

4

A White-Box Masking Scheme Resisting Computational and Algebraic Attacks

4.1 Motivation

White-box cryptography attempts to protect cryptographic secrets in pure software implementations. Due to their high utility, white-box cryptosystems (WBC) are deployed by the industry even though the security of these constructions is not well defined. A major breakthrough in generic cryptanalysis of WBC was Differential Computation Analysis (DCA), which requires minimal knowledge of the underlying white-box protection and also thwarts many obfuscation methods. To avert DCA, classic masking countermeasures originally intended to protect against highly related side-channel attacks have been proposed for use in WBC. However, due to the controlled environment of WBCs, new algebraic attacks against classic masking schemes have quickly been found. These algebraic DCA attacks break all classic masking countermeasures efficiently, as they are independent of the masking order.
In this chapter, we provide the first generic and combined masking scheme that resists state-of-the-art white-box attacks: computational and algebraic attacks. Classic masking schemes can be applied to WBC, however none of them can *individually* achieve security against both attacks. To fill this gap, we examine the `ISW` transformation introduced by Ishai et al. [ISW03] and extend it to the white-box context.

We improve the `ISW` transformation by adding a multiplicatively shared nonlinear share. This additional nonlinear share provides security against algebraic attacks. The secret sharing of our masking scheme then consists of two components; linear shares to resist computational attacks and non-linear shares to increase the degree of the decoding function to prevent algebraic attacks. We present the structure of generic masking that resists an arbitrary order computational and first or second-order algebraic attacks in Section 4.2. To analyze the security of our construction in Section 4.3, we focus on two security notions in cryptography: *probing security* addresses security against computational attacks, while *prediction security* addresses security against algebraic attacks. Remark from Section 2.1.1, the *probing model* was introduced by Ishai et al. [ISW03]. The model states that every tuple of n or less intermediate variables must be independent of any sensitive variable. As stated in [BRVW19], an n^{th}-order masking provides security against n^{th}-order probing attacks and n^{th}-order computational attacks with additional obfuscation layers.

To cover algebraic attacks, a new security notion called *Prediction Security* was defined in [BU18]. The prediction security of a circuit C (with an encoding function E), is based on the probability of an adversary ($\mathcal{A}$) to accurately predict values of any single function (of d^{th} order) over intermediate values computed in the circuit C (composed with encoding E). The aim of such a prediction is to distinguish two sequences of plaintexts (chosen by the adversary) by analyzing the corresponding software trace. For example, an n^{th}-order Boolean masking that is inherently protected against computational attacks is vulnerable against first-order algebraic attacks, since the adversary can utilize a linear function (i.e. a first-order function) and combine a subset of intermediate variables to recover the secret value.

In this work, we further show that the probing security and prediction security notions *are incomparable*. First, we prove that our masking scheme is indeed secure against computational attacks by showing that it is secure in the probing model with the given order using the non-interference notions by Barthe et al. [BBD+16]. We give a concrete construction for first and second-order prediction security and prove their security. We extend the security definitions given in [BU18] and give a novel composability proof for the second-order prediction secure constructions. Besides the formal proofs, we verify the probing security of our masking scheme using the tool `maskVerif` [BBC+19] for specific orders. Furthermore, we update and use the tool produced by [BU18] to experimentally verify the first-order prediction security of our scheme. The implementation that can be used with `maskVerif` and the updated version of the tool produced by [BU18] is available as open source.

https://github.com/UzL-ITS/white-box-masking

In Section 4.4 we introduce a proof-of-concept AES implementation to analyze the overhead and experimentally verify the security properties of our scheme using a simple leakage test. The analysis includes the number of needed gates and number of required randomness for different orders of protection. We show that our combined approach outperforms the previous approaches which required to combine two different masking schemes to resist both attacks.

Notation. First, we summarize the notation used throughout the chapter. In the following, we use some finite ring $(\mathbb{K}, \oplus, \otimes)$ with an *addition* operation $\oplus$ and a *multiplication* operation $\otimes$. We often omit the multiplication symbol $\otimes$ and thus write xy instead of $x \otimes y$. Although we introduce the notations using $\mathbb{K}$, we fix $\mathbb{K} = \mathrm{GF}(2)$. A vector space over $\mathbb{K}$ of dimension ℓ is denoted by $\mathbb{K}^\ell$. For $a, b \in \mathbb{Z}$ with $a < b$, we define $[a, b] := \{a, a+1, \ldots, b-1, b\}$. The letters $x, y, z, \ldots$ represent the sensitive variables. Random variables are represented by the letter r, with an index as r_i or r^i. To denote a random selection of a variable r from the field $\mathbb{K}$, we use $r \in_R \mathbb{K}$.

A *sensitive variable* x is split into $n+1$ linear shares $x_0, \ldots, x_n$ such that $x = \bigoplus_{i=0}^{n} x_i$ and a single share (e.g. x_0) split into $d+1$ non-linear shares $\tilde{x}_0, \ldots, \tilde{x}_d$ such that $x_0 = \prod_{j=0}^{d} \tilde{x}_j$. A vector of elements $(x_1, \ldots, x_n)$ is denoted by $\overline{x}$. For a subset $I \subseteq [0, n]$ of indices, we denote by $x_{|I} = (x_i)_{i \in I}$ the sub-vector of shares indexed by I.

A *gadget* G for a function $f \colon \mathbb{K}^a \to \mathbb{K}^b$ (with regard to a masking order) is an arithmetic circuit with $a \cdot (n+d+1)$ inputs and $b \cdot (n+d+1)$ outputs grouped into a vectors of shares $\overline{x}^{(1)}, \ldots, \overline{x}^{(a)}$, resp. b vectors of shares $\overline{y}^{(1)}, \ldots, \overline{y}^{(b)}$. The gadget needs to be correct, i. e. $G(\overline{x}^{(1)}, \ldots, \overline{x}^{(a)}) = (\overline{y}^{(1)}, \ldots, \overline{y}^{(b)})$ iff $f(x^{(1)}, \ldots, x^{(a)}) = (y^{(1)}, \ldots, y^{(b)})$ for all possible inputs and for all values generated by the random gates. The values assigned to wires that are not output wires are called *intermediate variables*.

As usual, we model the white-box implementations as Boolean circuits (C) represented by directed acyclic graphs. Each node in a circuit C, with $k > 0$ inputs, corresponds to a k-ary Boolean function. Nodes with the indegree equal to zero are called inputs of C and nodes with the outdegree equal to zero are called outputs of C.

Let z be an input of $C : \mathbb{F}_2^N \to \mathbb{F}_2^M$ and $\overline{\mathsf{x}} = (\mathsf{x}_1, \ldots, \mathsf{x}_N)$ be a vector of input nodes in some fixed order. For each node v in C, we say that it computes a Boolean function $f_v : \mathbb{F}_2^N \to \mathbb{F}_2$ defined as follows:

- for all $1 \leq i \leq N$ set $f_{\mathsf{x}_i}(z) = z_i$,

- for all non-input nodes v in C set $f_v(z) = g_v(f_{c_1}(z), \ldots, f_{c_k}(z))$, where $c_1, \ldots, c_k$ are nodes having an outgoing edge to v and $g_v : \mathbb{F}_2^k \to \mathbb{F}_2$.

The set of f_v for all nodes v in C is denoted $\mathcal{F}(C)$, the set of $f_{\mathtt{x}_i}$ for all input nodes $\mathtt{x}_i$ is denoted $\mathcal{X}(C)$, and the set of f_v for all non-input nodes v in C is denoted $\mathcal{F}(C \setminus \mathcal{X})$. Recall, that any Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ has a unique representation of the form $f(x) = \bigoplus_{b \in \mathbb{F}^n} a_b \, x_1^{b_1} \ldots x_n^{b_n}$, with $a_b \in \mathbb{F}_2$. The (algebraic) degree of f, denoted $\deg(f)$, is the maximum degree of a monomial $x_1^{b_1} \ldots x_n^{b_n}$, with $a_b = 1$. The bias of a Boolean function $g : \mathbb{F}_2^\ell \to \mathbb{F}_2$ is represented by $\mathcal{E}(\cdot)$ i.e., $\mathcal{E}(g) = |1/2 - wt(g)/2^\ell|$ and $wt(g)$ is the weight of g, i.e., the number of nonzero entries of its truth table. Bold numbers $\mathbf{0}$ and $\mathbf{1}$ are used to denote constant functions.
If $\mathcal{V} = \{g_1, \ldots, g_{|\mathcal{V}|}\}$ is a set of Boolean functions with the same domain $\mathbb{F}_2^n$ then the d-th order closure of $\mathcal{V}$ (denoted $\mathcal{V}^{(d)}$) we call the vector space of all functions obtained by composing any function of degree at most d with functions from $\mathcal{V}$, i.e., $\mathcal{V}^{(d)}$ contains functions of the form $f \circ (g_1(x), \ldots, g_{|\mathcal{V}|}(x))$ for all $f : \mathbb{F}_2^{|\mathcal{V}|} \to \mathbb{F}_2$, with $\deg(f) \leq d$. For example, $\mathcal{F}^{(1)}(C)$ is spanned by $\{\mathbf{1}\} \cup \mathcal{F}(C)$ and $\mathcal{F}^{(2)}(C)$ is spanned by $\{\mathbf{1}\} \cup \{g_i g_j \mid g_i, g_j \in \mathcal{F}(C)\}$.
In the next section, we introduce our masking scheme, which resists both computational and algebraic attacks by using an adapted version of the `ISW` transformation.

4.2 Secure Masking Construction

The proposed masking scheme is based on two ideas: an `ISW`-like masking to increase the number of shares required to eliminate computation attacks and using a multiplicative sharing to increase the degree of the decoding function. We call the first part linear sharing of order n and the second part non-linear sharing of degree d. The resulting construction is named (n, d)-masking. We start with the data transformation and define our masking function as:

$$\mathtt{Encode}(x, \tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_{n-1}) = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)\,,$$

where $\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_{n-1} \in_R \mathbb{F}_2$ are chosen randomly and independently from $\mathbb{F}_2$, and

$$x_n = x \oplus \textstyle\prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n-1} x_i \ .$$

Observe that our masking scheme is obtained from the `ISW` transformation by replacing the first share x_0 in `ISW` by a non-linear sharing $x_0 = \prod_{j=0}^{d} \tilde{x}_j$. The unmasking function

is defined as follows:

$$\texttt{Decode}(\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n) = \prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i.$$

The data transformation is followed by the transformations of each AND and XOR gate. Throughout the chapter, we define the transformed gates as `And` and `Xor` (or `And`$[n, d]$ and `Xor`$[n, d]$) gadgets respectively.

4.2.1 Gate Transformations

In this section the generic constructions for the `Xor` and `And` gadgets are presented. Additionally, we provide definition of the `RefreshMask` gadget, which is needed to protect against algebraic attacks. The scheme can be used for an arbitrary order n of linear masking and any degree d of the non-linear component. Though the constructions are general, the algebraic security depends on the variable structure (details can be found in Section 4.3). Depending on the non-linear degree d, the following intermediate variables need a special structure:

- The intermediate variable $\mathcal{U}$ used in `Xor` and specified in Equation (4.1),
- The intermediate variables $r_{j,0}$ in Equation (4.2), used in `And`,
- The intermediate variables $\mathcal{W}$ and $\mathcal{R}$ used in `RefreshMask`, Equation (4.3).

In the following descriptions we first introduce the *functionalities* of these variables which can be defined for arbitrary orders of n and d. Afterwards, we will show the *computational structure* of these variables for $d = 1$ and $d = 2$.
Let x and y be two bits and consider an (n, d)-masking scheme, i.e. x and y have been split into $(n+d+1)$ shares such that $\prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i = x$ and $\prod_{j=0}^{d} \tilde{y}_j \oplus \bigoplus_{i=1}^{n} y_i = y$.

`Xor`$[n, d]$ Gadget. A masked representation of $z = x \oplus y$ with $n + d + 1$ shares such that $\prod_{j=0}^{d} \tilde{z}_j \oplus \bigoplus_{i=1}^{n} z_i = z$ can be calculated as follows:

Step 0: The input shares are processed by `RefreshMask` gadgets;

$$\overline{x} \leftarrow \texttt{RefreshMask}(\overline{x}) \text{ and } \overline{y} \leftarrow \texttt{RefreshMask}(\overline{y}).$$

Step 1: The values of the non-linear shares are processed:

$$\tilde{z}_i = \tilde{x}_i \oplus \tilde{y}_i \text{ for } 0 \leq i \leq d.$$

Algorithm 7 $\texttt{Xor}(\overline{x}, \overline{y})$

Input: The shares $\overline{x} = ((\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]})$ and $\overline{y} = ((\tilde{y}_j)_{j\in[0,d]}, (y_i)_{i\in[1,n]})$.
Output: The shares of $x \oplus y$ as $\overline{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$.

$\overline{x} \leftarrow \texttt{RefreshMask}(\overline{x})$
$\overline{y} \leftarrow \texttt{RefreshMask}(\overline{y})$
for $0 \leq j \leq d$
    $\tilde{z}_j \leftarrow \tilde{x}_j \oplus \tilde{y}_j$
for $1 \leq i < n$
    $z_i \leftarrow x_i \oplus y_i$
$z_n \leftarrow x_n \oplus y_n \oplus \mathcal{U}$
return $\bar{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$

Step 2: Computation of linear shares:

$$z_i = \begin{cases} x_i \oplus y_i, & \text{for } 1 \leq i < n \\ x_i \oplus y_i \oplus \mathcal{U}, & \text{for } i = n. \end{cases}$$

where the functionality of $\mathcal{U}$ is defined as follows:

$$\mathcal{U} = \bigoplus_{\substack{I\subsetneq\{0,\ldots,d\} \\ I\neq\varnothing}} \prod_{i\in I} \tilde{x}_i \prod_{j\notin I} \tilde{y}_j \tag{4.1}$$

Moreover, we can introduce the computational structure of $\mathcal{U}$ for a secure masking scheme as follows:

- $\texttt{Xor}[n, 1]$: $\mathcal{U} = \tilde{x}_0\tilde{y}_1 \oplus \tilde{x}_1\tilde{y}_0$
- $\texttt{Xor}[n, 2]$: $\mathcal{U} = \tilde{x}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{y}_2(\tilde{x}_0 \oplus \tilde{y}_0)) \oplus \tilde{y}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{x}_0(\tilde{x}_2 \oplus \tilde{y}_2))$
- $\texttt{Xor}[n, d]$ for $d \geq 3$, the functionality of $\mathcal{U}$ can be defined as in Equation (4.1). However the computational structure should be described carefully in order not to create vulnerabilities in algebraic security.

$\texttt{And}[n, d]$ **Gadget.** A masked representation of $z = xy$ with $n + d + 1$ shares such that $\prod_{j=0}^{d} \tilde{z}_j \oplus \bigoplus_{i=1}^{n} z_i = z$ can be calculated as follows:

Step 0: The input shares are processed by `RefreshMask` gadgets;

$$\overline{x} \leftarrow \texttt{RefreshMask}(\overline{x}) \text{ and } \overline{y} \leftarrow \texttt{RefreshMask}(\overline{y}).$$

Step 1: The calculations of the values with multiplicative representation are processed. Additional random bits $r^{i,j}$ are generated in order to attain algebraic security in the second step.

$$\tilde{z}_i = \tilde{x}_i\tilde{y}_{i'} \oplus r^{i,1} \oplus \cdots \oplus r^{i,n} \text{ for } 0 \leq i \leq d \text{ where } i' = i+1 \bmod(d+1).$$

Step 2: The variables $r_{j,i}$ for $0 \leq i < j \leq n$ are generated as follows:

$$r_{j,i} = \begin{cases} (r_{i,j} \oplus (\tilde{x}_0 \cdots \tilde{x}_d)y_j) \oplus x_j(\tilde{y}_0 \cdots \tilde{y}_d), & \text{for } i = 0 \quad \textbf{(a)} \\ (r_{i,j} \oplus x_i y_j) \oplus x_j y_i, & \text{for } 1 \leq i \leq n \text{ where } r_{i,j} \in_R \mathbb{F}_2 \quad \textbf{(b)} \end{cases},$$

The calculations for $1 \leq i \leq n$ are processed as identical to the `ISW-And` gadget. However, for $i = 0$ the calculations require a special computational structure:

$$r_{j,0} = [r_{0,j} \oplus (\tilde{x}_0 \cdots \tilde{x}_d)y_j] \oplus x_j(\tilde{y}_0 \cdots \tilde{y}_d) \text{ for } 1 \leq j \leq n. \tag{4.2}$$

Observe that $r_{i,j}$ for $1 \leq i < j \leq n$ is assigned a uniformly random value. However, $r_{0,j}$ cannot be assigned as random. Instead, $r_{0,j}$ should be defined in such a way that the following equation holds:

$$\bigoplus_{j=1}^{n} r_{0,j} = \bigoplus_{\substack{I \subset \{0,\ldots,d\} \\ I \neq \varnothing}} \prod_{i \in I} \tilde{x}_i\tilde{y}_{i'} \prod_{j \notin I} (r^{j,1} \oplus \cdots \oplus r^{j,n}) \text{ where } i' = i+1 \bmod(d+1).$$

We denote the right-hand side of the above equation as $\mathcal{V}$. Note that the above functionality for $r_{j,0}$ (given on the right-hand side of Equation (4.2)) is not secure against an algebraic attack, even if it is only a first-order one. Below we provide a secure computational structure for the case of an $(n, 1)$ and $(n, 2)$-masking.

- $\texttt{And}[n, 1] : r_{j,0} = \tilde{x}_1(\tilde{x}_0 y_j \oplus r^{0,j}\tilde{y}_0) \oplus \tilde{y}_1(\tilde{y}_0 x_j \oplus r^{1,j}\tilde{x}_0) \oplus r^{1,j}(r^{0,1} \oplus \ldots \oplus r^{0,n})$.
- $\texttt{And}[n, 2] : r_{j,0} = \tilde{x}_0 \left[\tilde{x}_2(\tilde{x}_1 y_j \oplus r^{0,j}\tilde{y}_0) \oplus r^{1,j} v\tilde{y}_1\right] \oplus$
  $\tilde{y}_0 \left[\tilde{y}_1(\tilde{y}_2 x_j \oplus r^{1,j}\tilde{x}_2) \oplus r^{0,j} u\tilde{x}_2\right] \oplus$
  $\tilde{x}_0\tilde{y}_1(r^{1,j}\tilde{x}_2\tilde{y}_0 \oplus r^{2,j}\tilde{x}_1\tilde{y}_2) \oplus r^{0,j}\tilde{x}_1\tilde{y}_2(v \oplus \tilde{x}_2\tilde{y}_0) \oplus$
  $\tilde{x}_2\tilde{y}_0(r^{0,j}\tilde{x}_0 \oplus r^{1,j}\tilde{y}_1) \oplus uvr^{0,j}$.

  where $u = r^{1,1} \oplus \cdots \oplus r^{1,n}$ and $v = r^{2,1} \oplus \cdots \oplus r^{2,n}$.

Algorithm 8 $\texttt{And}(\overline{x}, \overline{y})$

Input: The shares $\overline{x} = ((\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]})$ and $\overline{y} = ((\tilde{y}_j)_{j\in[0,d]}, (y_i)_{i\in[1,n]})$.
Output: The vector of shares of xy as $\overline{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$.

$\overline{x} \leftarrow \texttt{RefreshMask}(\overline{x})$
$\overline{y} \leftarrow \texttt{RefreshMask}(\overline{y})$
for $0 \le i \le d$
    $\tilde{z}_i = \tilde{x}_i\tilde{y}_{i'}$ // $i' = i + 1 \bmod(d+1)$
    for $1 \le j \le n$
        $r^{i,j} \leftarrow \texttt{rand}(0,1)$
        $\tilde{z}_i = \tilde{z}_i \oplus r^{i,j}$
for $0 \le i \le n$
    for $i < j \le n$
        if $i = 0$ **then**
            $r_{j,0} \leftarrow$ as described in the text.
        else
            $r_{i,j} \leftarrow \texttt{rand}(0,1)$
            $r_{j,i} \leftarrow (r_{i,j} \oplus x_iy_j) \oplus x_jy_i$
for $1 \le i \le n$
    $z_i \leftarrow x_iy_i$
    for $0 \le j \le n$ and $j \ne i$
        $z_i \leftarrow z_i \oplus r_{i,j}$ // Denoted by $z_{i,j}$
return $\overline{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$

- $\texttt{And}[n,d]$ for $d \ge 3$ the circuit nodes that calculates $r_{j,0}$ should be structured in such a way that algebraic security properties are satisfied.

Step 3: The final step can be performed identical to an `ISW-And` gadget: For every $1 \le i \le n$, compute $z_i = x_iy_i \oplus \bigoplus_{i\ne j} r_{i,j}$.

$\texttt{RefreshMask}[n,d]$ **Gadget.** This operation has a crucial importance for generating an algebraically secure implementation. In fact, it has to be combined with each `Xor` and `And` gadget in order to obtain a fully secure masking scheme. The security details can be found in Section 4.3.

Step 1: For $0 \le i \le d$, calculate $\tilde{x}'_i = \tilde{x}_i \oplus \tilde{r}_i$ where $\tilde{r}_i \in_R \mathbb{F}_2$.

Step 2: First initialize $x'_i \leftarrow x_i$ for all $i \in [1,n]$ and for $1 \le i < j \le n$, calculate $x'_i = x'_i \oplus r_{i,j}$ and $x'_j = x'_j \oplus r_{i,j}$ where $r_{i,j} \in_R \mathbb{F}_2$.

Step 3: In the last step we sample $r_0 \in_R \mathbb{F}_2$ and define two intermediate variables as follows:

$$\mathcal{W}' = \bigoplus_{I \subsetneq \{0,\ldots,d\}} \prod_{i \in I} \tilde{x}_i \prod_{j \notin I} \tilde{r}_j \text{ and } \mathcal{W} = \bigoplus_{\substack{I \subsetneq \{0,\ldots,d\} \\ I \neq \varnothing}} \prod_{i \in I} (\tilde{x}_i \oplus r_0) \prod_{j \notin I} \tilde{r}_j,$$

Here, as usual, a product over the empty set I is evaluated as 1. Using the above equations we define the variable $\mathcal{R} = \mathcal{W} \oplus \mathcal{W}'$. Now, we can introduce the variables that need to be added to the final share x_n as:

$$x'_n \leftarrow x'_n \oplus \mathcal{W} \oplus \mathcal{R} \text{ where } \mathcal{R} = \mathcal{W}' \oplus \mathcal{W}. \tag{4.3}$$

Remark that we cannot directly add $\mathcal{W}'$ to the final share x_n due to algebraic security properties. Therefore, the variables $\mathcal{W}$ and $\mathcal{R}$ should be added to the final share in order to define an algebraically secure mask refreshing gadget. The computational structure of the circuit nodes to calculate $\mathcal{W}$ and $\mathcal{R}$ for `RefreshMask`$[n, 1]$ and `RefreshMask`$[n, 2]$ can be found below.

- `RefreshMask`$[n, 1] : \mathcal{W} = \tilde{r}_0(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0)$ and $\mathcal{R} = (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0) \oplus r_0$.
- `RefreshMask`$[n, 2] : \mathcal{W} = [\tilde{r}_2(\tilde{x}_0 \oplus r_0)][\tilde{r}_1 \oplus (\tilde{x}_1 \oplus r_0)] \oplus [\tilde{r}_1(\tilde{x}_2 \oplus r_0)][\tilde{r}_0 \oplus (\tilde{x}_0 \oplus r_0)] \oplus [\tilde{r}_0(\tilde{x}_1 \oplus r_0)][\tilde{r}_2 \oplus (\tilde{x}_2 \oplus r_0)]$

  $\mathcal{R} = (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0)(\tilde{r}_2 \oplus r_0) \oplus r_0 [\tilde{r}_2(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_0(\tilde{x}_1 \oplus r_0)] \oplus r_0 [\tilde{r}_2(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_2 \oplus r_0) \oplus \tilde{r}_0(\tilde{x}_2 \oplus r_0)]$.
- `RefreshMask`$[n, d]$ for $d \geq 3$ the circuit nodes that calculate $\mathcal{W}$ and $\mathcal{R}$ should be constructed in such a way that algebraic security properties are satisfied.

4.2.2 Correctness and Performance Analysis

Next, we introduce the transformation $\mathtt{T}^{(n,d)}$ to generate a Boolean circuit that is protected by an (n, d)-masking scheme by using the gadgets described in Section 4.2.1. The following lemma summarizes the correctness of the transformation $\mathtt{T}^{(n,d)}$ and proof can be found in Chapter A.

Lemma 4.1. *Let us denote the Boolean circuit C initialized with data D by $C[D]$. The transformation $\mathtt{T}^{(n,d)} : C[D] \mapsto C'[D']$ where C' uses* `And`*,* `Xor`*,* `RefreshMask` *gadgets*

Algorithm 9 `RefreshMask`$(\overline{x})$

Input: The shares $\overline{x} = ((\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]})$
Output: The shares $\overline{x} = ((\tilde{x}'_j)_{j\in[0,d]}, (x'_i)_{i\in[1,n]})$

for $0 \leq j \leq d$
  $\tilde{r}_j \leftarrow \texttt{rand}(0,1)$
  $\tilde{x}'_j \leftarrow \tilde{x}_j \oplus \tilde{r}_j$
for $1 \leq i \leq n$ $x'_i \leftarrow x_i$
for $1 \leq i \leq n$
  for $i+1 \leq j \leq n$
    $r_{i,j} \leftarrow \texttt{rand}(0,1)$
    $x'_i \leftarrow x'_i \oplus r_{i,j}$ // Denoted by $a_{i,j}$
    $x'_j \leftarrow x'_j \oplus r_{i,j}$ // Denoted by $b_{j,i}$
$r_0 \leftarrow \texttt{rand}(0,1)$ // r_0 is used to compute $\mathcal{W}$ and $\mathcal{R}$
$x'_n \leftarrow x'_n \oplus \mathcal{W} \oplus \mathcal{R}$
return $(((\tilde{x}'_j)_{j\in[0,d]}, (x'_i)_{i\in[1,n]})$

and `Encoding`, `Decoding` *functions described in Section 4.2 with randomness gates is a functionality preserving transformation, i.e.* $C[D]$ *and* $C'[D']$ *have the same input-output behavior.*

In conclusion, the transformation $\mathtt{T}^{(n,d)}$ can be used to transform any circuit to an (n,d)-masked circuit in a functionality preserving manner. Although we are using an n^{th} order linear masking, the scheme only provides an $(n-1)^{th}$ probing security. Due to the non-linear sharing, the masking loses one share to increase the decoding order. The algebraic security depends on the structure of Equations (4.1), (4.2), and (4.3) in each gadget as discussed above. Further details can be found in Section 4.3.2.

Performance Analysis. In order to compare our construction with the previous schemes, we analyze the performance of our scheme in terms of bitwise operations and randomness requirements. An analytical comparison of different orders and a comparison between the `ISW` transformation and (n,d)-masking scheme can be found in Table 4.1.

In the following analysis, for simplicity, we use the symbol vertical bar (|) to separate the number of Xor, And operations respectively. We exclude the `RefreshMask` gadgets inside the `Xor` and `And` gadgets to analyze the constructions straightforwardly. Since the structure of the special variables depends on the non-linear degree d, we use a symbolic approach to analyze the performance numbers for the higher orders (i.e. for $d \geq 3$). We

use subscripts to denote the number of operations within $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{R}$, e.g., $\mathcal{U}_x$ and $\mathcal{U}_a$ represent the number of bitwise Xor, And operations within $\mathcal{U}$ respectively.

Table 4.1: The number of bitwise operations in a masked `Xor`, `And` and `RefreshMask` (or `RefM` in short) gadget. Remark that $(n, 0)$-masking scheme corresponds to `ISW` gadgets. The last part of the table corresponds to the overhead of (n, d)-masking scheme compared to the `ISW` transformation.

	Xor	And	Randomness
`Xor`$[n, 0]$	$n + 1$	-	-
`And`$[n, 0]$	$2n(n + 1)$	$(n + 1)^2$	$n(n + 1)/2$
`RefM`$[n, 0]$	$n(n - 1)$	-	$n(n - 1)/2$
`Xor`$[n, 1]$	$n + 4$	2	-
`And`$[n, 1]$	$2n^2 + 5n - 1$	$n^2 + 7n + 2$	$n(n + 3)/2$
`RefM`$[n, 1]$	$n(n - 1) + 8$	3	$(n(n - 1)/2) + 2$
`Xor`$[n, 2]$	$n + 9$	6	-
`And`$[n, 2]$	$2n^2 + 15n - 2$	$n^2 + 27n + 3$	$n(n + 5)/2$
`RefM`$[n, 2]$	$n(n - 1) + 26$	16	$(n(n - 1)/2) + 3$
`Xor`$[n, d]$	$n + d + 2 + \mathcal{U}_x$	$\mathcal{U}_a$	-
`And`$[n, d]$	$n(2n + d - 1) + \mathcal{V}_x$	$n^2 + d + 1 + \mathcal{V}_a$	$n(n + 2d + 1)/2$
`RefM`$[n, d]$	$n(n - 1) + d + 1 + \mathcal{W}_x + \mathcal{R}_x$	$\mathcal{W}_a + \mathcal{R}_a$	$(n(n - 1)/2) + d + 1$
Overhead			
`Xor`$[n, d]$	$d + 1 + \mathcal{U}_x$	$\mathcal{U}_a$	-
`And`$[n, d]$	$n(2n + d - 3) + \mathcal{V}_x - 1$	$d + \mathcal{V}_a - n$	nd
`RefM`$[n, d]$	$d + 1 + \mathcal{W}_x + \mathcal{R}_x$	$\mathcal{W}_a + \mathcal{R}_a$	$d + 1$

As seen in Table 4.1, the `Xor` gadget can be transformed efficiently. The cost of the gadget in the `ISW` transformation is $n+1$ bitwise Xor operations while an (n, d)-masking requires $n + d + 2$ bitwise Xor operations and the additional cost of the variables $\mathcal{U}$. Therefore, the cost of the `Xor` gadget can be calculated as; $(n + d + 2 + \mathcal{U}_x)|\mathcal{U}_a$.
The cost of an `And` gadget can be analyzed easily by comparing it step-by-step with the `ISW` transformation. As seen in the construction in Section 4.2, the gadget can be divided into three stages.

- **Step 1** requires $n(d + 1)$ random bits; the cost of processing these values can be calculated as $n(d + 1)|d + 1$.
- **Step 2(a)** includes the calculations of $r_{j,0}$ for $1 \leq j \leq n$. For the $(n, 1)$ masking, $\mathcal{V}_x = 4n$ and $\mathcal{V}_a = 7n$. Additionally, the calculations of $r^{0,1} \oplus \ldots \oplus r^{0,n}$ require $n-1$

Xor operations. Similarly, for the $(n, 2)$ masking, we get $\mathcal{V}_x = 12n$ and $\mathcal{V}_a = 27n$. The intermediate variables u, v, and uv are calculated only once and they require $2(n-1)|1$ gates.

- **Step 2(b) & Step 3** involve the calculations of $r_{j,i}$ for $1 \leq i < j \leq n$, $i \neq 0$ and Step 3. These parts can be processed identical to the `ISW` transformation and cost $2n(n-1)|n^2$ gates, while the required number of random bits is $n(n-1)/2$. Observe that the cost of these parts are exactly the cost of an `ISW-AND` gadget with n shares.

To sum up, we express the cost of `And`$[n, d]$ gadget as $(n(2n+d-1)+\mathcal{V}_x)|(n^2+d+1+\mathcal{V}_a)$ gates, and the required randomness as $n(n+2d+1)/2$.

We analyze the performance of the `RefreshMask` gadget using a similar methodology. The total amount of required randomness and the number of required bitwise Xor operations can be calculated as $(n(n-1)/2)+d+1$ and $n(n-1)+d+1$ respectively. As in the previous gadgets, the calculations of $\mathcal{W}$ and $\mathcal{R}$ add more calculations to the structure. The numbers for `RefreshMask`$[n, 1]$ and `RefreshMask`$[n, 2]$ are given in Table 4.1.

Using the performance analysis, we show the exact overhead of our scheme. The numbers in the overhead section of Table 4.1 can be calculated by comparing the cost of the n^{th}-order `ISW` transformation with an (n, d)-masking scheme. As seen in the table, the cost principally depends on the calculation of the values $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{R}$, while the randomness is affected by the masking degrees n and d.

4.3 Security Against Computational and Algebraic Attacks

In this section, we prove that our proposed $[n, d]$ masking scheme resists both computational and algebraic attacks, up to $(n-1)^{th}$ order and d^{th} order, respectively. We use the definition of non-interference as defined in [BBD+16], which guarantees security against t probes for $t \leq n$ as proposed by Ishai et al. [ISW03], and the definition of security against algebraic attacks of order d as proposed in [BU18]. First, we recall briefly both security notions and then we prove that our (n, d) construction is secure against probing up to order $n-1$ and against algebraic attacks for $d = 1$ and $d = 2$. Remark that probing security implies security against computational attacks of the same order, since computational attacks correspond to side-channel attacks of the same order [BRVW19].

4.3.1 Security Notions

We first cover the security notions that we used to prove our security properties, starting with probing security as described in Section 2.1.
Note that security in the probing model is a necessary but not a *sufficient* condition for a secure white-box implementation, as probing secure implementations may still be vulnerable to algebraic attacks. To prevent algebraic attacks, *prediction security* is also necessary.

Definition 4.1 (Prediction Security (d-PS), [BU18]). Let $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit that takes N'-bit inputs, uses R_C random bits, and produces an M-bit output. Let $E : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary function that takes N-bit input, uses R_E random bits, and produces an N'-bit output. Consider the following security experiment with an adversary $\mathcal{A}$, an integer $d \geq 1$, and a predetermined variable $b \in \{0, 1\}$:

Algorithm 10 $\mathrm{PS}^{C,E,d}(\mathcal{A}, b)$

1: $(\tilde{f}, \overline{x}^{[0]}, \overline{x}^{[1]}, \tilde{y}, b) \leftarrow \mathcal{A}(C, E, d)$ where
$\tilde{f} \in \mathcal{F}^{(d)}(C), (\overline{x}^{[l]} = (x_1^{[l]}, \ldots, x_Q^{[l]}))_{l\in\{0,1\}}, x_i^{[l]} \in \mathbb{F}_2^N,\ \tilde{y} \in \mathbb{F}_2^Q$ and $l \in \{0, 1\}$
2: $(r_1, \ldots, r_Q) \in_R (\mathbb{F}_2^{R_E})^Q$
3: $(\tilde{r}_1, \ldots, \tilde{r}_Q) \in_R (\mathbb{F}_2^{R_C})^Q$
4: **for** $f \in \mathcal{F}^{(d)}(C)$
5: $\quad y(f) = (f(E(x_1^{[b]}, r_1), \tilde{r}_1), \ldots, f(E(x_Q^{[b]}, r_Q), \tilde{r}_Q))$
6: $F \leftarrow \{f \in \mathcal{F}^{(d)}(C) \mid y(f) = \tilde{y}\}$
7: **if** $F = \{\tilde{f}\}$ **then return** 1 **else return** 0

Finally, we define the advantage of an adversary $\mathcal{A}$ as

$$\mathrm{Adv}^{\mathrm{PS}}_{C,E,d}[\mathcal{A}] \;=\; \left|\Pr[\mathrm{PS}^{C,E,d}(\mathcal{A}, 0) = 1] - \Pr[\mathrm{PS}^{C,E,d}(\mathcal{A}, 1) = 1]\right|.$$

The pair (C, E) is said to be d^{th} order prediction-secure (d-PS) if for any adversary $\mathcal{A}$ the advantage is negligible.

In summary, prediction security analyzes the behavior of functions from $\mathcal{F}^{(d)}(C)$ composed with an encoding function E. Consider two elements $x, x' \in \mathbb{F}_2^N$, if an adversary is able to find a function $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ such that $f(E(x, \cdot), \cdot)$ is constant (or high-bias) but $f(E(x', \cdot), \cdot)$ is non-constant (or low-bias) then the adversary can distinguish these inputs and therefore the pair (C, E) is considered insecure. Thus, prediction security requires every function from the set $\mathcal{F}^{(d)}(C)$ to have low-bias.

The outline of the security analysis can be summarized as follows. First our gadgets and encoding function have been proven to satisfy ϵ-1-AS, i.e. Definition 4.4 and Definition 4.5 (resp. ϵ-2-AS Definition 4.6 and Definition 4.7). To show that a gadget (or an encoding function) satisfies the corresponding definition, we analyze the bias of functions $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ where $d = 1$ or 2. The ϵ-1-AS definition implies that f is either an affine function of inputs or f has low bias when the input is fixed, i.e. when f is a function of random values with a constant input. Similarly, ϵ-2-AS implies that f is either *a second-order combination of affine functions of input and circuit nodes* or f has low bias when the input is fixed, i.e. when f is a function of random values with a constant input.

The proofs of individual gadgets and encoding functions are followed by the composability results. The composability of 1-AS gadgets is proven as in [BU18]. On the other hand the composability of 2-AS gadgets are extended in a non-trivial way and the composability of 2-AS gadgets are given in two fold; an arbitrary combination of two gadgets operated in parallel order (Proposition 4.10) and in sequential order (Proposition 4.11). Since the former analysis includes the linear combination of the nodes, a straightforward approach is enough to prove the composability. In our approach, we need to check the non-linear combinations of circuit nodes within the different gadgets.

Using the above steps, an arbitrary circuit, generated by $\texttt{And}[n, d]$, $\texttt{Xor}[n, d]$ and $\texttt{RefreshMask}[n, d]$ gadgets for $d = 1$ and $d = 2$ are shown to satisfy algebraic security definitions respectively.

Next we prove the composability of an arbitrary circuit C with an encoding function E in Proposition 4.7 and Proposition 4.12 and show that $C(E(\cdot))$ satisfies algebraic security definitions respectively. This implies that every function from the set $\mathcal{F}^d(C(E(\cdot)))$ has bias $\leq \epsilon$.

However to achieve prediction security we need a better security bound than ϵ. To prove that a circuit is prediction secure we need to show that $\mathrm{Adv}^{\mathrm{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{-\kappa}$ depending on a security parameter κ and this can be achieved by adding *dummy* random nodes to the circuit. Using the maximum bias bound ϵ, the required number *dummy* random bits to achieve a security bound κ is restated in Proposition 4.1.

In [BU18], it is shown that a circuit C achieves d-PS, if there are enough random bits used in the circuit depending on the bias bound. Before giving the relation between random bits and d-PS, we first remark the set of functions that is needed for the analysis.

Definition 4.2 ([BU18]). Let $C(\overline{x}', \overline{r}_c) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit, $E(\overline{x}, \overline{r}_e) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ an arbitrary function and $d \geq 1$ an integer. For any function $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ and for any $\overline{x} \in \mathbb{F}_2^N$ define $f_{\overline{x}} : \mathbb{F}_2^{R_E} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2$ given by

$f_{\overline{x}}(\overline{r}_e, \overline{r}_c) = f(E(\overline{x}, \cdot), \cdot)$ and denote the set of all such functions as $\mathcal{R}_d$:

$$\mathcal{R}_d = \{f_{\overline{x}}(\overline{r}_e, \overline{r}_c) \mid f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}, \overline{x} \in \mathbb{F}_2^N\}.$$

Using the above definition, a bound for the d-PS is given as in the following proposition which indicates the required number of random bits.

Proposition 4.1 (Corollary 1 in [BU18]). *Let ϵ be the maximum bias among all functions from $\mathcal{R}_d$, i.e., $\epsilon = \max_{f_{\overline{x}} \in \mathcal{R}_d} \mathcal{E}(f_{\overline{x}})$. Let $e = -\log_2(1/2 + \epsilon)$ and κ be a security parameter. Then for any adversary $\mathcal{A}$ choosing vector size of Q:*

$$Adv^{PS}_{C,E,d}[\mathcal{A}] \leq 2^{-\kappa}$$

if $e > 0$ and $R_C \geq \kappa \cdot (1 + 1/e)$.

Although it may seem that prediction security covers probing security (or vice versa), both notions are in fact *incomparable*. Therefore, both notions are needed to analyze a secure white-box implementation. To illustrate the incomparability of the two notions, let us consider two examples; a white-box implementation protected with an *n^{th}-order Boolean masking* and *minimalist quadratic masking* defined in [BU18].

Example 4.1 (Probing Secure Masking Vulnerable to Algebraic Attacks). *Applying an `ISW` transformation to the circuit and the data results in an n^{th}-order probing secure implementation. However, a first-order algebraic attack can exploit a first-order (linear) combination of intermediate values which is equal to a predictable value. Therefore, an n^{th}-order Boolean masking is secure in the probing model, but not secure in prediction security, as shown in [BU18].*

Example 4.2 (Algebraically secure masking vulnerable to probing). *As the second example, we use the encoding function `Encode`$(x, x_0, x_1) = (x_0, x_1, x_0 x_1 \oplus x)$. As given in [BU18], the masking scheme satisfies first-order algebraic security. However, it is not probing secure, not even first-order probing secure. A leakage is caused by the* unbalanced *sharing where the third share $x_0 x_1 \oplus x$ statistically depends on the sensitive variable x. For any value x we have $\Pr_{x_0, x_1 \in_R \mathbb{F}_2}[(x_0 x_1 \oplus x) = x] = 3/4$. Thus, there exists no first-order function that is equal to a predictable vector, but there exists one node (the last share) that is highly correlated with a predictable vector.*
To practically verify this leakage, we implement a basic bitwise AES-128 circuit using the Sbox designed by Boyar and Peralta [BP09] and implement a basic leakage detection test using 500 traces with 45000 nodes ($N = 500$ and $M = 45000$). As seen in Fig. 4.1, the

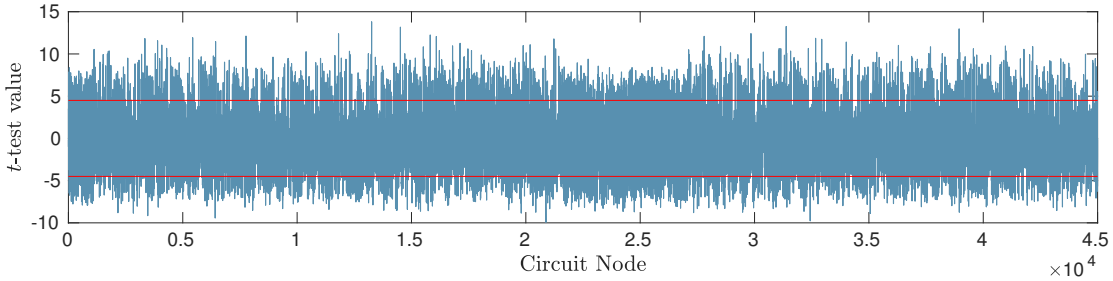


Figure 4.1: A first-order leakage detection on a circuit that simulates AES-128 with the masking defined in [BU18]. Clearly, the t-test value exceeds the threshold values shown by red lines.

test shows intense leakage. The details of the experimental setup regarding the leakage detection, trace collection and the variable selection can be found Section 4.4.1.

As illustrated in Example 4.1, prediction security is based on finding a degree-d function whose output equals to a predictable value. However, in probing we only need to find a set of variables which depends on a predictable value, as shown in Example 4.2. Thus, we need to prove the security of our scheme in two steps:

1. Prove probing security using Definition 2.1 for an arbitrary (n, d) scheme,
2. Prove prediction security using Definition 4.1 for $(n, 1)$ and $(n, 2)$ schemes.

4.3.2 Security Against Computational Attacks in the Probing Model

We first provide some auxiliary definitions that form the basis of our security proofs.

Definition 4.3 ((n, d)-family of shares). A vector $\overline{x} = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ of $n+d+1$ intermediate variables is called an (n, d)-*family of shares* if every tuple of the form $((\tilde{x}_i)_{i\in\tilde{I}}, (x_i)_{i\in I})$ such that $|\tilde{I}| \leq d+1$ and $|I| \leq n-1$ of $\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n$ is uniformly distributed and independent of any sensitive variable where $x = \prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i$ is a sensitive variable.

We can extend the definition as: two (n, d)-families of shares $\overline{x} = (\tilde{x}_0 \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ and $\overline{y} = (\tilde{y}_0 \ldots, \tilde{y}_d, y_1, \ldots, y_n)$ are called to be $(n-1)$-*independent* of one another if every tuple composed of $((\tilde{x}_i)_{i\in\tilde{I}}, (x_i)_{i\in I})$ and $((\tilde{y}_j)_{j\in\tilde{J}}, (y_j)_{j\in J})$ with $|\tilde{I}|, |\tilde{J}| \leq d+1$ and $|I|, |J| \leq n-1$ is uniformly distributed and independent of any sensitive variable. Two (n, d)-families are $(n-1)$-*dependent* of one another if they are not $(n-1)$-independent. To prove security of our scheme in the t-SNI notion, we decompose C into basic components, which we call *randomized elementary transformations* (or *gadgets*). Such

a component gets as input two $(n-1)$-independent (n,d)-families of shares, resp. one (n,d)-family of shares, and it returns a (n,d)-family of shares.
In this section, we first prove that the randomized elementary transformations specified as in Algorithm 7, Algorithm 8, and Algorithm 9 satisfy non-interference notions. One challenge for proving t-SNI security results from the fact that in the proposed sharing only *a subset of shares* is uniformly distributed. The product of non-linear shares x_0 (as expressed by $x_0 = \prod_{j=0}^{d} \tilde{x}_j$) is non-uniformly distributed or *biased*. Hence, x_0 can be predicted correctly by an adversary with high probability. Thus, the non-linear shares do not contribute to probing security. To address this fact, we consider the non-linear shares as public values accessible by the adversary or as *free probes*, due to the bias of their product. We use the following fact in our proofs.

Fact 4.1. *Let G be a masked operation that operates on an (n,d)-family of shares $\overline{x} = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ (or two (n,d)-families of shares $\overline{x} = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ and $\overline{y} = (\tilde{y}_0, \ldots, \tilde{y}_d, y_1, \ldots, y_n)$) as defined in Definition 4.3. A simulator can access non-linear shares $(\tilde{x}_0, \ldots, \tilde{x}_d)$ (or $(\tilde{x}_0, \ldots, \tilde{x}_d)$ and $(\tilde{y}_0, \ldots, \tilde{y}_d)$) as* free probes *or public inputs to the gadgets.*

Now we are ready to prove the security of our basic constructions. We start with the t-SNI property of the `RefreshMask` and `And` gadgets. Then, we continue with the t-NI property of the `Xor` gadget.

Proposition 4.2. *(t-SNI of `RefreshMask`) Let $\overline{x} = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ be an (n,d)-family of shares, with $n \geq 2$, as input of Algorithm 9 to refresh masking. Then every tuple of t_1 intermediate variables and t_2 output variables in Algorithm 9 such that $t_1 + t_2 \leq t$ can be simulated by at most t_1 linear shares taken from $\overline{x}$.*

Proof. In order to prove the proposition, we first assume that the simulator can access the values $(\tilde{x}_i)_{i\in[0,d]}$ by Fact 4.1 and we show that every set of intermediate variables with t_1 elements and every set of output variables with t_2 such that $t_1 + t_2 \leq t$ can be simulated from a set of input shares $U = ((\tilde{x}_i)_{i\in[0,d]}, (x_i)_{i\in I})$ such that $|I| \leq t_1$.
Let us first classify the variables. The intermediate variables are x_i, $r_{i,j}$, $\tilde{x}_i$, $\tilde{r}_j$, $a_{i,j}$, $b_{j,i}$ (where $a_{i,j}$, $b_{j,i}$ as defined in Algorithm 9) and the intermediate variables within $\mathcal{W}$, $\mathcal{R}$ and the outputs are $\tilde{x}'_i$, x'_i.
Next, we can define I as follows:

- For each selected variable x_i, $r_{i,j}$ and $a_{i,j}$ add i to I and $b_{j,i}$ add j to I.
- For each selected $\tilde{r}_j$, $\tilde{x}_j$ and $\tilde{x}_j \oplus \tilde{r}_j$, we don't need to add any value since $\tilde{x}_j$ is accessible by the simulator.

- **Line 11, $\mathcal{W}$ and $\mathcal{R}$**: If one of the variables of form $\prod_{i\in J}(\tilde{x}_i \oplus r_0)\prod_{i\notin J}\tilde{r}_i$ where $J \subsetneq \{0,\dots,d\}$ is selected, no values need to be added due to Fact 4.1. If one of the variables inside $\mathcal{R}$ is selected, no values need to be added since in the expression only shares $\tilde{x}_i$ and random variables are used.

It is clear that I contains at most t_1 elements since each selected value adds at most one index to I.
Now we can define the simulator. For all $i \in I$ the simulator can sample all $r_{i,j}$ for $j \in [i+1, n]$ and compute all partials sums $a_{i,j}$ and $b_{j,i}$ and thus the output x'_i. For all $i \in [0, d]$ the simulator can sample $\tilde{r}_i$ and compute the output $\tilde{x}'_i$. Moreover the simulator can compute $\mathcal{W}$ and $\mathcal{R}$ by sampling random variables and computing $(\tilde{x}_i)_{i\in[0,d]}$.
Finally, we need to consider the simulation of the output shares x'_i such that $i \notin I$. Observe that $i \notin I$ means that any random value in the partial sum of x'_i is not probed and is not involved in a partial sum of it. Hence we can simulate x'_i by an uniformly random value. Also, by Fact 4.1 any output variable $\tilde{x}'_i$ can be simulated. As a result any set of t_1 selected intermediate variables and any set of t_2 output variables can simulated by $U = ((\tilde{x}_i)_{i\in[0,d]}, (x_i)_{i\in I})$ such that $|I| \leq t_1$. □

Proposition 4.3. *(t-SNI of `And`) Let $\overline{x} = (\tilde{x}_0, \dots, \tilde{x}_d, x_1, \dots, x_n)$ and $\overline{y} = (\tilde{y}_0, \dots, \tilde{y}_d, y_1, \dots, y_n)$ be two $(n-1)$-independent (n,d)-families of shares, with $n \geq 2$, inputs of Algorithm 8 for `And`. Then every tuple of t_1 intermediate variables and t_2 output variables such that $t_1 + t_2 \leq t$ can be simulated by at most t_1 linear shares taken from $\overline{x}$ and $\overline{y}$.*

Proof. In order to prove the proposition, we first assume that the simulator can access the values $(\tilde{x}_i)_{i\in[0,d]}$ and $(\tilde{y}_i)_{i\in[0,d]}$ by Fact 4.1. Then we show that every set of t_1 intermediate variables and every set of t_2 output variables such that $t_1 + t_2 \leq t$ can be simulated by two sets of input shares $(\tilde{x}_i)_{i\in[0,d]}$ and $(x_i)_{i\in I}$ such that $|I| \leq t_1$, resp. $(\tilde{y}_j)_{j\in[0,d]}$ and $(y_j)_{j\in J}$ such that $|J| \leq t_1$.
We first need to construct the sets of indices I and J corresponding to the shares of x and y. The following two cases cover every variable in **Step 2(b)** and **Step 3**:

Group 1: For all x_i, y_i, $x_i y_i$, add i to I and J.

Group 2: For all $r_{i,j}$ or $z_{i,j}$ add, i to I and J where $z_{i,j}$ denotes the j^{th} partial sum of z_i.

Note that after thees steps, we have $I = J$ and we denote this common set as U.

Group 3: For all $x_i y_j \oplus r_{i,j}$, if $i \in U$ or $j \in U$, add both i, j to I and J.

Group 4: For all $x_i y_j$ add, i to I and j to J.

To cover **Step 1** and **Step 2(a)** we need to use the following classification:

Group 5: : For all $\tilde{x}_i$, $\tilde{y}_i$, $r^{i,j}$ and combination of these, no values are needed to be added due to Fact 4.1.

Group 6: For all $\tilde{x}_i y_j$ (resp. $\tilde{y}_j x_i$), add j to J (resp. i to I).

Group 7: For all values of the form $\prod_{i\in K} \tilde{x}_i \prod_{j\in L} \tilde{y}_j$ where $K, L \subsetneq \{0, \ldots, d\}$, no values are needed to be added due to Fact 4.1.

Clearly, I and J have at most one index per selected variable, and therefore $|I| \leq t_1$ and $|J| \leq t_1$.
We now define the simulator for the intermediate variables. The simulation of the variables in Group 1 and Group 4 can be performed easily.

Group 1: To simulate x_i, y_i, or $x_i y_i$, we can simply use the input variables, as both x_i and y_i are known from I and J.

Group 4: To simulate $x_i y_j$ we can simply use the input variables, as both x_i and y_j are known from I and J.

For the remaining groups **Group 1** and **Group 4** (i. e. probed variables $r_{j,i}$, $z_{i,j}$ or $x_i y_j \oplus r_{i,j}$), we use the following claim.

Claim 4.1. *If $i \notin U$, then $r_{i,j}$ is not selected and does not enter in the computation of any probed $z_{i,k}$. Similarly, if $j \notin U$, then $r_{j,i}$ is not selected and does not enter in the computation of any selected $c_{j,k}$.*

Proof. For $i < j$, the variable $r_{i,j}$ is used in all partial sums $c_{i,k}$ for $k > j$. Also $r_{i,j}$ is used in $r_{i,j} \oplus x_i y_j$, which is a part of $r_{j,i}$. Note that $r_{j,i}$ is used in all partial sums $c_{i,k}$ for $k > i$. □

For $1 < i < j$ let us consider the following cases:

Case 1: $\{i, j\} \in U$ means that all the variables $r_{i,j}$, $x_i y_j$, $x_i y_j \oplus r_{i,j}$, $x_j y_i$ and $r_{j,i}$ can be perfectly simulated while simulating $r_{i,j}$ by a uniformly random value.

Case 2: $i \in U$ and $j \notin U$ implies that we can simulate $r_{i,j}$ as a uniformly random value and if $x_i y_j \oplus r_{i,j}$ is also selected we can perfectly simulate it since $i \in U$ and $j \in J$ by Claim 4.1.

Case 3: $i \notin U$ and $j \in U$ indicates that any variable of the form $r_{i,j}$ and $z_{i,j}$ is not selected by Claim 4.1. More importantly $r_{i,j}$ is not used in any other selected value. Thus, we can simulate $r_{j,i}$ with a uniformly random value. Also we can simulate $x_i y_j \oplus r_{i,j}$ (observe that $x_i y_j \oplus r_{i,j} = x_j y_i \oplus r_{j,i}$) since $j \in U$ and $i \in J$.

Case 4: $i \notin U$ and $j \notin U$ means that if $x_i y_j \oplus r_{i,j}$ is selected we can simply simulate it with a uniformly random value, since $r_{i,j}$ is not selected and does not enter any calculation.

From the above analysis we can see that any variable $r_{i,j}$ can be simulated if $i \in U$ including all partial sums $z_{i,k}$ and z_i. Now, we need to consider the variables from **Step 1** and **Step 2(a)**.

- Every variable $\tilde{x}_i$, $\tilde{y}_i$, $\tilde{x}_i\tilde{y}_{i'}$, $\tilde{x}_i y_j$, $\tilde{y}_j x_i$ $r^{i,j}$ or xor of these values can be simulated according to Fact 4.1.
- Every variable in **Step 2(a)** can be simulated since the simulator accesses $\tilde{x}_{i\in[0,d]}$ and $\tilde{y}_{j\in[0,d]}$.

Hence, we show that any set of intermediate variables, with t_1 elements can be simulated by the sets $((\tilde{x}_i)_{i\in[0,d]}, (x_i)_{i\in I})$ and $((\tilde{y}_j)_{j\in[0,d]}, (y_j)_{j\in J})$ which are uniformly random and independent of any sensitive variable.
In the last part of the proof, we focus on the simulation of an arbitrary set of output variables $((\tilde{z}_i)_{i\in\tilde{\mathcal{O}}}, (z_i)_{i\in\mathcal{O}})$ where $\tilde{\mathcal{O}} \subset [0,d]$ and $\mathcal{O} \subset [1,n]$ with t_2 elements such that $t_1 + t_2 \leq t$. Let us first analyze the non-linear output shares $(\tilde{z}_i)_{i\in\tilde{\mathcal{O}}}$. Observe that we can simulate $r^{i,j}$ as uniformly random values and perfectly simulate $\tilde{z}_i$ by Fact 4.1.
Next, we focus on the output shares $(z_i)_{i\in\mathcal{O}}$. From the discussion above, we can see that we can simulate outputs z_i with $i \in U$ perfectly. Now, consider z_i with $i \notin U$. A set of indices V is constructed as follows: For each variable $x_i y_j \oplus r_{i,j}$ in Group 3 with $i \notin U$ and $j \notin U$ (corresponding to Case 4 described above), we add j to V if $i \in \mathcal{O}$ or i to V if $i \notin \mathcal{O}$. Note that we only considered variables in Group 3, where we increased I and J by two elements. As V was only increased by one element, we have $|U| + |V| \leq t_1$ and thus $|U| + |V| + |\tilde{\mathcal{O}}| + |\mathcal{O}| < n$. Hence, there is an index $j^* \in [0,n]$ such that $j^* \notin (U \cup V \cup \mathcal{O})$. By definition, we have

$$z_i = x_i y_i \oplus \bigoplus_{j=0;j\neq i}^{n} r_{i,j} = r_{i,j^*} \oplus \left(x_i y_i \oplus \bigoplus_{j=0;j\neq i;j\neq j^*}^{n} r_{i,j} \right).$$

We will now show that r_{i,j^*} and $r_{j^*,i}$ are not processed in the computation of any selected intermediate variable or another output variable $z_{i'}$ with $i' \in \mathcal{O}$. Observe that, if $i \notin U$ (resp. $j^* \notin U$) neither r_{i,j^*} (resp. $r_{j^*,i}$) nor any partial sum $z_{i,k}$ (resp. $z_{j^*,k}$) was selected. Therefore, $j^* \notin \mathcal{O}$ and z_{j^*} were also not selected. Hence, r_{i,j^*} and $r_{j^*,i}$ are not used in the computation of a selected intermediate variable.
In the last part of the proof, we need to show that r_{i,j^*} and $r_{j^*,i}$ are not needed for other output variables $z_{i'}$.
If $i < j^*$, then $x_i y_{j'} \oplus r_{i,j^*}$ was not selected (since $j^* \notin V$ and $i \in \mathcal{O}$). If $j^* < i$, then $x_{j*} y_i \oplus r_{j^*,i}$ was not selected (since $j^* \notin (V \cup \mathcal{O})$. Hence, r_{i,j^*} and $r_{j^*,i}$ are not used in the computation of any output variable $z_{i'}$ and we simulate z_i by sampling a random value. □

Proposition 4.4. *(t-NI of `Xor`) Let $\overline{x} = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ and $\overline{y} = (\tilde{y}_0, \ldots, \tilde{y}_d, y_1, \ldots, y_n)$ be two $(n-1)$-independent (n,d)-families of shares, with $n \geq 2$, as input of Algorithm 7 to compute `Xor`. Then every tuple of t_1 intermediate variables and t_2 output variables such that $t_1 + t_2 \leq t$ can be simulated by at most $t_1 + t_2$ linear shares taken from $\overline{x}$ and $\overline{y}$.*

Proof. In order to prove the proposition, we first assume that the simulator can access the values $(\tilde{x}_i)_{i\in[0,d]}$ and $(\tilde{y}_i)_{i\in[0,d]}$ and show that every set of intermediate variables including the output shares with $\leq t$ elements can be simulated by two sets of input shares $(\tilde{x}_i)_{i\in[0,d]}$ and $(x_i)_{i\in I}$ such that $|I| \leq t$ (resp. $(\tilde{y}_j)_{j\in[0,d]}$ and $(y_j)_{j\in J}$ such that $|J| \leq t$). We denote the concatenations of these tuples by $U = ((\tilde{x}_i)_{i\in[0,d]}, (x_i)_{i\in I})$ and $V = ((\tilde{y}_j)_{j\in[0,d]}, (y_j)_{j\in J})$.
We can define I and J as follows: for each selected x_i, y_i, $x_i \oplus y_i$ add i to I and J. Due to Fact 4.1, all selected variables $\tilde{x}_i$, $\tilde{y}_i$, $\tilde{x}_i \oplus \tilde{y}_i$, $\tilde{x}_i\tilde{y}_j$ and $\prod_{i\in K} \tilde{x}_i \prod_{j\notin K} \tilde{y}_j$, doe not increase the size of the sets I and J.
It is clear that I and J contains at most t elements since each selected variable adds at most one index to I and/or J. Remark that the above classification also covers the output shares.
Now we can define the simulator. Every variable of the form x_i, y_i, $x_i \oplus y_i$ (resp. $\tilde{x}_i$, $\tilde{y}_i$, $\tilde{x}_i \oplus \tilde{y}_i$) can be simulated by the sets U and V. Moreover every variable of the form $\prod_{i\in K} \tilde{x}_i \prod_{j\notin K} \tilde{y}_j$ where $K \subsetneq \{0, \ldots, d\}$ can be simulated by Fact 4.1. □

In conclusion, we prove the security against t probes of our individual gadgets such that $t < n$. Next, we analyze an arbitrary circuit C as a combination of our gadgets. As stated in [BBD+16], an algorithm is said to be t-NI if all gadgets are t-NI and every

non-linear usage of a secret state is guarded by t-SNI refreshing gadgets. Moreover, it is sufficient to make the algorithm t-SNI, if every input or the output of a t-NI gadget is processed by a t-SNI gadget. Since the `RefreshMask` operation is proven to be secure in the t-SNI notion, we can use the operation defined in Section 4.2.1 to generate an arbitrary circuit that is secure against $(n-1)^{th}$-order probing attacks and therefore secure against $(n-1)^{th}$-order computational attacks.

Experimental Verification. To support the results, we provide an experimental verification of the gadgets `And`$[n,d]$, `Xor`$[n,d]$ and `RefreshMask`$[n,d]$ for $d = 1$ and 2 and for $n = 1,2,3,4$ and 5 using `maskVerif` [BBC+19]. We implement our masking scheme (with the given orders n and d) inside the tool and experimentally verify the security features of our gadgets. The implementation of our scheme that can be used with `maskVerif` is available as open source [1]. The experiments are run on an Intel Core i5-6400 CPU@ 2.70GHz. A summary of the experimental results is given in Table 4.2.

[1] https://github.com/UzL-ITS/white-box-masking

Table 4.2: A summary of SNI/NI/Probing security verification of our gadgets. The inputs of an (n, d) gadget are two $(n-1)$-independent families of shares $\bar{x}$ and $\bar{y}$ (resp. one $(n-1)$-independent families of shares $\bar{x}$ for `RefreshMask` or `RefM` in short). The number of observations (#Obs.) represents the total number of (intermediate and output) variables within the specified gadget. The timing corresponds to the verification.

	Free Probes	# Obs.	SNI	NI	Probing
`RefM`$[2,1]$	$[\tilde{x}_0, \tilde{x}_1]$	23	0.01s	0.01s	0.01s
`Xor`$[2,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	16	-	0.01s	0.05s
`And`$[2,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	52	0.01s	0.01s	<0.01s
`RefM`$[3,1]$	$[\tilde{x}_0, \tilde{x}_1]$	30	0.01s	0.01s	0.01s
`Xor`$[3,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	19	-	0.01s	0.01s
`And`$[3,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	86	0.02s	0.02s	0.02s
`RefM`$[4,1]$	$[\tilde{x}_0, \tilde{x}_1]$	40	0.02s	0.01s	<0.01s
`Xor`$[4,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	22	-	0.01s	0.01s
`And`$[4,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	123	0.06s	0.05s	0.05s
`RefM`$[5,1]$	$[\tilde{x}_0, \tilde{x}_1]$	53	0.05s	0.01s	0.01s
`Xor`$[5,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	48	-	0.01s	0.01s
`And`$[5,1]$	$[\tilde{x}_0, \tilde{x}_1]$, $[\tilde{y}_0, \tilde{y}_1]$	170	2.25s	0.59s	0.45s
`RefM`$[2,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$	49	0.01s	0.01s	0.01s
`Xor`$[2,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	24	-	<0.01s	0.01s
`And`$[2,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	98	0.02s	0.01s	0.01s
`RefM`$[3,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$	56	0.01s	0.01s	0.01s
`Xor`$[3,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	27	-	0.01s	0.01s
`And`$[3,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	154	0.03s	0.01s	0.02s
`RefM`$[4,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$	66	0.02s	0.01s	0.01s
`Xor`$[4,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	30	-	0.01s	0.01s
`And`$[4,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	215	0.61s	0.08s	0.07s
`RefM`$[5,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$	79	0.04s	0.01s	0.01s
`Xor`$[5,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	33	-	0.01s	0.01s
`And`$[5,2]$	$[\tilde{x}_0, \tilde{x}_1, \tilde{x}_2]$, $[\tilde{y}_0, \tilde{y}_1, \tilde{y}_2]$	284	10.47s	1.06s	1.11s

4.3.3 Algebraic Security of the $(n, 1)$Masking Scheme

In this section, we analyze the first order prediction security (Definition 4.1) of our $(n, 1)$-masking scheme using the gadgets from Section 4.2.1. We proceed as follows. For our encoding function E and any Boolean circuit C constructed from our gadgets, we estimate $\epsilon = \max_{f_{\bar{x}} \in \mathcal{R}_1} \mathcal{E}(f_{\bar{x}})$ – the maximum bias over all functions in the set $\mathcal{R}_1$ specified in Definition 4.2, with $d = 1$. We show that ϵ is small. The bound on the prediction security of the $(n, 1)$-masking, we get combining this bias with the upper bound on 1-PS shown in Proposition 4.1: $\mathrm{Adv}^{\mathrm{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{-\kappa}$, assuming $e = -\log_2(1/2 + \epsilon) > 0$ and the number of random bits of the circuit C is $\geq \kappa \cdot (1 + 1/e)$.

To estimate the maximum bias ϵ we use two auxiliary notions: *algebraic encoding security* which addresses the security of the encoding function E and *algebraic circuit security* which addresses the security of any Boolean function C:

Definition 4.4 (Algebraic Encoding Security (ϵ-1-AS)). Let $E(\overline{x}, \overline{r}) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary encoding function. Let $\mathcal{Y}$ be the set of functions given by the output bits of E. The function E is called 1^{st}-order algebraically ϵ-secure (ϵ-1-AS) if for any $f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and for any $\overline{x} \in \mathbb{F}_2^N$ the bias of the function $f(\overline{x}, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ:

$$\max_{f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}, \overline{x} \in \mathbb{F}_2^N} \mathcal{E}(f(\overline{x}, \cdot)) \leq \epsilon$$

Definition 4.5 (Algebraic Circuit Security (ϵ-1-AS)). Let $C(\overline{x}, \overline{r}) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit and let ϵ be a real number, with $0 \leq \epsilon < 1/2$. Then C is called first-order algebraically ϵ-secure (ϵ-1-AS) if for any $f \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ one of the following conditions holds:

(a) f is an affine function of $\overline{x}$,

(b) for any $\overline{x} \in \mathbb{F}_2^N$, $\mathcal{E}(f(\overline{x}, \cdot)) \leq \epsilon$ where $f(\overline{x}, \cdot) : \mathbb{F}_2^{R_C} \to \mathbb{F}_2$.

In the rest of this section we show that our basic gadgets are ϵ-1-AS, for some $\epsilon < 1/2$ (Section 4.3.3.1). Using the fact that composition of two ϵ-1-AS circuits remains ϵ-1-AS [BU18] we get that the whole circuit C is ϵ-1-AS. In Section 4.3.3.2 we show that our encoding function E is ϵ-1-AS, for some $\epsilon < 1/2$, and finally we conclude that $\max_{f_{\overline{x}} \in \mathcal{R}_d} \mathcal{E}(f_{\overline{x}}) \leq \epsilon$.

4.3.3.1 ϵ-1-AS of the Gadgets

Using the above definitions, we employ the following methodology to prove the 1-PS of our scheme. We first divide the circuit into smaller circuits (namely `And`, `Xor` and `RefreshMask` gadgets as defined in Section 4.2.1) and show that the gadgets satisfy Definition 4.4. This gives us a bias bound for the individual gadgets. Using the 1-AS composability result in [BU18] we make sure that any composition of our gadgets (C) is also 1-AS. Finally, we combine C with the encoding function E and complete the 1-PS security proof of our scheme by using Proposition 4.1 .
While proving algebraic encoding security is quite straightforward, proving algebraic circuit security needs significant attention. The methodology to prove algebraic circuit security in [BU18] can be divided into two steps. The first step consists of showing

$\mathcal{E}(f(\overline{x},\overline{r})) \neq 1/2$ for all $f \in \mathcal{F}^{(1)}(C)$ and for all $\overline{x} \in \mathbb{F}_2^N$ except for constant functions and affine functions of $\overline{x}$. A verification algorithm is provided in [BU18]. The algorithm generates a truth table by evaluating the circuit on all possible inputs and records each node in the circuit. Another truth table is formed by selecting the values where the input is fixed $\overline{x} = \overline{c}$. That is, the second truth table corresponds to the values of the circuit nodes where the input $\overline{x}$ is fixed to a value $\overline{c}$ while $\overline{r}$ takes all possible values. Observe that the latter truth table is a subset of the former one. Finally, the algorithm compares the dimensions of the basis of the truth tables for each restriction, to check if there is a constant function f when the input is fixed to a value $\overline{c}$.
The second step is to find the maximum degree term (i.e. node in the circuit) and calculate the corresponding bias bound. As proven in [MS77], the degree of any nonzero Boolean function $g : \mathbb{F}_2^N \to \mathbb{F}_2$ gives us the following lower bound for the weight of the function: $wt(g) \geq 2^{N-\deg(g)}$, where N is the number of inputs of the function g. Symmetrically[2], we get that for any function $g \neq \mathbf{1}$ it is true that $wt(g) \leq 2^N - 2^{N-\deg(g)}$. Thus, for any non-constant function g we have:

$$2^{N-\deg(g)} \leq wt(g) \leq 2^N - 2^{N-\deg(g)}.$$

Using these inequalities we can analyze the bias bound of any non-constant function $f \in \mathcal{F}^{(1)}(C)$ as follows. First, we bound ϵ as:

$$\epsilon \;=\; \left|\frac{1}{2} - \frac{wt(f)}{2^N}\right| \;\leq\; \frac{1}{2} - \frac{2^{N-\deg(f)}}{2^N} \;=\; \frac{1}{2} - \frac{1}{2^{\deg(f)}}. \tag{4.4}$$

Next, observe that the maximum degree of f is equal to the maximum degree node in C, since f contains *only* linear combinations of the nodes. That is, for all $f \in \mathcal{F}^{(1)}(C)$, $\deg(f) \leq \max(\deg(c_i)_{c_i \in C})$. Thus, the linear-bias bound of the gadget can be estimated as:

$$\epsilon \;\leq\; \frac{1}{2} - 1/2^{\max(\deg(c_i)_{c_i \in C})}.$$

Due to the first part of the proof, we know that there are no constant functions and therefore the bias cannot grow. Using the discussion above, we will prove the security of our gadgets by showing that there exists no constant function $f(\overline{x},\cdot) \in \mathcal{F}^{(1)}(C)$ for all $\overline{x} \in \mathbb{F}_2^N$ and by calculating the corresponding bias bound of the gadgets. We start with the first-order algebraic security proof for a `RefreshMask`$[n, 1]$ gadget that uses the construction given in Section 4.2.1.

[2] Consider the function $g' = g \oplus 1$ which implies $wt(g') = 2^N - wt(g)$. The lower bound for $wt(g)$ implies the upper bound for $wt(g')$.

Proposition 4.5. *Let* $C(\overline{x}, \overline{r}) : \mathbb{F}_2^{n+2} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^{n+2}$ *be the circuit representation of the* `RefreshMask` *gadget using a masking scheme with an arbitrary order* n *and a fixed degree* $d = 1$*.* C *takes as input* $n+2$ *shares* $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \le i \le n})$ *and outputs* $n+2$ *shares* $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \le i \le n})$*. The gadget* `RefreshMask`$[n, 1]$ *is* ϵ*-1-AS with* $\epsilon := 1/4$*.*

Proof. In the first part of the proof, we show that, except of affine functions of inputs, there exists no function $f \in \mathcal{F}^{(1)}(C) \backslash \{\mathbf{0}, \mathbf{1}\}$ such that f is constant when inputs are fixed. Assume $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \le i \le n})$ are some fixed but arbitrary inputs and let $f \in \mathcal{F}^{(1)}(C) \backslash \{\mathbf{0}, \mathbf{1}\}$ be a function of random bits for the fixed input. As seen in Algorithm 9, all nodes used in `RefreshMask` gadget, behind the nodes for computing the values $\mathcal{W}$ and $\mathcal{R}$ in Line 11, are xor-gates. Thus, if f involves only those nodes it is either an affine function of inputs or it is non-constant.
Next, by the definition of $\mathcal{W}$ each input bit $\tilde{x}_0$ and $\tilde{x}_1$ in $\mathcal{W}$ is accompanied additively by a random value. And $\mathcal{R}$ contains only random values. To compute the expressions, behind xor-gates, there are used three and-gates which output the values:

- $\tilde{r}_0 \tilde{x}_1 \oplus \tilde{r}_0 r_0$,
- $\tilde{r}_1 \tilde{x}_0 \oplus \tilde{r}_1 r_0$, resp.
- $\tilde{r}_0 \tilde{r}_1 \oplus \tilde{r}_0 r_0 \oplus r_0 \tilde{r}_1 \oplus r_0 \tilde{r}_0$.

Thus, the only way to obtain a constant function f which involves the nodes used to compute the expression in Line 11, is to xor the first or the second value above with $\tilde{r}_0 r_0$, resp. with $\tilde{r}_1 r_0$. But, as one can easily check, no node used in Algorithm 9, computes the corresponding values. Hence we can conclude that there exists no constant function $f \in \mathcal{F}^{(1)}(C)$ such that inputs are fixed expect of affine functions of inputs.
In the second part, we examine the highest degree term in the gadget. The maximum degree term can be found in $\mathcal{R}$ with degree 2. Therefore the corresponding bias and the bias bound of the gadget can be calculated as $\epsilon \le 1/2 - 1/2^2 = 1/4$.
Thus the `RefreshMask` gadget is ϵ-1-AS with $\epsilon := 1/4$. □

We proceed with the first-order algebraic security proof for an `And`$[n, 1]$ gadget that uses the construction given in Section 4.2.1.

Proposition 4.6. *Let* $C((\overline{x}, \overline{y}), \overline{r}) : \mathbb{F}_2^{n+2} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^{n+2}$ *be the circuit representation of the* `And` *gadget using a masking scheme with an arbitrary order* n *and a fixed degree* $d = 1$*.* C *takes as input* $n+2$ *shares* $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \le i \le n})$ *and* $(\tilde{y}_0, \tilde{y}_1, (y_i)_{1 \le i \le n})$ *and outputs* $n+2$ *shares* $(\tilde{z}_0, \tilde{z}_1, (z_i)_{1 \le i \le n})$*. The gadget* `And`$[n, 1]$ *is* ϵ*-1-AS with* $\epsilon := 7/16$*.*

Proof. In the first part of the proof we show that, except of affine functions of the inputs, there exists no function $f \in \mathcal{F}^{(1)}(C)$ which is constant when inputs are fixed. From Proposition 4.5 and the fact that the refreshing and the main phase use different random bits, it follows that we need to consider functions involving only the nodes of the main part of the `And` gadget, i.e., the `RefreshMask` phase can be excluded.
First, let us reformulate the circuit C as follows:

$$C\colon ((\mathbb{F}_2^{n+2} \times \mathbb{F}_2^{n+2}), \mathbb{F}_2^{R_C}) \to \mathbb{F}_2^{n+2}$$
$$((\tilde{x}_0, \tilde{x}_1, (x_i)_{1\leq i\leq n}), (\tilde{y}_0, \tilde{y}_1, (y_i)_{1\leq i\leq n}), \bar{r}) \mapsto (\tilde{z}_0, \tilde{z}_1, (z_i)_{1\leq i\leq n}).$$

where $\bar{r}$ denotes the set of randomness that is used in the circuit. Next, we define three classes of edges within the circuit:

- R: The set of random bits,
- B: The set of linear shares, i.e. x_i and y_j for all $1 \leq i, j \leq n$,
- M: The set of non-linear shares, i.e. $\tilde{x}_0$, $\tilde{x}_1$, $\tilde{y}_0$ and $\tilde{y}_1$.

Using the above classification we can analyze the nodes $c_i \in C$ with respect to their input edges in the main part of the `And` gadget (i.e. starting in line 3 in Algorithm 8). We define the nodes as $c_i : (u_i^1, u_i^2) \mapsto v_i$ where $u_i^1,\ u_i^2 \in \mathbb{F}_2$ represent the input bits of the node and $v_i \in \mathbb{F}_2$ represents the output bit of the node. Depending on the edges of the nodes we classify them as follows

1. $u_i^1 \in \mathsf{R}$ or $u_i^2 \in \mathsf{R}$,
2. $u_i^1 \in \mathsf{B}$ or $u_i^2 \in \mathsf{B}$,
3. $u_i^1 \in \mathsf{M}$ and $u_i^2 \in \mathsf{M}$.

Each $c_i \in C$ is either one of the forms above or a combination of them e.g. $c_i : (v_j, v_k) \mapsto v_i$ or $c_i : (u_j^1, v_k) \mapsto v_i$ where $u_i^1 \in (\mathsf{R} \cup \mathsf{B} \cup \mathsf{M})$ and v_j, v_k are the output bits of the nodes c_j, c_k.
Assume that there exists a function $f \in \mathcal{F}^{(1)}(C)$ such that f is constant when the inputs $\overline{x}$ and $\overline{y}$ are fixed. Remark that for a fixed $\overline{c}_1, \overline{c}_2 \in \mathbb{F}_2^{n+2}$, the gadget calculates $\texttt{And}((\texttt{RefreshMask}(\overline{c}_1), \texttt{RefreshMask}(\overline{c}_2)), \overline{r})$, which equals to $\texttt{And}((\overline{c}'_1, \overline{c}'_2)), \overline{r})$ where $\overline{c}'_1$ and $\overline{c}'_2$ are the *non-fixed* outputs of $\texttt{RefreshMask}(\overline{c}_1)$ and $\texttt{RefreshMask}(\overline{c}_2)$, where by `And` we mean here the main phase of the gadget. Due to Proposition 4.5 we know that there exist no constant linear combinations of nodes inside `RefreshMask` with fixed

input. Thus, it is not possible to calculate a single bit input of the whole `And` gadget, since there exists no common node or no common random node between main part of the `And` gadget and the `RefreshMask` phase. Therefore, the only way of generating a constant function is to have a node that calculates $\tilde{x}_0\tilde{x}_1 \oplus x_1 \oplus \cdots \oplus x_n = x$ (resp. $\tilde{y}_0\tilde{y}_1 \oplus y_1 \oplus \cdots \oplus y_n$).
Any linear combination of the nodes of 1 and 2 cannot be constant due to the `RefreshMask` gadgets, since either a node is random (non-fixed by definition) or the node corresponds to linear masking (randomized by `RefreshMask`). Therefore, f should include at least one node from the 3^{rd} class or a combination of nodes that include multiplicative shares (M) to form the reconstructed multiplicative representation: x_0 or y_0. Clearly, such a combination can be found in **Step 1** and **Step 2(a)** where the following computations are processed:

- $\tilde{z}_0$ and $\tilde{z}_1$,
- $\tilde{x}_1(\tilde{x}_0 y_j \oplus r^{0,j}\tilde{y}_0) = \tilde{x}_1\tilde{x}_0 y_j \oplus r^{0,j}\tilde{x}_1\tilde{y}_0$ for $1 \leq j \leq n$,
- $\tilde{y}_1(\tilde{y}_0 x_j \oplus r^{1,j}\tilde{x}_0) = \tilde{y}_1\tilde{y}_0 x_j \oplus r^{1,j}\tilde{y}_1\tilde{x}_0$ for $1 \leq j \leq n$.

The use of parentheses indicates the order in which the nodes are used in the above equations. The resulting order eliminates the generation of an affine function of x_0 or y_0 (the shares represented by $\tilde{x}_0, \tilde{x}_1$ and $\tilde{y}_0, \tilde{y}_1$ respectively), although these nodes calculate the correct function ($r_{j,0}$ as seen in Equation (4.2)). Any linear combination of these nodes cannot be constant and thus there exists no constant function $f \in \mathcal{F}^{(1)}(C)$ when the inputs are fixed, besides the affine functions of input nodes.
In the second part, we examine the highest degree term in the gadget and find the corresponding bias. For `And`$[n, 1]$ the maximum degree term can be found in Line 16 of Algorithm 8. Specifically, $x_n y_n$ which contains a node of the form $\tilde{r}_0^x\tilde{r}_1^x\tilde{r}_0^y\tilde{r}_1^y$ where $\tilde{r}_0^x, \tilde{r}_1^x$ (resp. $\tilde{r}_0^y, \tilde{r}_1^y$) are the randomness used in `RefreshMask`$(\overline{x})$ (resp. `RefreshMask`$(\overline{y})$). Clearly the corresponding bias and the bias bound of the gadget can be calculated as 2^{-4} and $\epsilon \leq 1/2 - 1/2^4 = 7/16$ respectively. Thus the `And` gadget is ϵ-1-AS with $\epsilon := 7/16$.□

Although we are not giving a proof for the `Xor` gadget, the same discussion can be carried out and it can be shown that the `Xor`$[n, 1]$ gadget is ϵ-1-AS with $\epsilon := 1/4$. We provide experimental verification of the first-order gadgets, including the `Xor` gadget next.

Experimental Verification. To support the results, we provide experimental verification of the first-order gadgets `And`$[n, 1]$ and `Xor`$[n, 1]$ (and inherently `RefreshMask`$[n, 1]$) for

$n = 1, 2$ and 3 using the tool given in [BU18][3]. First we adapt our scheme to work with the tool, i.e. we implement our masking scheme (with the given orders n and d) as a class inside the tool. We then run the verification algorithm as explained above. The updated version of the tool including our scheme is available as open source [4].
We confirm the first-order algebraic security of our scheme for different orders of probing security. Details are shown in Table 4.3. The algorithm is run on an Intel Xeon Silver 4114 CPU@2.20GHz and, as shown in the table, the time that algorithm takes increases exponentially with the increasing number of nodes within the gadgets. The bias bound does not depend on the linear degree n, since the maximum degree term is found within the terms that depend on the non-linear degree d.

Table 4.3: First-order algebraic security verification of individual gadgets. Input corresponds to the number of shares for both inputs (i.e. $2(n+2)$). Random states the number of random values (R_C) within the circuit and it is calculated by the randomness requirement of two `RefreshMask` gadgets and additional randomness in the gadget. The number of intermediate variables represents the number of nodes in the gadget.

	Max degree	Bias Bound	Input	Random	Intermediate	Time
`Xor`$[1,1]$	2	1/4	6	6	8	3.5s
`And`$[1,1]$	4	7/16	6	6	12	4s.
`Xor`$[2,1]$	2	1/4	8	8	8	45.7s
`And`$[2,1]$	4	7/16	8	13	24	$\approx$ 114min
`Xor`$[3,1]$	2	1/4	10	10	8	$\approx$ 17min
`And`$[3,1]$	4	7/16	10	19	36	$\approx$ 5 days

4.3.3.2 Encoding-Circuit Composability

In the last part of the security analysis, we use the composability result of ϵ-1-AS given in [BU18]. Since the gadgets (`Xor`$[n,1]$, `And`$[n,1]$, `RefreshMask`$[n,1]$, as defined in Section 4.2.1) are ϵ-1-AS, an arbitrary combination of these gadgets is also ϵ-1-AS by Proposition 4 in [BU18]. Moreover, we observe that `Encode`$[n,1]$ is ϵ-1-AS with $\epsilon := 1/2^2$ according to Definition 4.4. Recall that for an $(n,1)$ scheme, the highest degree term is found in the last share: $x_n = x \oplus \tilde{x}_0\tilde{x}_1 \oplus \bigoplus_{i=1}^{n-1} x_i$ and clearly no linear combination of $(\tilde{x}_0, \tilde{x}_1, x_1, \ldots, x_n)$ is constant. Thus, for `Encode`$[n,1]$ $\forall f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and $\forall \overline{x} \in \mathbb{F}_2^N$ the bias of $f(\overline{x}, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ:

$$\max_{f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}, \overline{x} \in \mathbb{F}_2^N} \mathcal{E}(f(\overline{x}, \cdot)) \leq 1/4.$$

[3] https://github.com/cryptolu/whitebox
[4] https://github.com/UzL-ITS/white-box-masking

Finally, we can combine our construction with the encoding-circuit linear-composability result from [BU18].

Proposition 4.7. *Let $C(\overline{x}', \overline{r}_c) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit, and let $E(\overline{x}, \overline{r}_e) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be a function. If E is encoding ϵ-1-AS and C is circuit ϵ-1-AS then, for $d = 1$, it is true:*

$$\max_{f_{\overline{x}} \in \mathcal{R}_d} \mathcal{E}(f_{\overline{x}}) \leq \epsilon,$$

where $\mathcal{R}_d$ is defined in Definition 4.2.

4.3.4 Algebraic Security of the $(n, 2)$Masking Scheme

To prove the second-order prediction security (2-PS) of the $(n, 2)$-masking scheme we proceed, similarly as in Section 4.3.3, as follows. For our encoding function E and any Boolean circuit C constructed from gadgets as defined in Section 4.2.1 we will analyze $\epsilon = \max_{f_{\overline{x}} \in \mathcal{R}_2} \mathcal{E}(f_{\overline{x}})$ – the maximum bias over all functions in the set $\mathcal{R}_2$ defined in Definition 4.2, for $d = 2$. The main result of this section is that the maximum bias over all such functions is small. To obtain a bound on the prediction security of the $(n, 2)$-masking, we combine this bias with the upper bound on 2-PS from Proposition 4.1: $\mathrm{Adv}^{\mathrm{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{-\kappa}$, assuming $e = -\log_2(1/2 + \epsilon) > 0$ and the number of random bits of the circuit C is $\geq \kappa \cdot (1 + 1/e)$. To prove that for all $f_{\overline{x}}$ in $\mathcal{R}_2$ the bias $\mathcal{E}(f_{\overline{x}})$ is small, we use auxiliary definitions which extend the notion of the first-order algebraic security defined in [BU18] in a non-trivial way.

Definition 4.6 (Algebraic Encoding Security (ϵ-2-AS)). Let $E(\overline{x}, \overline{r}) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary encoding function. Let $\mathcal{Y}$ be the set of functions given by the output bits of E and let ϵ be a real number, with $1/4 \leq \epsilon < 1/2$. The function E is called second-order algebraically ϵ-secure (ϵ-2-AS) if, for $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$, it is true that:

1. $\forall f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and $\forall \overline{x} \in \mathbb{F}_2^N$ the bias of $f(\overline{x}, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ':
$$\max_{f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}, \overline{x} \in \mathbb{F}_2^N} \mathcal{E}(f(\overline{x}, \cdot)) \leq \epsilon'.$$
2. $\forall f \in \mathcal{Y}^{(2)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and $\forall \overline{x} \in \mathbb{F}_2^N$ the bias of $f(\overline{x}, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ:
$$\max_{f \in \mathcal{Y}^{(2)} \setminus \{\mathbf{0}, \mathbf{1}\}, \overline{x} \in \mathbb{F}_2^N} \mathcal{E}(f(\overline{x}, \cdot)) \leq \epsilon.$$

Definition 4.7 (Algebraic Circuit Security (ϵ-2-AS)). Let $C(\overline{x}, \overline{r}) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit and let ϵ be a real number, with $1/4 \leq \epsilon < 1/2$. Then C is called second-order algebraically ϵ-secure (ϵ-2-AS) if

1. C is ϵ'-1-AS, with $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$, and
2. for any function $f \in \mathcal{F}^{(2)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ one of the following conditions holds:
   (a) f has a form $f = g_0 \oplus \sum_{i=1}^{t} g_i h_i$, where $g_0, g_1, \ldots g_t$ are affine functions of $\overline{x}$ and $h_1, \ldots, h_t \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$,
   (b) for all $\overline{x} \in \mathbb{F}_2^N$ it holds: $\mathcal{E}(f(\overline{x}, \cdot)) \leq \epsilon$, where $f(\overline{x}, \cdot) : \mathbb{F}_2^{R_C} \to \mathbb{F}_2$.

The idea behind Definition 4.6 is quite intuitive: similarly as in Definition 4.4 we require that the bias for all functions of degree 2 over the output nodes of the encoding E, is bounded by ϵ. However, since a function $f_{\overline{x}}$ in $\mathcal{R}_2$ can be, e.g., a product of two functions g and h of degree one, such that g is over the output nodes of E and h is over the nodes of the circuit C, we need to require additionally that the bias of the 1st order functions is much smaller than ϵ to guarantee that the bias $\mathcal{E}(gh) \leq \epsilon$. As we will see, the value ϵ' as defined above, is chosen appropriately. Similarly, to guarantee that a composition of two ϵ-2-AS circuits remains ϵ-2-AS, we require in condition 1 in Definition 4.7 that the circuits are ϵ'-1-AS since, e.g., f can be a product of a function over the nodes of the first circuit and a function over the second circuits, both functions of degree one. From similar composability reasons, the condition 2.(a) in Definition 4.7 is needed.
The rest of this section is organised as follows. First, we estimate the values ϵ for ϵ-2-AS of the basic gadgets `RefreshMask`, `And`, and `Xor` (Section 4.3.4.1). Then we prove the composability result, i.e., that combining ϵ-2-AS circuits leads to the ϵ-2-AS composed circuit (Section 4.3.4.2). Finally, we show that our encoding scheme E is ϵ-2-AS, with $\epsilon := 31/64$, and that from the ϵ-2-AS property of C we get an estimation on $\max_{f_{\overline{x}} \in \mathcal{R}_d} \mathcal{E}(f_{\overline{x}})$, for $d = 2$ (Section 4.3.4.3).

4.3.4.1 ϵ-2-AS of the Gadgets

Using the new definition we will prove the second-order prediction security of our $(n, 2)$ basic gadgets. We first show that there exists no constant function $f(\overline{c}, \cdot) \in \mathcal{F}^{(2)}(C)$ for all $\overline{c} \in \mathbb{F}_2^N$. In the second step, we calculate the corresponding *first-order* and *second-order* bias bounds. We start with the ϵ-2-AS of the `RefreshMask`$[n, 2]$ gadget.

Proposition 4.8. *Let $C(\overline{x}, \overline{r}) : \mathbb{F}_2^{n+3} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^{n+3}$ be the circuit representation of the `RefreshMask` gadget using a masking scheme with an arbitrary order n and a fixed degree*

$d = 2$. C takes as input $n+3$ shares $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1\leq i\leq n})$ and outputs $n+3$ shares $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1\leq i\leq n})$. The gadget `RefreshMask`$[n, 2]$ *is ϵ-2-AS with $\epsilon := 31/64$.*

Proof. First let us consider a function $f \in \mathcal{F}^{(2)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$. As before we denote the function f with a constant input $\overline{c} \in \mathbb{F}_2^{n+3}$ as $f_{\overline{c}}(\overline{r}_1)$. If $f_{\overline{c}}$ can be written of the form $f_{\overline{c}} = g_0 \oplus \sum_{i=1}^{t} g_i h_i$, where $g_0, g_1, \ldots, g_t$ are affine functions of $\overline{c}$ and $h_1, \ldots h_t \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ then there exists $\overline{c} \in \mathbb{F}_2^{n+3}$ such that $f_{\overline{c}}$ is constant. However, in this case $f_{\overline{c}}$ satisfies condition 2.(a).
Assume $f_{\overline{c}}$ does not reside in condition 2.(a). As seen in Algorithm 9, all nodes, except of those used to compute the values $\mathcal{W}$ and $\mathcal{R}$ in Line 11, are xor-gates and all of them contain as input a random variable (i.e. not fixed). Next, due to the computational structure of $\mathcal{W}$ and $\mathcal{R}$ given in `RefreshMask`$[n, 2]$, the input nodes $\tilde{x}_0, \tilde{x}_1$, and $\tilde{x}_2$ used in the expressions are accompanied by a random value. Note that over the nodes used to compute the expression $x_n \oplus \mathcal{W} \oplus \mathcal{R}$ in Line 11, one can construct a constant function f of degree 2, but this yields a function satisfying condition 2.(a). In particular, to compute the term $[\tilde{r}_2(\tilde{x}_0 \oplus r_0)][\tilde{r}_1 \oplus (\tilde{x}_1 \oplus r_0)]$ in $\mathcal{W}$ we use five nodes:

- $u_1 = \tilde{x}_0 \oplus r_0$,
- $u_2 = u_1\tilde{r}_2 = \tilde{r}_2\tilde{x}_0 \oplus \tilde{r}_2 r_0$,
- $u_3 = \tilde{x}_1 \oplus r_0$,
- $u_4 = u_3 \oplus \tilde{r}_1 = (\tilde{x}_1 \oplus r_0 \oplus \tilde{r}_1)$,
- $u_5 = u_4 u_2 = \tilde{x}_0\tilde{r}_2 r_0 \oplus \tilde{x}_1\tilde{r}_2 r_0 \oplus \tilde{x}_0\tilde{x}_1\tilde{r}_2 \oplus \tilde{x}_0\tilde{r}_1\tilde{r}_2 \oplus \tilde{r}_2\tilde{r}_0 \oplus \tilde{r}_1\tilde{r}_2 r_0$.

Then, the function f over the nodes $u_1, \ldots, u_5$ is either non-constant, an affine function of inputs, or has the form as in condition 2.(a). For example, $f = u_2 \oplus \tilde{r}_2 r_0 = \tilde{r}_2\tilde{x}_0 \oplus \tilde{r}_2 r_0 \oplus \tilde{r}_2 r_0 = \tilde{r}_2\tilde{x}_0$ is constant, if $\tilde{x}_0 = 0$, but it satisfies the condition 2.(a). Moreover, observe that one cannot obtain a constant function using u_5 due to the term $\tilde{r}_1\tilde{r}_2 r_0$ which cannot be removed by a third-order combination of the nodes. The same arguments can be used for functions over the nodes for computing the remaining terms of $\mathcal{W}$, i.e., $[\tilde{r}_1(\tilde{x}_2 \oplus r_0)][\tilde{r}_0 \oplus (\tilde{x}_0 \oplus r_0)]$ and $[\tilde{r}_0(\tilde{x}_1 \oplus r_0)][\tilde{r}_2 \oplus (\tilde{x}_2 \oplus r_0)]$.
Finally, each node contains only one non-linear share, thus any first or second-order combination cannot contain all three non-linear shares such that the variable $\tilde{x}_0\tilde{x}_1\tilde{x}_2$ is formed without a random value accompanying it. For example, a second order combination of the nodes $u_5 = [\tilde{r}_2(\tilde{x}_0 \oplus r_0)][\tilde{r}_1 \oplus (\tilde{x}_1 \oplus r_0)]$ and $u_5' = [\tilde{r}_1(\tilde{x}_2 \oplus r_0)][\tilde{r}_0 \oplus (\tilde{x}_0 \oplus r_0)]$ equals to: $u_5 u_5' = \cdots \oplus \tilde{r}_0\tilde{r}_1\tilde{r}_2 r_0$ and we cannot remove $\tilde{r}_0\tilde{r}_1\tilde{r}_2 r_0$, thus the combination cannot be constant. Hence $f_{\overline{c}}$ cannot be a constant for all $\overline{c} \in \mathbb{F}_2^{n+3}$.

In the second part of the proof we analyze value m such that $m = \max(\deg(c_i)_{c_i \in C})$. Observe that the highest degree term in the gadget can be found in $\mathcal{R}$ with degree 3. Thus the linear bias bound of the gadget can be seen as follows:

$$\epsilon' = \mathcal{E}(f') \leq \frac{1}{2} - \frac{1}{2^3} \text{ where } f' \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}.$$

This result implies that $\texttt{RefreshMask}[n, 2]$ is ϵ'-1-AS gadget. Moreover, the highest degree term of $f \in \mathcal{F}^{(2)}(C)$ is less than or equal to 6 which implies:

$$\epsilon = \mathcal{E}(f) \leq \frac{1}{2} - \frac{1}{2^6} \text{ where } f \in \mathcal{F}^{(2)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}.$$

Observe that, in the first part of the proof we showed that there exists no function $f \in \mathcal{F}^{(2)}(C)$ such that $f(\overline{c}, \cdot)$ is constant (expect of functions of the form condition 2.(a)), which implies both linear and second-order biases cannot grow. Thus the `RefreshMask` gadget is ϵ-2-AS with $\epsilon := 31/64$. □

Next, we prove the second-order algebraic security of the $\texttt{And}[n, 2]$ gadget.

Proposition 4.9. *Let $C((\overline{x}, \overline{y}), \overline{r}) : \mathbb{F}_2^{n+3} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^{n+3}$ be the circuit representation of the `And` gadget using a masking scheme with an arbitrary order n and a fixed degree $d = 2$. C takes as input $n + 3$ shares $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1 \leq i \leq n})$, $(\tilde{y}_0, \tilde{y}_1, \tilde{y}_2, (y_i)_{1 \leq i \leq n})$ and outputs $n+3$ shares $(\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, (z_i)_{1 \leq i \leq n})$. The gadget $\texttt{And}[n, 2]$ is ϵ-2-AS with $\epsilon := (1/2 - 1/2^{12})$.*

Proof. Similar to the proof of Proposition 4.6, we reformulate the circuit C as follows:

$$C \colon ((\mathbb{F}_2^{n+3} \times \mathbb{F}_2^{n+3}), \mathbb{F}_2^{R_C}) \to \mathbb{F}_2^{n+3}$$
$$((\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1 \leq i \leq n}), (\tilde{y}_0, \tilde{y}_1, \tilde{y}_2, (y_i)_{1 \leq i \leq n}), \bar{r}) \mapsto (\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, (z_i)_{1 \leq i \leq n}).$$

Next we use the classification of the nodes that we used in the proof of Proposition 4.6:

- R: The set of random bits,
- B: The set of linear shares, i.e. x_i and y_j for all $1 \leq i, j \leq n$,
- M: The set of non-linear shares, i.e. $\tilde{x}_0$, $\tilde{x}_1$, $\tilde{x}_2$, $\tilde{y}_0$, $\tilde{y}_1$, and $\tilde{y}_2$.

Using the above classification we can analyze the nodes $c_i \in C$ with respect to its input edges. We define the nodes as $c_i : (u_i^1, u_i^2) \mapsto v_i$ where $u_i^1,\ u_i^2 \in \mathbb{F}_2$ represent the input bits of the node and $v_i \in \mathbb{F}_2$ represents the output bit of the node. The base classification of the depending nodes is as follows, (1) $u_i^1 \in \mathsf{R}$ or $u_i^2 \in \mathsf{R}$, (2) $u_i^1 \in \mathsf{B}$ or $u_i^2 \in \mathsf{B}$, (3)

$u_i^1 \in \mathsf{M}$ and $u_i^2 \in \mathsf{M}$. Each $c_i \in C$ is either one of the forms above or a combination of them, e.g., $c_i : (v_j, v_k) \mapsto v_i$ or $c_i : (u_j^1, v_k) \mapsto v_i$ where $u_i^1 \in (\mathsf{R} \cup \mathsf{B} \cup \mathsf{M})$ and v_j, v_k are the output bits of the nodes c_j, c_k.

First let us consider a function $f \in \mathcal{F}^{(2)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$. As before we denote the function f with a constant inputs $\overline{c}_1, \overline{c}_2 \in \mathbb{F}_2^{n+3}$ as $f_{\overline{c}_1,\overline{c}_2}(\overline{r}_1)$. If $f_{\overline{c}_1,\overline{c}_2}$ can be written of the form $f_{\overline{c}_1,\overline{c}_2} = g_0 \oplus \sum_{i=1}^{t} g_i h_i$, where $g_0, g_1, \dots g_t$ are affine functions of $\overline{c}_1, \overline{c}_2$ and $h_1, \dots h_t \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$, then there exists $\overline{c}_1, \overline{c}_2 \in \mathbb{F}_2^{n+3}$ such that $f_{\overline{c}_1,\overline{c}_2}$ is constant and $f_{\overline{c}_1,\overline{c}_2}$ lies in condition 2.(a) of Definition 4.7.

Assume $f_{\overline{c}_1,\overline{c}_2}$ does not reside in condition 2.(a) of Definition 4.7. As in the proof of Proposition 4.6 we can state that for a fixed $\overline{c}_1, \overline{c}_2 \in \mathbb{F}_2^{n+3}$, the gadget calculates $\texttt{And}((\texttt{RefreshMask}(\overline{c}_1), \texttt{RefreshMask}(\overline{c}_2)), \overline{r})$, which equals to $\texttt{And}((\overline{c}_1', \overline{c}_2')), \overline{r})$ where $\overline{c}_1'$ and $\overline{c}_2'$ are the *non-fixed* outputs of $\texttt{RefreshMask}(\overline{c}_1)$ and $\texttt{RefreshMask}(\overline{c}_2)$; Here, by `And` we mean the main phase of the gadget. Due to Proposition 4.8 we know that there exists no constant second-order combinations of nodes inside `RefreshMask` with fixed input. Thus it is not possible to calculate a single bit input of the whole `And` gadget, since there exists no common node or no common random node between the main part of the `And` gadget and the `RefreshMask` phase. Therefore, the only way of generating a constant function is to have a node that calculates $\tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus x_1 \oplus \cdots \oplus x_n = x$ (resp. $\tilde{y}_0\tilde{y}_1\tilde{y}_2 \oplus y_1 \oplus \cdots \oplus y_n$).

This observation indicates that f should include nodes from the third class or a combination of nodes that include multiplicative shares (M) which can be found in **Step 1** and **Step 2(a)**. However, the nodes $c_i \in C$ contain at most one value from each multiplicative representation, i.e., each node contains only one non-linear share. Thus, any first or second-order combination cannot contain all three non-linear shares and $f(\overline{x}, \overline{y}, \cdot)$ cannot be fixed for all $\overline{x}, \overline{y} \in \mathbb{F}_2^{n+3}$.

In the second part of the proof we examine the highest degree term in the circuit. The maximum degree term can be found in Line 16 of Algorithm 8 for $\texttt{And}[n, 2]$. We can see that the maximum degree term for this case is 6. The linear bias bound of the gadget is:

$$\mathcal{E}(f') < \epsilon' := \frac{1}{2} - \frac{1}{2^6} \text{ , where } f' \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$$

Thus, $\texttt{And}[n, 2]$ is ϵ'-1-AS. Using the same argument, we can see that the maximum degree term $f \in \mathcal{F}^{(2)}(C)$ is less than or equal to 12. This result is followed by:

$$\mathcal{E}(f) \leq \epsilon := \frac{1}{2} - \frac{1}{2^{12}} \text{ , where } f \in \mathcal{F}^{(2)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}.$$

Observe that in the first part of the proof we showed that there exists no function $f \in \mathcal{F}^{(2)}(C)$ such that $f_{\bar{c}_1,\bar{c}_2}$ is constant, which implies both linear and second-order biases cannot grow. Thus, the $\texttt{And}[n, 2]$ gadget is ϵ-2-AS circuit where $\epsilon := 1/2 - 1/2^{12}$.□

Using the same idea we can prove that the $\texttt{Xor}[n, 2]$ gadget is a ϵ-2-AS circuit with $\epsilon := 1/2 - 1/2^6$.

4.3.4.2 Circuit Composability

In the previous subsection, we have shown that our basic gadgets `RefreshMask`, `And`, and `Xor` are ϵ-2-AS, for some specific values ϵ. Now, we prove that any circuit obtained by the composition of such gadgets remains ϵ-2-AS. To cover all cases, we consider separately the *parallel* (Proposition 4.10) and the *sequential* (Proposition 4.11) composability of two circuits. In particular, from Proposition 4.10 we can deduce, that if one applies, e.g., $\texttt{And}(\overline{x}, \overline{y})$ for inputs $\overline{x} = \texttt{And}(\overline{x}_1, \overline{y}_1)$ and $\overline{y} = \texttt{Xor}(\overline{x}_2, \overline{y}_2)$ then, from Proposition 4.10, the parallel composition:

$$\texttt{And}(\overline{x}_1, \overline{y}_1); \texttt{Xor}(\overline{x}_2, \overline{y}_2)$$

with input $\overline{x}_1, \overline{y}_1, \overline{x}_2, \overline{y}_2$ and output $\overline{x}, \overline{y}$ is an ϵ-2-AS circuit. Moreover, due to Proposition 4.11, we get that

$$\texttt{And}(\texttt{And}(\overline{x}_1, \overline{y}_1), \texttt{Xor}(\overline{x}_2, \overline{y}_2))$$

remains ϵ-2-AS.

Proposition 4.10 (ϵ-2-AS-Circuit-Parallel-Composability). *Assume $C_1(\overline{x}_1, \overline{r}_1)$ and $C_2(\overline{x}_2, \overline{r}_2)$ are two (disjoint [5]) ϵ-2-AS circuits. Let C be the circuit obtained by parallel composition of C_1 and C_2, i.e. by considering the input of C_1 and the input of C_2 as the input of C and, analogously, the output of C_1 and the output of C_2 as the output of C. Moreover let $\overline{r}_1$ and $\overline{r}_2$ be the extra random input of C:*

$$C(\overline{x}_1, \overline{x}_2, (\overline{r}_1, \overline{r}_2)) = (C_1(\overline{x}_1, \overline{r}_1), C(\overline{x}_2, \overline{r}_2)).$$

Then $C(\overline{x}_1, \overline{x}_2, (\overline{r}_1, \overline{r}_2))$ is also an ϵ-2-AS circuit.

Proof. Assume C_1 and C_2 are ϵ-2-AS and let $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$. From [BU18] we know that the composition C is ϵ'-1-AS. Thus, C satisfies the condition 1 in Definition 4.7.

[5] Remark that, the two disjoint circuits implies that the *random nodes* of the circuits C_1 and C_2 are disjoint. Particularly, C_1 and C_2 can have the same input.

To see that the condition 2 is true as well, let us consider a function $f(\overline{x}_1, \overline{x}_2, (\overline{r}_1, \overline{r}_2)) \in \mathcal{F}^{(2)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$. If f has a form as defined in condition 2.(a), i.e., $f = g_0 \oplus \sum_{i=1}^{t} g_i h_i$, where $g_0, g_1, \ldots g_t$ are affine functions of $\overline{x}_1$ and $\overline{x}_2$ and $h_1, \ldots h_t \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$, then we are done. So, let as assume this is not the case. We will show that for any $\overline{x}_1, \overline{x}_2$, the bias is bounded as follows

$$\mathcal{E}(f(\overline{x}_1, \overline{x}_2, \cdot, \cdot)) \leq \epsilon.$$

Assume $\overline{x}_1$ and $\overline{x}_2$ are arbitrary, but fixed inputs. To simplify the notation, let $\tilde{f}$ denote the function $\tilde{f}(\overline{r}_1, \overline{r}_2) = f(\overline{x}_1, \overline{x}_2, (\overline{r}_1, \overline{r}_2))$.

For example, if x denotes the first element of the input vector $\overline{x}_1$, x' – the first element of $\overline{x}_2$, bits r, r' – the first two elements of random vector $\overline{r}_1$, and w is an internal node in C_2 then, for $\tilde{f}(\overline{r}_1, \overline{r}_2) = xr \oplus x' f_w \oplus rr'$ we have $\tilde{f}(\overline{r}_1, \overline{r}_2) = rr'$, if $x = x' = 0$, $\tilde{f}(\overline{r}_1, \overline{r}_2) = f_w \oplus rr'$, if $x = 0$ and $x' = 1$, etc.

In the most general case, $\tilde{f}$ has the form

$$\tilde{f}(\overline{r}_1, \overline{r}_2) \;=\; c \;\oplus\; u_2(\overline{r}_1) \;\oplus\; v_2(\overline{r}_2) \;\oplus\; \sum_{i=1}^{\tau} u_1^i(\overline{r}_1)\; v_1^i(\overline{r}_2), \tag{4.5}$$

where $c \in \{\mathbf{0}, \mathbf{1}\}$ is a constant and u_2, v_2, u_1^i, and v_1^i are functions such that:

$$\begin{aligned} &u_2 \in \mathcal{F}^{(2)}(C_1) \setminus \{\mathbf{0}, \mathbf{1}\}, \quad v_2 \in \mathcal{F}^{(2)}(C_2) \setminus \{\mathbf{0}, \mathbf{1}\}, \text{ and} \\ &u_1^i \in \mathcal{F}^{(1)}(C_1) \setminus \{\mathbf{0}, \mathbf{1}\}, \quad v_1^i \in \mathcal{F}^{(1)}(C_2) \setminus \{\mathbf{0}, \mathbf{1}\} \text{ for } i = 1, \ldots \tau. \end{aligned}$$

To prove that the bias of $\tilde{f}$ is bounded by ϵ, we consider two cases.

Case 1: Function u_2 or v_2 in Eq. (4.5) is non-trivial, i.e., that u_2 or v_2 contains at least one term. Let, w.l.o.g., v_2 be non-trivial. For every fixed (but arbitrary) $\overline{r}_1$, function (4.5) can be represented as a function in $\mathcal{F}^{(2)}(C_2)$ as follows:

$$\tilde{f}_{\overline{r}_1}(\overline{r}_2) \;=\; c \;\oplus\; c_0 \;\oplus\; v_2(\overline{r}_2) \;\oplus\; \sum_{i=1}^{\tau} c_i\; v_1^i(\overline{r}_2),$$

where $c_0 = v_2(\overline{r}_1)$ and, in case $\tau \geq 1$, $c_i = v_1^i(\overline{r}_1)$, for $i = 1, \ldots, \tau$. By assumption that C_2 is ϵ-2-AS, from condition 2.(b) of Definition 4.7, we get that $\mathcal{E}(\tilde{f}_{\overline{r}_1}(\overline{r}_2)) \leq \epsilon$ and since this bound is true for every $\overline{r}_1$, we can conclude that the function $\tilde{f}$, as defined in (4.5), has the bias bounded by ϵ, too.

Case 2: Function $\tilde{f}$ in Eq. (4.5) has the form $\tilde{f}(\overline{r}_1, \overline{r}_2) \;=\; c \oplus \sum_{i=1}^{\tau} u_1^i(\overline{r}_1)\; v_1^i(\overline{r}_2)$, with $\tau \geq 1$. Now, for every fixed $\overline{r}_1$, function (4.5) can be represented as a function in

$\mathcal{F}^{(1)}(C_2)$ as follows:

$$\tilde{f}_{\overline{r}_1}(\overline{r}_2) \;=\; c \;\oplus\; \sum_{i=1}^{\tau} c_i \; v_1^i(\overline{r}_2),$$

where $c_i = u_1^i(\overline{r}_1)$, for $i = 1, \ldots, \tau$. If for every $\overline{r}_1$ it would be true that some $c_i \neq 0$, then we could deduce immediately that the bias of $\tilde{f}$ is bounded by $\epsilon' \leq \epsilon$. Unfortunately, it can happen that for some vectors $\overline{r}_1$, all coefficients c_i vanish implying that $\tilde{f}_{\overline{r}_1}(\cdot)$ has bias 1/2. Below, we argue that there are sufficiently many values for $\overline{r}_1$ such that at least one coefficient c_i is nonzero. In consequence we will be able to bound the bias of $\tilde{f}$ in this case.

Consider the coefficient c_1. Recall that it is defined as a function $c_1 = u_1^1(\overline{r}_1)$ in $\mathcal{F}^{(1)}(C_1)$. From the assumption, the bias of $u_1^1(\cdot)$ is bounded as follows

$$\mathcal{E}(u_1^1(\overline{r}_1)) = |1/2 - wt(u_1^1)/2^{|\overline{r}_1|}| \leq \epsilon'.$$

From this inequality, one can deduce that the number $wt(u_1^1)$ of values for $\overline{r}_1$, for which $c_1 = u_1^1(\overline{r}_1) = 1$, is at least $wt(u_1^1) \geq 2^{|\overline{r}_1|}\,(1/2 - \epsilon')$. Let, for short, $R_1 := \{\overline{r}_1 \mid c_1 = u_1^1(\overline{r}_1) = 1\}$ denote the set of all such vectors $\overline{r}_1$. Its cardinality is

$$|R_1| \geq 2^{|\overline{r}_1|}\left(1/2 - \epsilon'\right).$$

Now, we consider the functions $\tilde{f}_{\overline{r}_1}(\cdot)$ restricting $\overline{r}_1$ to random strings from R_1, i.e. we consider

$$\tilde{f}_{\overline{r}_1}(\overline{r}_2) \;=\; c \;\oplus\; \sum_{i=1}^{\tau} c_i \; v_1^i(\overline{r}_2), \text{ with } \overline{r}_1 \in R_1.$$

From the assumptions we know that every such $\tilde{f}_{\overline{r}_1}$ has bias bounded by ϵ':

$$\mathcal{E}(\tilde{f}_{\overline{r}_1}) = |1/2 - wt(\tilde{f}_{\overline{r}_1})/2^{|\overline{r}_2|}| \leq \epsilon', \quad \text{for all } \overline{r}_1 \in R_1.$$

This means that for every $\overline{r}_1 \in R_1$ it is true:

$$2^{|\overline{r}_2|}\left(1/2 - \epsilon'\right) \;\leq\; wt(\tilde{f}_{\overline{r}_1}) \;\leq\; 2^{|\overline{r}_2|}\left(1/2 + \epsilon'\right).$$

Combining this inequality with the bound on $|R_1|$, one can conclude that

$$2^{|\overline{r}_1|+|\overline{r}_2|}\left(1/2 - \epsilon'\right)^2 \;\leq\; |\{(\overline{r}_1, \overline{r}_2) \mid \tilde{f}(\overline{r}_1, \overline{r}_2) = 1\}| \;\leq\; 2^{|\overline{r}_1|+|\overline{r}_2|}\left(1/2 + \epsilon'\right).$$

Now, using our definition for $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$, the left-hand side can be written as

$2^{|\overline{r}_1|+|\overline{r}_2|}\,(1/2-\epsilon)$ and $(1/2+\epsilon')$ on the right-hand side can be bounded by $(1/2+\epsilon)$. Thus we get

$$2^{|\overline{r}_1|+|\overline{r}_2|}\,(1/2-\epsilon) \;\leq\; |\{(\overline{r}_1,\overline{r}_2) \mid \tilde{f}(\overline{r}_1,\overline{r}_2)=1\}| \;\leq\; 2^{|\overline{r}_1|+|\overline{r}_2|}\,(1/2+\epsilon)\,.$$

This completes the proof that in Case 2 the bias bound $\mathcal{E}(\tilde{f}) \leq \epsilon$ holds. □

Proposition 4.11 (ϵ-2-AS-Circuit-Sequential-Composability). *Consider ϵ-2-AS circuits $C_1(\overline{x}_1,\overline{r}_1)$ and $C_2(\overline{x}_2,\overline{r}_2)$. Let C be the circuit obtained by connecting the output of C_1 to the input $\overline{x}_2$ of C_2 and letting the input $\overline{r}_2$ of C_2 be the extra input of C:*

$$C(\overline{x}_1,(\overline{r}_1,\overline{r}_2)) = C_2(C_1(\overline{x}_1,\overline{r}_1),\overline{r}_2).$$

Then $C(\overline{x}_1,(\overline{r}_1,\overline{r}_2))$ is also an ϵ-2-AS circuit.

Proof. We will proceed analogously to the proof of Proposition 4.10. Assume C_1 and C_2 are ϵ-2-AS and let $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2}-\epsilon}$. Due to [BU18] the composition C is ϵ'-1-AS and thus, the first condition in Definition 4.7 is satisfied.

To see that also the second condition is true, let us consider a function $f(\overline{x}_1,\overline{r}_1,\overline{r}_2) \in \mathcal{F}^{(2)}(C)\setminus\{\mathbf{0},\mathbf{1}\}$. If f has a form as defined in condition 2.(a), i.e., $f = g_0 \oplus \sum_{i=1}^{t} g_i h_i$, where $g_0, g_1, \ldots g_t$ are affine functions of $\overline{x}_1$ and $h_1, \ldots h_t \in \mathcal{F}^{(1)}(C)\setminus\{\mathbf{0},\mathbf{1}\}$, then we are done. If this is not the case, we need to show that for any $\overline{x}_1$ the bias

$$\mathcal{E}(f(\overline{x}_1,\cdot,\cdot)) \leq \epsilon.$$

Before we start our analysis, note that in our construction, the input nodes of C_2 coincide with the output nodes of C_1. Now assume $\overline{x}_1$ is an arbitrary, but fixed input vector and let $\tilde{f}$ denote the function $\tilde{f}(\overline{r}_1,\overline{r}_2) = f(\overline{x}_1,\overline{r}_1,\overline{r}_2)$. In the most general case it has the form

$$\tilde{f}(\overline{r}_1,\overline{r}_2) \;=\; c \;\oplus\; u_2(\overline{r}_1) \;\oplus\; v_2(y(\overline{r}_1),\overline{r}_2) \;\oplus\; \sum_{i=1}^{\tau} u_1^i(\overline{r}_1)\; v_1^i(y(\overline{r}_1),\overline{r}_2), \tag{4.6}$$

where $c \in \{\mathbf{0},\mathbf{1}\}$ is a constant, $y(\overline{r}_1) = C_1(\overline{x}_1,\overline{r}_1)$ denotes the output of C_1 on $(\overline{x}_1,\overline{r}_1)$, and u_2, v_2, u_1^i, and v_1^i are functions such that:

$$\begin{aligned} &u_2 \in \mathcal{F}^{(2)}(C_1)\setminus\{\mathbf{0},\mathbf{1}\}, \quad v_2 \in \mathcal{F}^{(2)}(C_2)\setminus\{\mathbf{0},\mathbf{1}\}, \text{ and} \\ &u_1^i \in \mathcal{F}^{(1)}(C_1)\setminus\{\mathbf{0},\mathbf{1}\}, \quad v_1^i \in \mathcal{F}^{(1)}(C_2)\setminus\{\mathbf{0},\mathbf{1}\}, \text{ for } i=1,\ldots\tau. \end{aligned}$$

To prove that the bias of $\tilde{f}$ is bounded by ϵ, we consider three cases.

Case 1: Function v_2 in Eq. (4.6) is non-trivial, i.e., assume that v_2 has at least one term. For every fixed (but arbitrary) $\overline{r}_1$, function $\tilde{f}$ can be expressed as a function in $\mathcal{F}^{(2)}(C_2)$ as follows:

$$\tilde{f}_{\overline{r}_1}(\overline{r}_2) \;=\; c \;\oplus\; c_0 \;\oplus\; v_2(\hat{y}, \overline{r}_2) \;\oplus\; \sum_{i=1}^{\tau} c_i \; v_1^i(\hat{y}, \overline{r}_2),$$

where $\hat{y} := y(\overline{r}_1) = C_1(\overline{x}_1, \overline{r}_1)$, $c_0 := u_2(\overline{r}_1)$, and $c_i := u_1^i(\overline{r}_1)$, for $i = 1, \ldots, \tau$ (note, that in this case τ can be 0 or for $\tau \geq 1$, all coefficient values c_i can be 0). We may assume that $\tilde{f}$ does not involve input nodes $\overline{x}_2$ of C_2; Indeed if the input nodes are used then we represent them as the corresponding output nodes of C_1. This means that, in particular, we may assume that v_2 is not of the form $g_0 \oplus \sum_{i=1}^{t} g_i h_i$, where $g_0, g_1, \ldots g_t$ are affine functions of $\overline{x}_2 = \hat{y}$ and $h_1, \ldots h_t \in \mathcal{F}^{(1)}(C_2) \setminus \{\mathbf{0}, \mathbf{1}\}$; If v_2 has such a form, then one can consider g_0 as a function in $\mathcal{F}^{(2)}(C_1) \setminus \{\mathbf{0}, \mathbf{1}\}$ and $g_1, \ldots, g_t$ as functions in $\mathcal{F}^{(1)}(C_1) \setminus \{\mathbf{0}, \mathbf{1}\}$ and, as a consequence, v_2 would vanish.

Thus, by the assumption that C_2 is ϵ-2-AS, we get that $\mathcal{E}(\tilde{f}_{\overline{r}_1}(\overline{r}_2)) \leq \epsilon$ and since this bound is true for every $\overline{r}_1$, we can conclude that the function $\tilde{f}$, as defined in (4.6), has the bias bounded by ϵ, too.

Case 2: $\tilde{f}$ in Eq. (4.6) has the form $\tilde{f}(\overline{r}_1, \overline{r}_2) \;=\; c \oplus u_2(\overline{r}_1) \oplus \sum_{i=1}^{\tau} u_1^i(\overline{r}_1)\; v_1^i(y(\overline{r}_1), \overline{r}_2)$, with $\tau \geq 1$. In this case, for every fixed $\overline{r}_1$, we can represent $\tilde{f}$ a function in $\mathcal{F}^{(1)}(C_2)$ as follows:

$$\tilde{f}_{\overline{r}_1}(\overline{r}_2) \;=\; c \;\oplus\; c_0 \;\oplus\; \sum_{i=1}^{\tau} c_i \; v_1^i(\hat{y}, \overline{r}_2),$$

where, as in Case 1, $\hat{y} := C_1(\overline{x}_1, \overline{r}_1)$, $c_0 := u_2(\overline{r}_1)$, and $c_i := u_1^i(\overline{r}_1)$, for $i = 1, \ldots, \tau \geq 1$. Note, that for some strings $\overline{r}_1$ it can happen that all coefficients $c_i = 0$, what means that for such $\overline{r}_1$ function $\tilde{f}_{\overline{r}_1}$ has bias $1/2$. Below, we argue that there are sufficiently many $\overline{r}_1$ such that at least one coefficient c_i is non-zero. This will suffice to bound the bias of $\tilde{f}$.

We consider the coefficient c_1 that, recall, is defined as $c_1 := u_1^1(\overline{r}_1)$ for the function $u_1^1(\cdot)$ in $\mathcal{F}^{(1)}(C_1)$. From the assumption, its bias is bounded as follows

$$\mathcal{E}(u_1^1(\overline{r}_1)) = |1/2 - wt(u_1^1)/2^{|\overline{r}_1|}| \leq \epsilon'.$$

It follows that $wt(u_1^1) \geq 2^{|\overline{r}_1|}\,(1/2 - \epsilon')$. Let $R_1 := \{\overline{r}_1 \mid c_1 = u_1^1(\overline{r}_1) = 1\}$ denote the set of all such vectors $\overline{r}_1$. Its cardinality is at least $2^{|\overline{r}_1|}\,(1/2 - \epsilon')$. Consider $\tilde{f}_{\overline{r}_1}(\cdot)$ restricting

$\overline{r}_1$ to vectors from R_1 only:

$$\tilde{f}_{\overline{r}_1}(\overline{r}_2) \;=\; c \;\oplus\; \sum_{i=1}^{\tau} c_i \; v_1^i(\overline{r}_2), \text{ with } \overline{r}_1 \in R_1.$$

From the assumptions, we know that every such $\tilde{f}_{\overline{r}_1}$ has its bias bounded by ϵ':

$$\mathcal{E}(\tilde{f}_{\overline{r}_1}) = |1/2 - wt(\tilde{f}_{\overline{r}_1})/2^{|\overline{r}_2|}| \leq \epsilon', \quad \text{for all } \overline{r}_1 \in R_1.$$

This means that: $2^{|\overline{r}_2|}\,(1/2-\epsilon') \;\leq\; wt(\tilde{f}_{\overline{r}_1}) \;\leq\; 2^{|\overline{r}_2|}\,(1/2+\epsilon')$, for all $\overline{r}_1 \in R_1$, and combining this with the bound on $|R_1|$, we get

$$2^{|\overline{r}_1|+|\overline{r}_2|}\left(1/2-\epsilon'\right)^2 \;\leq\; |\{(\overline{r}_1,\overline{r}_2) \mid \tilde{f}(\overline{r}_1,\overline{r}_2)=1\}| \;\leq\; 2^{|\overline{r}_1|+|\overline{r}_2|}\left(1/2+\epsilon'\right).$$

Using our definition for $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2}-\epsilon}$ we can conclude that

$$2^{|\overline{r}_1|+|\overline{r}_2|}\,(1/2-\epsilon) \;\leq\; |\{(\overline{r}_1,\overline{r}_2) \mid \tilde{f}(\overline{r}_1,\overline{r}_2)=1\}| \;\leq\; 2^{|\overline{r}_1|+|\overline{r}_2|}\,(1/2+\epsilon).$$

This completes the proof that in Case 2 the bias $\mathcal{E}(\tilde{f}) \leq \epsilon$, too.

Case 3: Function $\tilde{f}$ in Eq. (4.6) has the form $\tilde{f}(\overline{r}_1,\overline{r}_2) \;=\; c \;\oplus\; u_2(\overline{r}_1)$. In this case the bound on the bias of $\tilde{f}$ follows directly from the assumption that C_1 is ϵ-2-AS. □

4.3.4.3 Encoding-Circuit Composability

Finally, we prove that a composition of an ϵ-2-AS encoding function E with an ϵ-2-AS circuit C leads to a construction for which the second-order closure of $\mathcal{F}(C(E))$ contains functions of bias $\leq \epsilon$. Similarly to the security analysis of $\texttt{Encode}[n,1]$, the highest degree term of $\texttt{Encode}[n,2]$ is found in the last share : $x_n = x \oplus \tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus \bigoplus_{i=1}^{n-1} x_i$ and clearly no first or second-order combination of $(\tilde{x}_0,\tilde{x}_1,\tilde{x}_2,x_1,\ldots,x_n)$ is constant. Thus, the following holds for $\texttt{Encode}[n,2]$:

1. $\forall f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}$ and $\forall \overline{x} \in \mathbb{F}_2^N$ the bias of $f(\overline{x},\cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ':
$$\max_{f\in\mathcal{Y}^{(1)}\setminus\{\mathbf{0},\mathbf{1}\},\overline{x}\in\mathbb{F}_2^N} \mathcal{E}(f(\overline{x},\cdot)) \leq 3/8.$$
2. $\forall f \in \mathcal{Y}^{(2)} \setminus \{\mathbf{0},\mathbf{1}\}$ and $\forall \overline{x} \in \mathbb{F}_2^N$ the bias of $f(\overline{x},\cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ:
$$\max_{f\in\mathcal{Y}^{(2)}\setminus\{\mathbf{0},\mathbf{1}\},\overline{x}\in\mathbb{F}_2^N} \mathcal{E}(f(\overline{x},\cdot)) \leq 31/64.$$

The composability of our ϵ-2-AS encoding function E with the ϵ-2-AS circuit C is formulated in the proposition below, which can be proven analogously to Proposition 4.11.

Proposition 4.12. *Let $C(\overline{x}', \overline{r}_c) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit, and let $E(\overline{x}, \overline{r}_e) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be a function. If E is encoding ϵ-2-AS and C is circuit ϵ-2-AS then, for $d = 2$, it is true:*

$$\max_{f_{\overline{x}} \in \mathcal{R}_d} \mathcal{E}(f_{\overline{x}}) \leq \epsilon,$$

where $\mathcal{R}_d$ is defined in Definition 4.2.

4.3.5 Prediction Security – a Summary

Table 4.4: Summary of the ϵ-1-AS and ϵ-2-AS bounds for the `And`, `Xor` and `RefreshMask` gadgets and encoding function. The variable $e = -\log_2(1/2 + \epsilon)$ where $\epsilon = \max_{f_{\overline{x}} \in \mathcal{R}_d} \mathcal{E}(f_{\overline{x}})$ as in Proposition 4.1 and R_c denotes the minimum number of randomness to achieve 128-bit security.

	`And`	`Xor`	`RefreshMask`	`Encode`	$e \approx$	$R_c \geq$
ϵ-1-AS	7/16	1/4	1/4	1/4	9.3×10^{-2}	1.503
ϵ-2-AS	2047/4096	31/64	31/64	31 /64	3.5×10^{-4}	3.6×10^5

In the previous sections, we have shown the algebraic circuit security of our gadgets and algebraic encoding security of our encoding functions. In this section, we give the quantitative bounds on prediction security. We work with a security bound $\kappa = 128$ which implies $Adv^{PS}_{C,E,1} \leq 2^{-128}$ as defined in Definition 4.1.
We first consider the first-order prediction security bound of an $(n, 1)$ scheme. According to Proposition 4.1, $e = -\log_2(1/2 + \epsilon) = -\log_2(1/2 + 7/16) \approx 0.093$. The number of required random bits to achieve 128-bit security can be calculated as: $R_c \geq \kappa \cdot (1 + 1/e) = 128 \cdot (1 + 1/0.093) \approx 1503$. In conclusion, a circuit C composed with $(n, 1)$ gadgets and `Encode`$[n, 1]$ is 1-PS ($Adv^{PS}_{C,E,1} \leq 2^{-128}$) if the circuit contains $R_c \geq 1503$ random bits.
Next, we compute the second-order prediction bound for an $(n, 2)$ scheme. According to Proposition 4.1, $e = -\log_2(1/2 + \epsilon) = -\log_2(1/2 + 2047/4096) \approx 3.5 \times 10^{-4}$. As a result the number of required random bits to achieve 128-bit security is drastically increased to: $R_c \geq \kappa \cdot (1 + 1/e) = 128 \cdot (1 + 1/3.5 \times 10^{-4}) \approx 3.6 \times 10^5$. A summary of first and second-order algebraic security properties is given in Table 4.4.

Comparison with other masking scheme. An (n, d) scheme as defined in Section 4.2 is a combination of linear and multiplicative components. The allocation of these

components results in different orders of protections and thus the scheme has two corner cases: **(1)** $d = 0$ with $n \geq 1$ and **(2)** $d \geq 1$ with $n = 0$. The first case **(1)** acts as an additive masking. Such schemes are widely used in the literature, e.g. for Boolean masking [RP10], Threshold Implementations [NRS09], polynomial masking [RP12] and domain oriented masking [GMK16]. The common point of these schemes is that the degree of their encoding function is one, thus they are vulnerable to algebraic attacks, i.e. not prediction secure. On the other hand, the latter case **(2)** corresponds to a multiplicative masking scheme. A straightforward implementation of multiplicative masking is vulnerable to side-channel attacks [GT03,FMPR11], i.e. is not probing secure. However, the scheme can become secure if it is implemented with a solution that deals with the zero value, as given in [MQ18]. Although the scheme in [MQ18] is secure against side-channel attacks, the additive masking phase of the scheme is still vulnerable to algebraic attacks. Therefore, previously proposed masking schemes in the literature need to be combined with other masking schemes to accomplish both *prediction* and *probing* security notions.

A straightforward approach is to employ both linear and multiplicative components, as done in the *affine masking* by Fumaroli et al. [FMPR11]. The scheme processes a sensitive value x in the form of $r_1 x \oplus r_0$ such that $r_1, r_0 \in_R \mathbb{F}_2^n$ and *fixed* for each execution of the algorithm. As stated by the authors, affine masking is not *perfectly* secure against higher-order SCA but provides *practical* security. Indeed, some pairs of intermediate variables of the scheme depend on sensitive variables. A second-order side-channel attack can break the affine masking. Also, it is not clear how to generalize the scheme to higher orders. Another approach to combine linear and multiplicative components is given in [BU18]. However, the scheme alone does not provide security against computation attacks as described in Example 4.2. As a result, our scheme can be seen as a generalization of affine masking and the scheme by [BU18] in the sense of employing both linear and multiplicative components while providing *provable* security in both *prediction* and *probing* security notions. A summary of the security properties of the our scheme with different security orders is presented in Table 4.5.

Table 4.5: The security properties of masking schemes. The mark ○ (resp. ●) means the scheme is vulnerable (resp. resistant against) both to computational and algebraic attacks. Mark ◑ (resp. ◐) stands for vulnerability to computational but resistant against algebraic attacks (resp. resistant against computational but vulnerability to algebraic attacks). Remark that a masking scheme with $(n, 0)$ is the `ISW` transformation [ISW03] while a masking scheme with $(1, 1)$ is the scheme in [BU18]. The example structures for the masking schemes with $(2, 1)$ and $(3, 1)$ can be found in Chapter A.

d \ n	0	1	2	n
0	○	◐ [ISW03]	◐ [ISW03]	◐ [ISW03]
1	◑	◑ [BU18]	● Example A.1	●$[n, 1]$
2	◑	◑	● Example A.2	●$[n, 2]$
d	◑	◑	●	● Section 4.2.1

4.4 A Proof-of-Concept AES Implementation

Table 4.6: The number of gadgets in one round of AES.

	`SubBytes`	`MixColumns`	`AddRoundKey`	`ShiftRows`
`And`	16×32	-	-	-
`Xor`	16×83	27	128	-

In this section we introduce a white-box AES design based on the masking scheme defined in Section 4.2. The AES block cipher consists of multiple rounds of operations on its state. The operations include three linear layers: `MixColumns`, `ShiftRows`, and `AddRoundKey`, as well as the non-linear `SubBytes` layer. The bitwise implementation for the linear operations are straightforward. For the AES-Sbox we use the bitwise AES-Sbox design by Boyar and Peralta [BP09]. The resulting exact number of `And` and `Xor` gadgets within one round of AES-128 is given in Table 4.6. The resulting total number of bitwise operations[6] can be derived by combining the basic circuit size Table 4.6 and the performance analysis of the protected gadgets from Table 4.1. A visual representation of the AES-128 implementations with $(n, 0)$ (i.e. `ISW`-transformation), $(n, 1)$-masking scheme and $(n, 2)$-masking scheme is shown in Fig. 4.2. As a base line, we also provide numbers for the reference masking proposed in [BU18], where algebraically

[6]The bitwise `SubBytes` design by Boyar and Peralta [BP09] also requires Not gates. Although we didn't give the explicit description of a `Not` gadget in our masking scheme, it can be easily defined as identical to the `Not` gadget in the `ISW` transformation i.e. by flipping the n^{th} (linear) share.

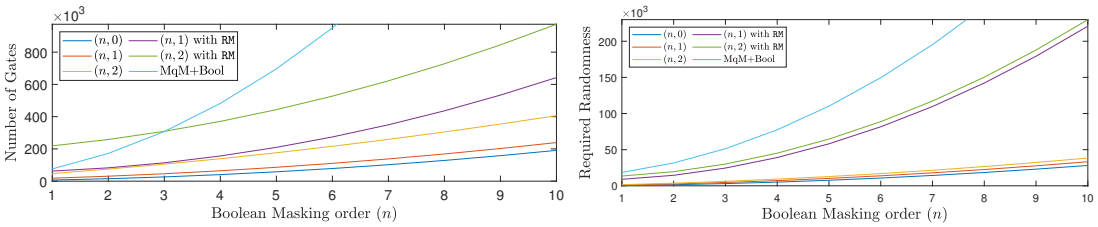


Figure 4.2: Total number of bitwise operations and required randomness for one round of AES-128 with different $(n, 0)$, $(n, 1)$ and $(n, 2)$ masking schemes with and without initial `RefreshMask` gadgets

secure gadgets and a probing secure masking were combined (each input is associated with a `RefreshMask` gadget, and the proposed minimalist quadratic Masking is combined with a second order Boolean masking).

As shown in Fig. 4.2, our hybrid construction outperforms the idea of using a first-order linear masking on top of a non-linear masking. As stated in [BU18], using a combination of two masking schemes even with the first-order protections requires roughly 200.000 gates per AES round. Since the foundation of our scheme is the `ISW` transformation, we can increase the probing security property of our scheme efficiently. However, increasing the non-linear order is the bottleneck of our scheme. When we compare the smallest possible implementations, we see that one round of AES-128 with $(2, 0)$, $(2, 1)$ and $(2, 2)$-masking schemes requires 15201, 30808(82678) and 74875(258415)gates respectively. The values in parentheses correspond to the gadgets where the inputs are *first* processed by `RefreshMask` gadgets. Clearly, `RefreshMask` gadgets impose a heavy overhead on our scheme. Therefore, a significant performance advantage can be achieved by further optimizing the `RefreshMask` gadget. While the first-order algebraically secure implementation requires a small overhead over an unprotected implementation, the second-order algebraically secure implementation comes at substantial cost. One round of AES-128 with $(2, 1)$, $(3, 1)$ and $(4, 1)$-masking schemes requires 30808(82678), 46115(113945) and 64494(156264) gates, respectively. Hence, the security against computational attacks can be increased with comparably moderate overhead when compared with the overhead of increasing the security against algebraic attacks. Furthermore, the randomness requirements of our scheme increases similarly to the `ISW`-transformation, as seen in Fig. 4.2.

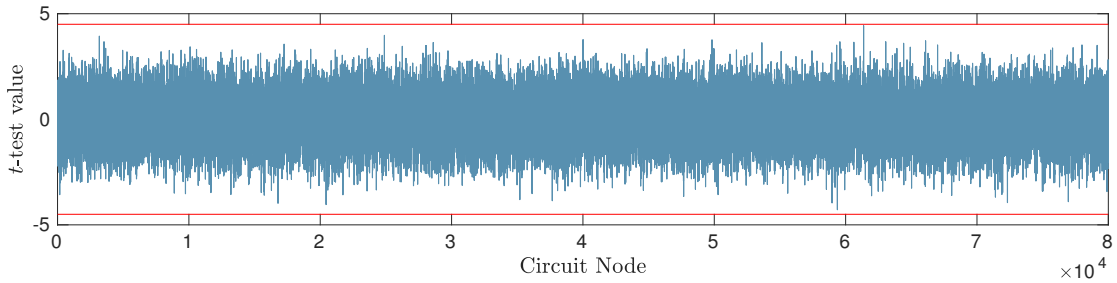


Figure 4.3: A first-order leakage test on a circuit that simulates the AES-128 with $(2, 1)$-masking defined in Section 4.2.1. Clearly, t-test value lie in threshold values as drawn by red lines ($[-4.5, 4.5]$).

4.4.1 Experimental Evaluation

To experimentally verify the security properties of our scheme we used the proof-of-concept AES-128 implementation. The implementations using $(n, 0)$, $(n, 1)$ and $(n, 2)$ masking schemes including the analysis are available as open source [7].
Software traces are simulated by encrypting N random plaintext and collecting the output of each node. We denote the i^{th} trace (corresponding to the encryption of i^{th} plaintext) by $t_i = \{v_1^i, \ldots, v_M^i\}$ where v_j^i denotes the output of j^{th} node and M denotes the number of the nodes in the circuit. Using the software traces we demonstrate a simple leakage detection test by the test vector leakage assessment (TVLA) as proposed by Goodwill et al. [GGJR+11] and as described in Section 2.1.4. Using the experimental setup we implement a first-order leakage detection test using 10000 traces (i.e. $n_f + n_r = 10000$) and $M = 80000$ (corresponds to the two round of AES-128). As expected the test shows no observable leakage. The illustration of the test can be seen in Fig. 4.3.

4.5 Conclusion

White-box cryptography has become a popular method to protect cryptographic keys in an insecure software realm potentially controlled by the adversary. All white-box cryptosystems in the literature have been practically broken due to differential computation analysis. Algebraic attacks have shown the inefficacy of classic side-channel countermeasures when they are applied in the white-box setting. Therefore, the need for a secure and reliable method to protect white-box implementations against both attacks has become evident.
We have proposed the first masking scheme that combines linear and non-linear

[7] https://github.com/UzL-ITS/white-box-masking

components to achieve resistance against both computational and algebraic attacks. The new scheme extends the `ISW` transformation to resist algebraic attacks by increasing the order of the decoding function. We have analyzed the two prevalent security notions in the white-box model, *probing security* and *prediction security*, and underlined the incompatibility of the notions, which reveals that a scheme should satisfy both notions. We have used the well-known SNI setting to prove $(n-1)^{th}$ order probing security of an (n, d)-masking scheme and thus we showed that our scheme can resist $(n-1)^{th}$-order computation attacks. We proved first and second-order prediction security for the concrete construction of the $(n, 1)$ and $(n, 2)$ masking scheme, respectively. We have defined the scheme generically such that it be extended to any orders of n and d, as long as the computational structure satisfies the algebraic properties. We have examined the implementation cost of our scheme for arbitrary orders of protection and compare it with the `ISW` transformation. We have extended the algebraic verification tool to support our scheme and to validate our results. The updated code has been made publicly available. Finally, a proof-of-concept bit-wise AES-128 implementation was provided to perform leakage detection and extensive performance analysis. The analysis showed that the new combined masking scheme outperforms the previous approaches which require to combine two different masking schemes to resist both attacks.

Part II

Attacks and Countermeasures for Post-quantum Cryptography

5

Introduction to Post-Quantum Cryptography

5.1 Motivation

In Part I we investigate the concepts belonging to modern computer science and modern cryptography. In the next we deep dive into future of cryptography. Today, we are witnessing the next level of computing machines, quantum computers, which requires new definitions, constructions and concepts for secure communication. Quantum computers don't just result in an increase in computation power which is answered by small (but effective) revisions such as increasing the key size or updated parameter sets. To counter new insights that may impact the security properties of cryptographic algorithms, new horizons for the families of algorithms are essential. When we consider the asymmetric encryption schemes, most are based on security primitives such as discrete logarithm problem (DLP) and factoring problem (FP). These problems are defined as cryptographic assumptions and they are always under the threat of the probability of breakthroughs in these. Therefore there is always a need for schemes that do not rely on number-theoretical assumptions as, quantum machines provide practical solutions to FP or DLP [Sho99].
Quantum computers have huge impact on asymmetric schemes while, symmetric schemes are saved by a slight update. As stated by the National Institute for Standards and Technology (NIST), larger key sizes for the symmetric schemes such as AES are needed to mitigate the security concerns and achieve the same security level as today [Gro96]. Therefore the current search for secure construction focuses on public-key schemes.

5.2 Post-Quantum Cryptography Standardization

It is not certain that we will be able to built a quantum computer or produce a commercialized quantum computer in a foreseeable future, nevertheless we need to be prepared for this situation. To coordinate the massive international research efforts in this area the NIST has been conducting the Post-Quantum Cryptography (PQC) Standardization Process, in which 3^{rd}-round *finalists* and *alternate candidates* have been recently announced [AASA+20] and there is 4^{th}-round on the horizon.
The project, that started in 2017 with 69 candidates, aims to select one or more signature schemes, and Key-encapsulation Algorithms (KEM). The first round evaluated the candidates with respect to their security, performance, and other characteristics and 26 algorithms were selected to proceed to the next round. The second round started early 2020 is based on the public feedback and internal reviews, candidates and finalists for the third round were established. The selected public-key encryption and key-encapsulation algorithms are Classic McEliece [CCU+20], CRYSTALS-KYBER [BDK+18], NTRU [CDH+17], and SABER [DKRV18] while the selected digital signatures are CRYSTALS-DILITHIUM [DKL+18], FALCON [FHK+18], and Rainbow [DS05]. Moreover, eight alternate candidate algorithms were also selected for further improvements by the cryptographic community. The alternate candidates are listed as: BIKE [ABB+17], FrodoKEM [ABD+19], HQC [MAB+18], NTRU Prime [BCLVV16], SIKE [ACC+17], GeMSS [CFMR+17], Picnic [Pic20], and SPHINCS+ [BHK+19].

5.2.1 Physical Attacks on Post-quantum Schemes

Side-channel resistance of cryptographic schemes is becoming more and more relevant as they are deployed in real-life condition. These attacks have been successfully performed for over 25 years and many countermeasures have been devised against them. As the countermeasures often come at the cost of running time or memory consumption of the protocol, many implementations are still vulnerable to them. Side-channel resistance is also one of the important evaluation criteria of the NIST PQC standardization process. A summary of attacks and countermeasures for signatures and KEM candidates submitted in the third round is given in Table 5.1.

5.2.1.1 Public-key Encryption and Key-encapsulation Schemes

The Classic McEliece [CCU+20] cryptosystem is build on the idea to use a random binary Goppa code as the public key. There are multiple DPA attacks [CEvMS15, HMP10] on this system, that allow the extraction of the secret key. Moreover, Chen et al. [CEvMS16]

Table 5.1: A summary of attacks and countermeasures on PKE and KEM Summary Digital Signature Algorithms

Schemes	Power Analysis	Countermeasures
Classic McEliece	[CEvMS15, HMP10]	[CEvMS16]
CRYSTALS-KYBER	[SKL+20]	[OSPG16]
NTRU	[SKL+20, AKJ+18, ABGV, WZW13]	
SABER	[SKL+20, Ver19]	[VBDK+20, Ver19]
BIKE	[SKC+19]	-
FrodoKEM	[SKL+20]	-
HQC	[SRSWZ20]	-
NTRU Prime	[SKL+20, HCY19]	-
SIKE	[ZYD+20]	[ZYD+20]
CRYSTALS-DILITHIUM	[RJH+18, FDK20]	[MGTF19]
FALCON	[KA21]	-
Rainbow	[YL17, PSKH18]	[Yi18]
GeMSS	[YL17, PSKH18]	-
Picnic	[GSE21]	[SBWE20]
SPHINCS+	[KGB+18]	-

explain how to apply masking to the McEliece cryptosystem and present a masked FPGA implementation resistant to DPA.

The CRYSTALS-KYBER [BDK+18] cryptosystem is based on the learning-with-errors (LWE) problem over module lattices. The designers provide an overview of potential attacks and emphasized that without dedicated protection, the scheme will be vulnerable to Side-channel attacks like DPA. A recent work by Bos et al. [BGR+21] provides a masked implementations.

NTRU [CDH+17] is an cryptosystem that relies on lattice-based cryptography to securely encrypt and decrypt information. The work by An et al. [AKJ+18] breaks the implementation using only a single power trace and proposing countermeasures to their attack. Lee et al. [LSCH10] also discuss NTRU's vulnerability to power analysis attacks and propose countermeasures against these with very little overhead.

The SABER [DKRV18] Key Encapsulation Mechanism (KEM) relies on the hardness of the Module Learning With Rounding problem (MLWR), which is believed to remain secure even against quantum computers and is said to be IND-CCA2 secure. Beirendonck et al. [VBDK+20] discuss masking the scheme to prevent Side-channel attacks with a 2.5 overhead factor. Verhulst [Ver19] implements SCA using the key generation and encryption as these operations only operate on ephemeral secrets. Moreover, it has been shown that the decryption mechanism of SABER is susceptible to DPA. Finally, Ngo et al. [NDGJ21] show a successful side-channel attack on a first-order masked

implementation of secure Saber

BIKE [ABB+17] is a code-based Key Encapsulation Mechanism (KEM) relying on Quasi-Cyclic Moderate Density Parity-Check codes for its security. At CHES 2016, a constant-time multiplication was introduced by Chou at al. [Cho16], aiming to protect the scheme against timing attacks. However, this countermeasure is shown to be vulnerable against DPA [RHHM17, SKC+19].

FrodoKEM [ABD+19] relies on the learning with errors problem. Sim et al. [SKL+20] proposed a single-trace Side-Channel attack making use of power traces targeting the scheme.

HQC [MAB+18] stands for Hamming Quasi-Cyclic and is a code-based public key encryption scheme. Schamberger et al. [SRSWZ20] were able to use a power Side-Channel attack against an HQC-128 implementation on an ARM Cortex-M4 microcontroller to extract 93.2% of the possible keys with less than 10000 measurement traces.

NTRU Prime [BCLVV16] is a lattice-based cryptosystem that uses the large Galois group instead of cyclotomics for increased security. Huang et al. [HCY19] performed a correlation power analysis attack on NTRU Prime and were able to reveal all coefficients of each private-key polynomial very easily. They propose three countermeasures against their attack, with performance being a necessary tradeoff.

SIKE [ACC+17] stands for supersingular isogeny key encapsulation and is based on pseudo-random walks in supersingular isogeny graphs. Recently Zhang et al. [ZYD+20] analyzed SIKE and proposed a DPA attack on the scheme. They also proposed a countermeasure to eliminate the leakage with very little cost in terms of time and memory.

5.2.1.2 Digital Signature Schemes

CRYSTALS-DILITHIUM [DKL+18] is a digital signature scheme based on the hardness of lattice problems over module lattices, particularly the Learning with Error (LWE) problem. Ravi et al. [RJH+18] investigate the security of DILITHIUM against Side-Channel attacks focusing on the LWE instance. Fournaris et al. [FDK20] also make use of correlation power Analysis to attack DILITHIUM's polynomial multiplication operation. Migliore et al. [MGTF19] explore how to securely mask the scheme and verify that their masked implementation no longer presents any leakage. Their masking scheme, using a slightly tweaked version of Dilithium, makes it possible to apply masking with an overhead of 7.3 to 9 factor.

Falcon [FHK+18] is a cryptographic signature scheme based on NTRU lattices using fast Fourier sampling as a trapdoor function. Karabulut et al. [KA21] show a Side-Channel

attack targeting unique floating-point multiplications within FALCON's Fast Fourier Transform.
Rainbow [FHK+18] belongs to the family of multivariate public key cryptosystems and is based on the Oil-Vinegar signature scheme by Patarin [Pat97]. Park et al. [PSKH18] managed to extract the full secret key from a Rainbow implementation on an 8-bit AVR microcontroller using correlation power analysis.
GeMSS [CFMR+17] stands for "a Great Multivariate Short Signature" and is a multivariate based signature scheme with a fast verification process and a medium to large public-key.
SPHINCS+ [BHK+19] is a stateless hash-based signature scheme, based on the idea of the Lamport signature scheme. Kannwischer et al. [KGB+18] were able to compromise a SPHINCS-256 implementation using a DPA attack and extracted a 32-bit chunk of the secret key.
Next, we introduce the details of the Picnic signature family, which will be our focus in the following chapters.

5.3 Picnic Signature Scheme

The Picnic signature scheme [ZCD+20] was selected as an alternate candidate and follows Ishai et al.'s MPC-in-the-head (or MPCitH, short for *multi-party computation in-the-head*) paradigm for constructing Zero-knowledge (ZK) proof systems [IKOS07]. One of the attractive features of MPCitH-style signatures is that they require no number-theoretic hardness assumptions, since the typical construction of such schemes only relies on symmetric key primitives. Concretely, following the standard Fiat–Shamir paradigm [FS87], signatures in the MPCitH paradigm can be proven secure in the random oracle model, as long as the underlying hash function and block cipher are secure. Quoting [Nat20], "NIST also sees Picnic's reliance on only assumptions about symmetric primitives as an advantage in case the need arises for an extremely conservative signature standard in the future". In the following we give the building blocks of the Picnic signature scheme.

5.3.1 Zero-knowledge Proofs

A *zero-knowledge proof* between a *prover* P and a *verifier* V is a two-player game. The goal of the the prover P is to convince the verifier V that they know a certain secret x without revealing *any* information about this secret. Zero-knowledge proofs are extremely useful for different cryptographic applications such as signature schemes

or multi-party computations. In this work, we only need a certain kind of well-structured protocol, called a Σ-protocol. In the following, let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be an nondeterministic Polynomial-time (NP)-relation, i. e. for all $x, w \in \{0,1\}^*$, the value $R(x,w)$ can be computed in polynomial time and if $R(x,w) = 1$, we have $|x| \leq |w|^{O(1)}$. Here, we identify the relation R with its binary characteristic function $R\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ with $R(x,w) = 1$ iff $(x,w) \in R$. The value x is a *witness* to w. By L_R, we denote the language associated with R, i. e. $L_R = \{w \mid \exists x \text{ s.t. } R(x,w) = 1\}$. In some parts of this work, we make the relation R explicit using a function $\phi\colon \{0,1\}^* \to \{0,1\}^*$ with the natural interpretation of $L_\phi = \{w \mid \exists x \text{ s.t. } \phi(x) = w\}$.

Definition 5.1 (Σ-protocol [HL10]). The goal of the protocol $\Pi_R(y)$ between two players P and V is to convince V that $y \in L_R$, where $y \in \{0,1\}^*$ is known to both players. Such a protocol is called a Σ-protocol for the relation R if it satisfies the following conditions:

- Π_R has the following communication pattern:
  1. **Commit**: P sends a first message $\mathbf{a}$ to V,
  2. **Challenge**: V sends a random element $\mathbf{e}$ to P,
  3. **Prove**: P replies with a second message $\mathbf{z}$.
- **Completeness**: If both players P and V are honest and $\mathbf{y} \in L_R$, then $Pr[(\mathsf{P},\mathsf{V})(\mathbf{y}) = \texttt{accept}] = 1$.
- s**-Special Soundness**: For any $\mathbf{y}$ and any set of $s \geq 2$ of accepting conversations $\{(\mathbf{a}, \mathbf{e}_i, \mathbf{z}_i)\}_{i\in[s]}$ with $\mathbf{e}_i \neq \mathbf{e}_j$ if $i \neq j$, a witness $\mathbf{x}$ for $\mathbf{y}$ can be efficiently computed.
- **Special honest-verifier ZK**: There exists a PPT simulator S that on input $\mathbf{y} \in L_R$ and $\mathbf{e}$ outputs a triple $(\mathbf{a}', \mathbf{e}, \mathbf{z}')$ with the same probability distribution as real conversations $(\mathbf{a}, \mathbf{e}, \mathbf{z})$ of the protocol.

Furthermore, a Σ-protocol is a *public-coin* protocol, if the verifier V only sends random messages. Hence, the Fiat-Shamir transformation [FS86] or the Unruh transformation [Unr15] can be used to make them non-interactive in the random oracle model. Note that the Unruh transformation always gives security against quantum adversaries, while the Fiat-Shamir transformation does not do this in general [ARU14]. Nevertheless, recently it was shown that the Fiat-Shamir transformation is still secure against quantum adversaries for a large class of protocols [LZ19, DFMS19].

5.3.2 MPC-in-the-head Paradigm

Next, we describe the basic approach to construct a Zero-knowledge Proof of Knowledge system for an arbitrary NP language L, following Ishai et al. [IKOS07] and its generalization due to Giacomelli et al. [GMO16]. Given L, we can define an *NP relation* $R(\mathbf{x}, \mathbf{w})$ which returns 1 if its input consists of a valid pair of *statement* $\mathbf{x} \in L$ and corresponding *witness* $\mathbf{w}$, and outputs 0 otherwise. An MPCitH proof system (P, V) is built upon some N-party MPC protocol that jointly computes a function f, where f takes $\mathbf{x}$ and $\mathbf{w}$ as public and private input, respectively, and outputs $f_{\mathbf{x}}(\mathbf{w}) = R(\mathbf{x}, \mathbf{w})$. For example, for a given encryption algorithm Enc of a block cipher like LowMc [ARS+15], one can define $f_{\mathbf{x}}(\mathbf{w}) \coloneqq \mathsf{Enc}(sk, p) \stackrel{?}{=} c$, where the statement $\mathbf{x} = (p, c)$ is a plaintext-ciphertext pair, and witness $\mathbf{w} = sk$ is a private encryption key, respectively. In this case, the prover P proves knowledge of a private key that produces a certain public ciphertext from the corresponding public plaintext.

At a high level, an MPCitH prover P attempts to convince the verifier V that they hold a valid witness $\mathbf{w}$, by letting V check that the MPC protocol has been correctly carried out "in P's head" on input $\mathbf{w}$. We now consider an MPC protocol Π_C for the corresponding arithmetic circuit C defined over a finite field $\mathbb{F}$, where the statement information $\mathbf{x}$ (e.g., the plaintext-ciphertext pair) is hard-coded such that $C(\cdot) = f_{\mathbf{x}}(\cdot)$. We assume that the witness is expressed by an n-dimensional vector and C takes a set of n input wires denoted by IN. We write $\mathbf{w} = (w)_{w \in \mathsf{IN}} \in \mathbb{F}^n$ for the complete input.[1] To initialize the protocol, the prover P first additively secret shares each input w wire such that $w = w_1 + \ldots + w_N$ in $\mathbb{F}$, and considers each share w_i as a private input to a party P_i. Then P internally runs Π_C to obtain $\mathsf{view}_1, \ldots, \mathsf{view}_N$, where each view_i consists of P_i's private input w_i, the random tape of P_i and all incoming messages that P_i observes during the execution of Π_C. The proof system now proceeds by following the typical "commit–challenge–response" flow. Using a secure commitment scheme, P sends $\mathsf{Commit}(\mathsf{view}_i)$ for all $i \in [N]$ as the first message. Upon receiving distinct challenges $i_1, \ldots, i_t \in [N]$ from the verifier V, the prover P sends back the corresponding t views $\mathsf{view}_{i_1}, \ldots, \mathsf{view}_{i_t}$ as well as the commitment opening information as a response. Finally, the verifier V accepts the proof iff the opened views are consistent with each other and they produce 1 as output of the protocol Π_C. The (honest verifier) zero knowledge is guaranteed as long as the underlying MPC Π_C has *t-privacy* in the semi-honest model (i.e., the distribution of any $\leq t$ views during an honest execution of the protocol is

[1] Note that we're slightly abusing the notation here. Throughout, we use the same notations (typically w, x, y and z) for both *wires* and *wire values*, but it should be clear from the context which they indicate.

polynomial-time simulatable, given the output from Π_C and corresponding $\leq t$ parties' private input).

5.3.2.1 ZKBoo: Zero-knowledge for Boolean Circuits

An important Σ-protocol based on the MPC-in-the-head paradigm is called ZKBoo [GMO16]. The goal of the protocol is to convince the verifier that the prover has an input x to an arithmetic circuit ϕ such that $\phi(x) = y$, where ϕ and y are publicly known. The general idea behind ZKBoo is the partition of ϕ into a $(2,3)$-*decomposition*, i. e. the computation of this circuit is split into three *branches* ϕ_0, ϕ_1, ϕ_2. The input x to ϕ is furthermore split into three shares x_0, x_1, x_2 such that the computation of ϕ_i only depends on the shares x_i and x_{i+1}. After this computation by the prover, the verifier chooses a random index $\mathbf{e} \in_R \{0,1,2\}$ and is given the computations of $\phi_{\mathbf{e}}$ and $\phi_{\mathbf{e}+1}$ along with the inputs $x_{\mathbf{e}}$ and $x_{\mathbf{e}+1}$. This information can be used to verify the computations on these branches without revealing the complete input x to the verifier. Due to the small size of the communication — roughly dominated by the number of multiplication gates in ϕ — the ZKBoo-protocol has seen wide use. We provide a detailed description of the protocol in Fig. 5.1 and the details can be found below.

Definition 5.2 ((2,3) circuit decomposition [GMO16]). A(2,3)-decomposition for the function ϕ is a set of functions,

$$\mathcal{D} = \{\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}\} \cup \mathcal{F},$$

where Share, Rec and Output_i are defined as in Section 7.3. Let Π^*_ϕ be the algorithm described in Fig. 7.2 with $n = 2$, we have the following definitions.

- **Correctness:** We say that $\mathcal{D}$ is correct if $Pr[\phi(x) = \Pi^*_{\phi(x)}] = 1$ for all $x \in X$. The probability is over the choice of the random tapes R_i.
- **Privacy**: We say that $\mathcal{D}$ has 2-privacy if it is correct and for all $e \in [0,2]$ there exists a PPT simulator S_e such that $((R_i, w_i)_{i\in\{e,e+1\}}, y_{e+2})$ and $S_e(\phi, y)$ have the same probability distribution for all $x \in X$.

Most famously, an optimized version of ZKBoo called ZKB++ is the basis of the post-quantum secure Picnic signature scheme — an alternate candidate in round three of the NIST standardization process [CDG+17]. Note that Picnic2 (resp. Picnic3) also use the MPC-in-the-head paradigm, but are based on the KKW protocol [KKW18] which allows for a preprocessing phase and better parameter tuning [KZ20].

A (2,3)-decomposition of a function ϕ is given as Π_ϕ. The verifier and the prover have input $y \in L_\phi$. The prover knows x such that $y = \phi(x)$.

Commit: The prover does the following:

1. Generate random tapes R_0, R_1, R_2.
2. Run $\Pi_\phi(x)$ with randomness R_0, R_1, and R_2 to obtain views w_0, w_1, w_2 and outputs y_0, y_1, y_2.
3. Commit to $c_i = \mathsf{Comm}(w_i, R_i)$ for $i \in [0, 2]$.
4. Send $\mathbf{a} = (y_0, y_1, y_2, c_0, c_1, c_2)$.

Prove: The verifier chooses an index $\mathbf{e} \in [0, 2]$ and sends it to the prover. The prover answers to the verifier's challenge sending opening $c_\mathbf{e}$, and $c_{\mathbf{e}+1}$ thus revealing $\mathbf{z} = (R_j, w_j)_{j \in \{\mathbf{e}, \mathbf{e}+1\}}$.

Verify: The verifier runs the following checks:

1. If $Rec(y_0, y_1, y_2) \neq y$, output `reject`.
2. If $\exists\ i \in \{\mathbf{e}, \mathbf{e}+1\}$ such that $y_i \neq \mathrm{Output}_i(w_i)$, output `reject`.
3. If $\exists\ j$ such that the j-th output is not equal to $w_\mathbf{e}^{(j)} \neq \phi_\mathbf{e}^{(j)}(w_\mathbf{e}, R_\mathbf{e}, w_{\mathbf{e}+1}, R_{\mathbf{e}+1})$, output `reject`.
4. Output `accept`.

Figure 5.1: ZKBoo protocol as defined by Giacomelli et al. [GMO16].

5.3.3 MPC in the preprocessing model.

In work following [GMO16, CDG+17], Katz, Kolesnikov, and Wang [KKW18] showed that a particular communication-efficient MPC protocol *in the preprocessing model* is well suited to MPCitH proofs, and variants of their protocol appear in subsequent work [dDOS19,BN20,KZ20]. The core idea of MPC in the preprocessing model is to split the protocol Π_C into an *offline* phase Π_C^{off} and an *online* phase Π_C^{on}. Importantly, the offline phase Π_C^{off} can be computed *independently of the witness*. By precomputing correlated randomness in advance during Π_C^{off}, one can reduce communication in Π_C^{on} drastically. In the traditional MPC setting, this was already used, e.g., in SPDZ [DPSZ12], MiniMAC [DZ13], and TinyOT [NNOB12]. While the original KKW proof system is focused on the protocol for *Boolean circuits*, it also works with *arithmetic circuits* in a straightforward manner as observed in [dDOS19, BN20], so we present the latter case here for the sake of generality.

(Offline Phase) The offline phase Π_C^{off} of KKW works as follows: for each input

wire $w \in \mathsf{IN}$ to the circuit C, and for each output wire z from all the multiplication gates, each party P_i locally generates *random shares* $\lambda_i^w, \lambda_i^z \in \mathbb{F}$ using its own random tape. Then the parties compute random shares for all internal wires, by running the circuit:

- for each addition gate that takes wires x and y as input, party P_i locally computes a new share $\lambda_i^z = \lambda_i^x + \lambda_i^y$ for the output wire z;
- for each multiplication gate that takes wires x and y as input, party P_i obtains shares of the *multiplication triples* (sometimes called *Beaver triples* [Bea92]) $(\lambda_i^x, \lambda_i^y, \lambda_i^{xy})$, such that $\lambda^{xy} = \lambda^x \lambda^y$.

(Multiplication Triples) To generate multiplication triples in the MPCitH setting, the parties choose λ^x and λ^y implicitly by reading their shares from their random tapes. Then to obtain shares of λ^{xy}, the first $N-1$ parties read random shares from their random tapes. As the prover P knows all the shares, P can simply solve for the N-th party's share so that the shares reconstruct λ^{xy}, as required. We call the sequence of values λ_N^{xy} for all multiplication gates *auxiliary information*, denoted $\mathsf{aux} \in \mathbb{F}^{|C|}$. Note that the complete information needed for the first $N-1$ parties can be derived from their respective seeds seed_i used to generate P_i's tape. The information needed for party P_N can be derived from seed_N and from aux. Hence, we define each party P_i's *state* information as follows: for all $i = 1, \dots N-1$, let $\mathsf{state}_i \coloneqq \mathsf{seed}_i$, and for P_N we have $\mathsf{state}_N \coloneqq \mathsf{seed}_N || \mathsf{aux}$.

(Online Phase) Given the preprocessed state information, the online phase Π_C^{on} proceeds by computing the masked witness $\hat{w} = w + \sum_{i \in [N]} \lambda_i^w$ for each input wire. Now, each gate takes (masked) inputs $\hat{x} = x + \sum_{i \in [N]} \lambda_i^x$ and $\hat{y} = y + \sum_{i \in [N]} \lambda_i^y$ and can be computed as follows, where all computations on shares are carried out in $\mathbb{F}$:

- Addition: each P_i locally computes $\hat{x} + \hat{y}$.
- Addition by constant c: each P_i locally computes $\hat{x} + c$.
- Multiplication by constant c: each P_i locally computes $c \cdot \hat{x}$.
- Multiplication: this computation consumes a single triple $((\lambda_i^x)_{i \in [N]}, (\lambda_i^y)_{i \in [N]}, (\lambda_i^{xy})_{i \in [N]})$. Each party P_i first locally computes $s_i = \lambda_i^z - \hat{x} \cdot \lambda_i^y - \hat{y} \cdot \lambda_i^x - \lambda_i^{xy}$ and broadcasts s_i. Then the masked output $\hat{z} = xy + \sum_{i \in [N]} \lambda_i^z$ can be obtained as $\hat{z} = \sum_{i \in [N]} s_i + \hat{x}\hat{y}$ by each party.

Notice that Π_C^{on} only broadcasts once for each multiplication gate, thanks to the

correlated randomness computed during the offline phase. All other operations are computed locally by the parties.

Protocol. Below we present a basic framework for three-round MPCitH-PP proof systems. Here, we describe the protocol for one MPC instance (with non-negligible soundness error) and in Fig. C.1 we include a complete description of the KKW proof system that uses many instances in parallel (to achieve negligible soundness error). As the offline protocol proceeds independently of the secret witness, an MPCitH-PP prover can safely open the states of all N parties for the verification of the preprocessing phase (i.e., triple generation).

(Commit) The prover P first samples a random seed for each P_i and executes Π_C^{off} to obtain the states of all N parties after the offline phase. Then, using these states and the masked witness $(\hat{w})_{w\in\mathsf{IN}}$ as input, P executes Π_C^{on} to obtain all broadcast messages observed during the online phase. Finally, P sends commitments to the states and broadcast messages to the verifier V.

(Challenge) V asks P to open either the offline or the online phase. For the latter case, V also randomly picks a party index $i^\star$, whose view is to remain hidden.

(Response) To open the offline phase, P sends all random seeds used during Π_C^{off}. To open the online phase, P sends broadcast messages coming from the party $P_{i^\star}$ during Π_C^{on}, as well as all the state information of the remaining $N-1$ parties.

(Verification) To check the offline phase, V simply uses random seeds to execute Π_C^{off} as P would do, to obtain the resulting states of all N parties. Then V checks that these states form a correct opening to the commitment of the offline phase. To check the online phase, V simulates Π_C^{on} with the broadcast messages from $P_{i^\star}$ and the states of the remaining $N-1$ parties as input, so as to obtain the broadcast messages of the other $N-1$ parties. Then, V checks that these broadcast messages form a correct opening to the commitments of the online phase.

5.3.4 Summary of Picnic Signature Scheme

The signature scheme Picnic is an instance of the MPCitH paradigm described above. The function f is the LowMc block cipher[2], the signer's secret key is the witness $\mathbf{w}$, and the public key is $(\mathbf{x}, \mathbf{c})$. A signature consists of a proof of knowledge of $\mathbf{w}$ such

[2]The details of the LowMc cipher can be found in Section C.5

that $f_{\mathbf{x}}(\mathbf{w}) = \mathbf{c}$. In the block cipher notation, if the secret key is denoted sk, then the public key is a plaintext-ciphertext pair $(\mathbf{x}, \text{LowMC}_{sk}(\mathbf{x}))$ where $\mathbf{x}$ is a randomly chosen plaintext block, and the signature proves knowledge of a key relating the plaintext $\mathbf{x}$ and the ciphertext $\text{LowMC}_{sk}(\mathbf{x})$. The proof is made non-interactive by the Fiat-Shamir transform, and the message to be signed is bound to the proof by hashing it into the challenge.

The parameter sets for the algorithms submitted to the NIST project must meet one of five security levels. Picnic defines parameters for security levels L1, L3 and L5, corresponding to the security of AES 128, 192 and 256, respectively. For instance, parameters at level L1 aim to provide 128-bit security against classical attacks.

6

Side-channel Analysis of Picnic Signatures

6.1 Motivation

In this chapter, we explore probing attacks on the MPC-in-the-head paradigm using the Picnic signature scheme and MPC-in-the-head paradigm with preprocessing using Picnic3 signature scheme. We first remark the attack surface as illustrated in Fig. 6.1. Our Goal is to show that the MPCitH phase using Picnic (or as in short MPC-LowMC phase) is indeed vulnerable to to Side-Channel attacks, which then can be used to recover the sensitive values.

As our target, we focus on the reference implementation given by the authors [Ste] which uses the Unruh transformation with security parameters `L1`. However, our analysis is independent of the actual transformation (Fiat-Shamir or Unruh Transformation) and can be adapted to the different security parameters. The first attack targets the initial

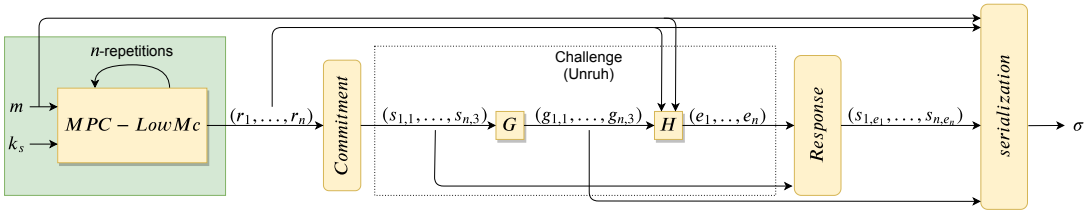


Figure 6.1: An overview of the Picnic signature scheme where m is constant plaintext that is used in LowMC such that $\mathsf{LowMC}(m, k_s) = y$, $sk = ((y, m), k_s)$ is the secret key and $pk = (y, m)$ is the public key. The figure is adapted from [AOTZ20].

sharing of the secret key before its use in the MPC-LowMC implementation while the second attack targets an intermediate state in the shared Sbox implementation of MPC-LowMC.

6.2 Probing Attacks on MPC-in-the-head Paradigm

In general, the goal of a *probing attack* is to reconstruct the secret input x given to some algorithm A by obtaining values used in the computation of $\mathsf{A}(x)$. We say that an algorithm A is k-secure, if at least k probes are needed to reconstruct the secret x. Combining the masking technique with masking order n and modifying the circuits used in A by using n-SNI gadgets results in an algorithm A' that is $n \cdot k$-secure [BBD+16]. Now, consider the case that A' is an implementation of the $(n+1)$-party MPC-in-the-head zero knowledge protocol as given in Section 5.3.2.1. As the protocol gives out t shares to the verifier, the security of A' drops down by t to $n \cdot k - t$, as t input shares are now known to the attacker. Motivated by this, we now introduce our adversarial model and experimental results.

We assume an adversary who can access the physical device that can run the Picnic signatures. They can measure side-channel traces, such as power or electromagnetic emanation, of the device while signing chosen messages. Moreover, they obtain the signatures as output, and can verify (and thus see the revealed values) or use them arbitrarily in an attack. Observe, that according to noisy leakage model the side-channel trace contains each intermediate value perturbed with a noisy leakage function. Depending on the signature the revealed values vary and the adversary can employ these variables, to recover the secret. Remark, that depending on the scenario *the secret* is changing (the details is given in Section 6.3). Therefore, our countermeasures introduced in Chapter 8 are to thwart these two scenarios.

6.2.1 Experimental Results

In order to illustrate that this is not only a theoretical weakness, we study the ZKBoo protocol using the $(2,3)$-circuit decomposition as defined in Section 5.3.2.1. We first show that an attack using the opened views is indeed possible by using *a single probed* value using the experimetal setup given in Chapter B.

For ZKBoo, we assume the scenario where two of three shares (x_0, x_1) are revealed by the protocol. The MPC-in-the-head measurements have a weak and noisy dependence on x_2, which can be exploited due the revealed shares a'_0 and a'_1. In order to validate the straightforward exploitability, we use a simple RvR t-test setup. We collect traces

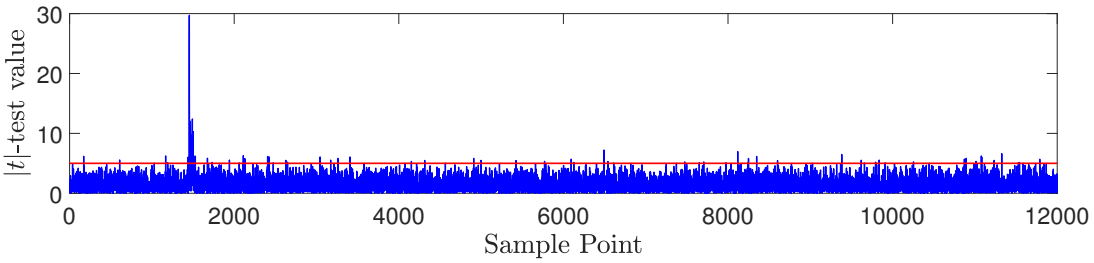


Figure 6.2: A t-test based leakage detection of a single output bit (a') in Picnic using the classification based on $i = a'_0 \oplus a'_1$. The details of the experimental setup and formulation can found in Chapter B.

of Picnic signature generation using our practical setup as described in Chapter B. The analysis uses the side-channel information of the unopened view and the two opened shares of a multiplication gadget. We target a single bit (e.g. $a' = a'_0 \oplus a'_1 \oplus a'_2$) inside the `SboxLayer` and classify the traces into two groups depending on the value of $a'_0 \oplus a'_1$. The result of the t-test in Fig. 6.2 shows the clear dependence between the unrevealed share a'_2 and the observable measurement traces, as the t-value clearly exceeds 4.5.

6.3 Probing Attacks on Picnic3

We describe two probing side-channel attacks against Picnic3 and experimentally confirm them against our M4 port of the optimized Picnic3 implementation described in Chapter B. The values revealed by the prover, allowing the verifier to check the consistency of the MPC protocol, can be employed by an adversary in a side-channel attack. We assume the same scenario. Furthermore, we assume a leakage model where an implementation leaks weak and noisy information about each intermediate variable, therefore measurements of the MPC-in-the-head simulation leak a weak and noisy dependence on secret values due to the revealed values. We make use of the RvR tests to show the clear *presence* of leakage.

6.3.1 Probing the Masked Secret of Unopened Online Phase

This attack is specific to MPCitH with preprocessing, and only occurs when 5 protocol rounds are compressed to 3. Hence, the attack below works in principle for any direct implementation of signatures derived from three-round KKW-based protocols. We also remark, that this attack cannot be mitigated by the SNIitH approach [SBWE20] (Chapter 7); in particular, the attack below works *independently of the number of unopened parties' views* since it targets an input to the MPC (i.e., $\hat{w}$), not a share of the

secret. We are thus motivated to design an alternative solution to thwart this attack in the next section.

We first note the three-round KKW scheme executes both the offline and online phase of each MPC instance, in contrast to the five-round case. We denote by $\mathcal{C}$ the executions chosen for the online phase, i. e., the executions where the offline phase is not public.

The attack exploits the following: if the k-th execution of the offline phase is selected to be part of the signature (i.e., if $k \notin \mathcal{C}$), the preprocessed masks and state of all N parties is made public for the verifier, therefore the corresponding online phase must remain hidden. Concretely, since the secret witness wire value w is masked by random bits in Item 1c of the prover in Fig. C.1, the attacker's goal is to learn the masked witness wire values $\hat{w}^{(k)}$ in execution k for the unopened online executions $k \notin \mathcal{C}$. Since $\hat{w}^{(k)} = \lambda_1^{w,(k)} + \ldots + \lambda_N^{w,(k)} + w$ and $\lambda_i^{w,(k)}$ is made public (for all i), by probing $\hat{w}^{(k)}$ the attacker can solve for the secret key bit w. Here, $\lambda_i^{w,(k)}$ denotes the value of λ_i^w in execution k.

In order to validate the attack, we use our experimental setup (as described in Section 8.4) and the RvR approach. The experiment shows that there is an *exploitable leakage*, i.e., an amount of leakage sufficient, despite measurement noise, to allow recovery of intermediate values that depend on the secret key. For this experiment, we reduce the Picnic3 parameters (as in Fig. C.1) M and τ to 4 and 2 respectively, in order to collect traces more quickly, however we keep the number of parties N as 16 and collected traces corresponding to execution of the first MPC instance. During the collection phase, we run the Picnic3 signing function with random messages and a fixed secret key. More specifically, we measure the execution of the first line of Algorithm 35, and collect 22,056 side-channel traces. Note, that the root seed is also random due to the choice of a random message to be signed. We first separate the traces belonging to signatures that reveal the first preprocessing phase, since our measurement covers the whole first MPC instance. The reduced number of MPC instances is only to reduce the number of possible challenges and to increase the number of traces per challenge. Then we classify the remaining traces into two sets according to the revealed values $\lambda_1^{w,(k)} + \cdots + \lambda_{16}^{w,(k)}$. The result of the analysis in Fig. 6.3 (left side) shows a clear dependence between the unrevealed value $\hat{w}^{(k)}$ and the observable trace, as the $|t|$-value clearly exceeds 5.7 which shows an exploitable leakage. As seen in the right hand side of the Fig. 6.3, the leakage becomes clear after 2,725 traces.

The code we measure (the first line of Algorithm 35) corresponds to the calculation of $\mathsf{roundkey}_0$ thus the leakage corresponds to the bits of $\mathsf{roundkey}_0$ which is equal to $\mathsf{matMul}(\hat{sk}, K_0)$. Solving the equation for the $sk = \hat{sk} - (\lambda_1^{sk} + \cdots + \lambda_{16}^{sk})K_0$ where $(\lambda_i^{sk}$

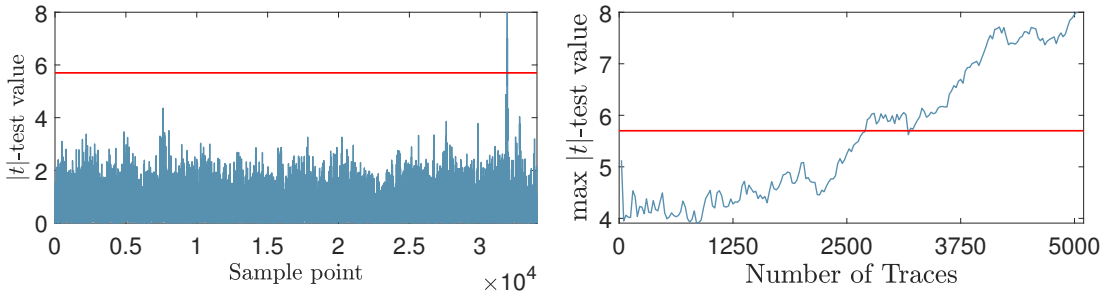


Figure 6.3: A first-order RvR test on the unprotected Picnic3 implementation using revealed values from the offline phase. The traces are classified based on the bit $\lambda_1^w + \cdots + \lambda_{16}^w$. Values above the 5.7 threshold (red line) indicate there are strong leakages (left). Moreover, the maximum $|t|$-value increases with respect to number of traces and the leakage becomes clear after 2,725 traces (right).

is known for all i and K_0 is a constant) leads to the secret value sk.

6.3.2 Probing the Unopened Party

The second attack uses the revealed values for the online phase i.e. $\hat{w}^{(k)}$ and λ_i^w for $i \neq i_k$. This attack is a straightforward variant of the one by Gellersen et al. [GSE21], but adapted to work with Picnic3. In contrast to the attack described above, we now target an MPC execution whose *online* phase is selected to be part of the signature (i.e., if $k \in \mathcal{C}$). In that case there is a single party P_{i_k} whose internal state must remain hidden for the privacy of the MPC protocol to hold. By design, the values $\hat{w}^{(k)}$ and λ_i^w for $i \neq i_k$ are revealed during the verification. Thus the measurements have a weak and noisy dependence to the value $\lambda_{i_k}^w$ which can be exploitable due to the revealed values. We validate the attack using the same experimental setup and parameters as in Section 6.3.1. During the collection phase, we again process the Picnic3 with random messages with a fix secret key and measure the execution of the preprocessing phase (Algorithm 32), where the following is computed:

$$\mathsf{roundkey}_0 = \lambda_{i_k} + \sum_{i \neq i_k} \lambda_i \text{ and } \lambda^{sk} = \mathsf{matMul}(\mathsf{roundkey}_0, K_0^{-1}). \tag{6.1}$$

Since $\hat{sk} = sk + \lambda^{sk}$, we have the following equation for λ_{i_k},

$$\hat{sk} = sk + (\lambda_{i_k} + \sum_{i \neq i_k} \lambda_i) K_0^{-1}, \text{ and } \lambda_{i_k} = (\hat{sk} - sk) K_0 - \sum_{i \neq i_k} \lambda_i. \tag{6.2}$$

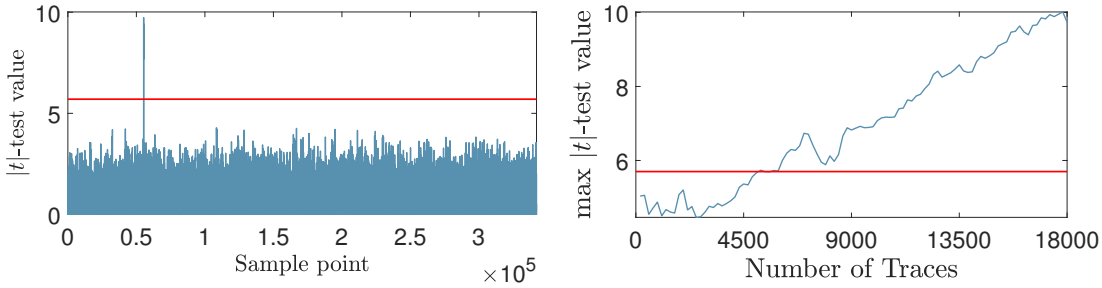


Figure 6.4: A first-order RvR test on the unprotected Picnic3 implementation using revealed values from the online phase. The traces are classified based on a single bit of $\hat{sk}K_0$. Values above the 5.7 threshold (red line) indicate there are strong leakages (left). Also, the maximum $|t|$-value increases with respect to number of traces and the leakage becomes absolute after 6,000 traces (right).

Finally, we substitute the secret value λ_{i_k} into Eq. (6.1),

$$\mathsf{roundkey}_0 = (\hat{sk} - sk)K_0 - \sum_{i \neq i_k} \lambda_i + \sum_{i \neq i_k} \lambda_i = (sk + \hat{sk})K_0 \tag{6.3}$$

From Eq. (6.3) we observe that $\mathsf{roundkey}_0$ can be probed (over multiple traces) since sk is a constant value. Then λ_{i_k} can be calculated as $\mathsf{roundkey}_0 - \sum_{i \neq i_k} \lambda_i$, and used to obtain the secret (as described above). The result of the analysis in Fig. 6.4 shows a clear dependence between the unrevealed value $\mathsf{roundkey}_0$ and the observable trace, as the $|t|$-value clearly exceeds 5.7 which shows an exploitable leakage.

7

SNI-in-the-head: Protecting MPC-in-the-head Protocols against Side-channel Analysis

7.1 Motivation

In this chapter, we study the applicability of masking techniques to protocols relying on the MPC-in-the-head paradigm and how one can prevent SCA attacks. To show the versatility of our approach, we use the ZKBoo protocol [GMO16] as an example. The main insight of our approach is that both the MPC-in-the-head approach and the masking approach to protect against SCA are MPC protocols and can thus be viewed in a unified way.

We first generalize the notion of $(2,3)$-decompositions of functions introduced in [GMO16]. This allows us to apply MPC-based *masking techniques*—which are widely used to counteract DPA attacks—in the setting of MPC-in-the-head protocols. To use MPC-based DPA protection in MPC-in-the-head protocols, we need gadgets, where a strict subset of the output variables does not reveal any information about the input variables. Formally, this requirement is captured by the notion of *strong non-interference* (SNI) [BBD+16]. While SNI gadgets are known in the literature, none of them are compatible with the function decompositions needed for ZKBoo, as the dependency between the partial functions is imbalanced. Hence, we design suitable

balanced SNI gadgets to obtain a generalized version of ZKBoo, called $(n+1)$-ZKBoo that reveals $\lceil n/2 \rceil + 1$ shares out of $n+1$. Any attacker obtaining $n - (\lceil n/2 \rceil + 1)$ additional variables thus only knows n out of $n+1$ shares and is still not able to recover the complete input. To show the feasibility of our defense mechanisms, we implemented this algorithm for $n+1 = 5$ (thus revealing three shares). Our experiments show that the extraction of a single additional variable is not sufficient to reconstruct the input. To protect against n probes, the size of the communication of $(n+1)$-ZKBoo is about $(n+1)/4$ times larger than those of the original ZKBoo, while the running time of $(n+1)$-ZKBoo is about $(n+1)(n+2)/9$ times larger than ZKBoo.

Notation. First, we summarize the notation used in the rest of the chapter. In the following, we fix some finite ring $(\mathbb{K}, \oplus, \otimes)$ with an *addition* operation $\oplus$ and a *multiplication* operation $\otimes$. As usual, we often omit the multiplication symbol $\otimes$ and thus write xy instead of $x \otimes y$. For $a, b \in \mathbb{Z}$ with $a < b$, we define $[a, b] := \{a, a+1, \ldots, b-1, b\}$. The letters $x, y, z, \ldots$ represent the sensitive variables. Random variables are represented by the letter r, with an index as r_i. To denote a random selection of a variable r from the field $\mathbb{K}$, we use $r \in_R \mathbb{K}$.

Typically, a variable x is split into $n+1$ shares $x_0, \ldots, x_n$ such that $x = \bigoplus_{i=0}^{n} x_i$. The value n is called the *masking order*. This technique of masking was popularized in [CJRR99b]. A vector of shares $(x_0, \ldots, x_n)$ is denoted by $\overline{x}$, and the underlying masked value is given by $x = \bigoplus_{i=0}^{n} x_i$. For a subset $I \subseteq [0, n]$ of indices, we denote by $x_{|I} = (x_i)_{i \in I}$ the sub-vector of shares indexed by I. A *gadget* G for a function $f \colon \mathbb{K}^a \to \mathbb{K}^b$ (with regard to a masking order) is an arithmetic circuit with $a \cdot (n+1)$ inputs and $b \cdot (n+1)$ outputs grouped into a vectors of shares $\overline{x}^{(1)}, \ldots, \overline{x}^{(a)}$, resp. b vectors of shares $\overline{y}^{(1)}, \ldots, \overline{y}^{(b)}$. The arithmetic circuits have five kinds of gates: the unary $\oplus_\alpha$ gate with $\alpha \in \mathbb{K}$, which on input x outputs $x \oplus \alpha$; the unary $\otimes_\alpha$ gate with $\alpha \in \mathbb{K}$, which on input x outputs $x \otimes \alpha$; the binary $\oplus$ gate which on inputs x, x' outputs $x \oplus x'$; the binary $\otimes$ gate, which on inputs x, x' outputs $x \otimes x'$; and the random gate with fan-in 0 that produce a uniformly chosen random element $r \in_R \mathbb{K}$. Note that in the case of $\mathbb{K} = \mathrm{GF}(2)$, these gates directly correspond to AND, XOR, and NOT gates. The gadget needs to be correct, i. e. $G(\overline{x}^{(1)}, \ldots, \overline{x}^{(a)}) = (\overline{y}^{(1)}, \ldots, \overline{y}^{(b)})$ iff $f(x^{(1)}, \ldots, x^{(a)}) = (y^{(1)}, \ldots, y^{(b)})$ for all possible inputs and for all values generated by the random gates. The values assigned to wires that are not output wires are called *intermediate variables*.

We also make use of a statistically binding commitment scheme and will denote the commitment algorithm as Comm (see e. g. [Gol07] for a formal definition). We omit the modulus operation $\bmod (n+1)$ to improve readability. Logarithms are always taken

with base 2, i. e. $\log(x) := \log_2(x)$.

7.2 Security Notion for MPC-in-the-head Protocol

We consider opened views as a part of probing values in Definition 2.1, Chapter 2. Thus we show that an additional probe *shatters* the independence of the side-channel traces and the sensitive variables.

Formally speaking, a single adversarial probe disables the simulators capability (which is defined in Definition 2.1) to simulate variables using a set of independent and uniformly chosen variables. Hence the multiplication gadget used in ZKBoo is not sufficient to guarantee SCA-resistance. Using the discussion given above, we can formally restate the simulation in Definition 2.1 as follows:

Definition 7.1 ($(t_{\mathcal{A}}, t_{\mathcal{E}})$-SNI Security for MPC-in-the-head protocol). Let G be a gadget which takes as input $n+1$ shares $(x_i)_{0\leq i\leq n}$ and outputs $n+1$ shares $(y_i)_{0\leq i\leq n}$. The gadget G is said to be $(t_{\mathcal{A}}, t_{\mathcal{E}})$-SNI secure if for any set of $t_{\mathcal{A}}$ probed intermediate variables, $t_{\mathcal{E}}$ opened variables and any subset $\mathcal{O} \subset [0, n]$ of output indices, such that $t_{\mathcal{A}} + t_{\mathcal{E}} + |\mathcal{O}| \leq t$, there exists a subset $I \subset [0, n]$ of input indices with $|I| \leq t_{\mathcal{A}}$, such that the $t_{\mathcal{A}}$ intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.

Definition 7.1 is equivalent to Definition 2.1 if $t_{\mathcal{E}} = 0$, i. e. if there exist no opened values. More formally, t-SNI implies (t_1, t_2)-SNI for all $t_1 + t_2 \leq t$. On the other hand, this leaked data might be chosen carefully such that $t_{\mathcal{E}}$ leaked output variables only give information on $t_{\mathcal{E}}/2$ input variables. In such a case, using a t-SNI gadget might actually give a $(t_{\mathcal{A}}, t_{\mathcal{E}})$ gadget with $t_{\mathcal{A}} + t_{\mathcal{E}}/2 = t$, giving a more fine-granular view.

The above definition captures the intuition that protocols following the MPC-in-the-head paradigm leak information all by themselves due to the opening of some views. Without the presence of side-channel attacks, this is not a problem, as the privacy of the underlying MPC protocol guarantees that no information about the secret is leaked. But in the presence of side-channel attacks, this leaked information can drastically help the attacker. Using the $(t_{\mathcal{A}}, t_{\mathcal{E}})$-SNI notion, we can design MPC-in-the-head protocols that achieve t-SNI security even if a subset of the views are revealed. In the next section we provide a circuit decomposition and define our protocol.

7.3 Constructing SNI-secure Decompositions

In this section, we introduce a decomposition of an arithmetic circuit secure in the SNI notion. We start with a generic decomposition definition that will be used for the circuit decomposition in the following sections. In [GMO16], the notion of a $(2,3)$-decomposition was introduced. Informally, such a decomposition splits a function ϕ into three *branches* such that the computations of two of those branches are not enough to reconstruct the complete computation of the function. In ZKBoo, two of these branches are revealed, while the third branch stays hidden. As shown in Chapter 6, this allows DPA attacks against ZKBoo, as a single probed value from this third branch might be sufficient to reconstruct the complete computation. In this section, we thus aim to construct function decompositions that are SNI-secure and withstand such probes. We first introduce a generalization of $(2,3)$-decompositions, called $(k, n+1)$-decompositions, consisting of $n+1$ branches, where each branch depends only on k branches. Defining *balanced* SNI-secure versions of the multiplication gadget and the refresh gadget allows us to construct a $(\lceil n/2 \rceil + 1, n+1)$-decomposition for functions represented by arithmetic circuits. Moreover, we show that this is n-SNI.

7.3.1 Decomposing a Function

Let $\phi\colon X \to Y$ be an arbitrary function. The protocol is performed on an input value $x \in X$ that computes $\phi(x) = y$. We assume that the computation of ϕ can be split into d steps. For example, if ϕ is implemented via a circuit, d is the number of gates. We use a transformation on the function ϕ to split the evaluation and the secret x into $n+1$ branches such that revealing n of them brings no information about the secret value x. The first step is to apply a surjective (possibly randomized) algorithm Share to x to split it into input shares $x_0, \ldots, x_n$. The input shares and the intermediate values for the i-th branch are stored in w_i, which is called a *view*, and contains $(d+1)$ elements $w_i^{(0)}, \ldots, w_i^{(d)}$. The 0-th value $w_i^{(0)}$ of a view w_i is simply its input share x_i. The single steps of the computation are described by a set of $(n+1) \cdot (d+1)$ functions $\mathcal{F} = \{\phi_i^{(j)} | \ \forall\ 0 \leq i \leq n \text{ and } 0 \leq j \leq d\}$. In order to guarantee that k views are sufficient to recompute a single branch, the functions $\phi_i^{(j)}$ take input from the k branches $i, i+1, \ldots, i+(k-1)$. The remaining values $w_i^{(j)}$ can be computed in the following iterative way:

$$w_i^{(j)} = \phi_i^{(j)}((w_m^{[0,j-1]}; R_m)_{i \leq m \leq i+(k-1)}) \text{ for } 0 \leq i \leq n,$$

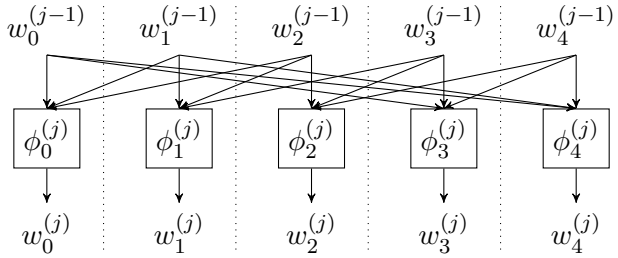


Figure 7.1: The representation of the branches for the j-th gadget ϕ^j of the $(k = 3, n = 4)$ decomposition for the function ϕ. Observe that each branch requires at most $k = 3$ views.

where $w_m^{[0,j-1]} = (w_m^{(0)}, \ldots, w_m^{(j-1)})$. Here, R_i denotes the source of randomness within the i-th branch. As an example we can see the visual representation of $\phi^{(j)}$ for $n+1 = 5$ in Fig. 7.1.

After evaluating the d functions, the output value y_i is computed from w_i by the functions Output_i, i. e. $y_i = \mathsf{Output}_i(w_i)$. Finally, the output values y_i are recombined as $\mathsf{Rec}(y_0, \ldots, y_n) = y = \phi(x)$.

Now, we can introduce the complete $(k, n+1)$-decomposition definition generalizing the definition given in [GMO16]. Note that the influence of the parameter k comes from the arity of the functions $\phi_i^{(j)}$, which take input from at most k branches $i, i+1, \ldots, i+(k-1)$.

Definition 7.2. A $(k, n+1)$-decomposition $\mathcal{D}$ of a function $\phi\colon X \to Y$ is a set of functions

$$\mathcal{D} = \{\mathsf{Share}, (\mathsf{Output}_i)_{0 \le i \le n}, \mathsf{Rec}\} \cup \mathcal{F},$$

such that Share, Output_i, Rec, and $\mathcal{F}$ are defined as above. Let Π_ϕ be the evaluation protocol defined in Fig. 7.2.

The decomposition must also have the following properties:

- **Correctness:** $Pr[\phi(x) = \Pi_\phi(x)] = 1$ for all $x \in X$, where the probability is over the random choices.
- **n-Privacy:** The protocol is correct and for all $e \in [0, n]$ there exists a PPT algorithm S_e such that the two distributions $S_e(\phi, y)$ and $(\{R_i, w_i\}_{i \in \{e, e+1, \ldots, e+(n-1)\}}, y_{e-1})$ are statistically indistinguishable.

The goal of the next subsection is the construction of $(k, n+1)$-decompositions for functions $\phi\colon \mathbb{K}^{\phi_{\text{in}}} \to \mathbb{K}^{\phi_{\text{out}}}$ implemented by an arithmetic circuit. Furthermore, we want this decomposition to be n-SNI to prevent the attacks described in Chapter 6. Note that the construction of a $(n, n+1)$-decomposition is just a simple generalization of the

Let $\phi\colon X \to Y$ be a function and $\mathcal{D}$ be a $(k, n+1)$-decomposition of ϕ. For an input $x \in X$, perform the following:

1. Generate the random tapes R_i for $0 \leq i \leq n$.
2. Generate the secret shares: $(x_0, \ldots, x_n) \leftarrow \mathsf{Share}(x; r_1, \ldots, r_n)$ where r_i is sampled from the random tape R_i.
   - Initialise $w_i^{(0)} \leftarrow x_i$ for $0 \leq i \leq n$.
   - For $1 \leq j \leq d$ compute

     $$w_i^{(j)} = \phi_i^{(j)}((w_m^{[0,j-1]}; R_m)_{i \leq m \leq i+(k-1)}) \text{ for } 0 \leq i \leq n$$

     .
3. Compute $y_i = \mathsf{Output}_i(w_i, R_i)$ for $0 \leq i \leq n$.
4. Output $y = \mathsf{Rec}(y_0, \ldots, y_n)$.

Figure 7.2: A protocol Π_ϕ using a decomposition $\mathcal{D}$ to evaluate $\phi(x)$. The figure is adapted from [GMO16].

linear $(2, 3)$-decomposition of [GMO16] and still vulnerable to the same attacks. In the next section, we will thus construct a $(k, n+1)$-decomposition for all $k \geq \lceil n/2 \rceil + 1$. These decomposition will allow to construct algorithms secure against $n - k$ probes. As $k = \lceil n/2 \rceil + 1$ gives the best security against DPAs, we focus on this case. The main technical problem to construct an $(\lceil n/2 \rceil + 1, n+1)$-decomposition is the fact that each gate/function $\phi_i^{(j)}$ can have inputs only from branches $i, i+1, \ldots, i+\lceil n/2 \rceil$. Taking a closer look at the existing construction of gadgets against side-channel attacks for multiplication (for example, the ISW gadget of [ISW03] or the more refined version of [RP10]) shows that the computation of the i-th branch depends on i other branches. These gadgets are thus not suited for our approach. To guarantee that each branch depends only on $\lceil n/2 \rceil$ other branches, we construct *balanced* gadgets.

7.3.2 Constructing Balanced Gadgets

Next we focus on the gadgets. As gates such as unary addition, unary multiplication, and binary addition are linear, there is no need for secure gadgets for these operations. We thus only need to examine the two essential SNI-secure gadgets needed for the multiplication operation. To obtain a secure multiplication operation, a *refresh gadget* is also needed, whenever a variable is used in multiple multiplication gates. See e. g. [BBC+19] for a more formal treatment.

We need to analyze and adapt these gadgets because all known SNI-secure gadgets have an unbalanced structure, which causes the need for more than $\lceil n/2 \rceil$ other views to compute some output share. Therefore, the main goal is to generate gadgets such that every branch needs at most $\lceil n/2 \rceil$ other input shares in order to compute the corresponding output share.

7.3.2.1 Balancing the Multiplication Gadget

First, we shortly review the multiplication gadget defined in [RP10] and proven to be n-SNI in [BBD+16]. Let $(x_0, \ldots, x_n)$ and $(y_0, \ldots, y_n)$ be the shares of the two sensitive variables x and y. The multiplication gadget to calculate the output shares $(z_0, \ldots, z_n)$ of $z = xy$ can be summarized in three steps as follows:

1. For $0 \leq i < j \leq n$, sample $r_{i,j} \in_R \mathbb{K}$.
2. Calculate $r_{j,i} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$ for $0 \leq i < j \leq n$.
3. Calculate $z_i = x_i y_i \oplus \bigoplus_{j=0; j \neq i}^{n} r_{i,j}$ for $0 \leq i \leq n$.

As seen in the description above, the calculation of z_i requires $n - i$ *fresh random values* ($r_{i,j}$ such that $i < j$) and i *intermediate products* ($r_{i,j}$ such that $i > j$). In order to generate a $(\lceil n/2 \rceil + 1, n)$-decomposition, we need to have a *balanced* multiplication gadget such that every index requires about the same number of random values and intermediate products.

Informally, we can illustrate the intermediate values of the multiplication gadget as a matrix $\mathtt{A}$ with $\mathtt{A}_{i,j}$ defined as (i) $x_i y_i$ for $i = j$, (ii) $r_{i,j} \in_R \mathbb{K}$ for $i < j$, and (iii) $(r_{i,j} \oplus x_i y_j) \oplus x_j y_i$ for $i > j$.

Hence we can represent the output shares as $z_i = \bigoplus_{j=0}^{n} \mathtt{A}_{i,j}$. Using this representation, $n(n+1)/2$ random values (and intermediate products) can be reorganised in such a way that each row contains at most $\lfloor n/2 \rfloor$ intermediate products. In order to do so, we define for $i \in [0, n]$ the interval J_i as follows:

- If n is even, we define $J_i = \{i + 1, \ldots, i + \lfloor n/2 \rfloor\}$.
- If n is odd and $i < (n + 1)/2$, we also define $J_i = \{i + 1, \ldots, i + \lfloor n/2 \rfloor\}$.
- Finally, if n is odd and $i \geq (n + 1)/2$, we define $J_i = \{i + 1, \ldots, i + \lfloor n/2 \rfloor, i + \lfloor n/2 \rfloor + 1\}$.

As always, modular arithmetic is used here, i. e. $|J_i| \in \{\lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1\}$ for all i. In order to generate a *balanced* multiplication gadget, one can take a partial transpose of the matrix $\mathtt{A}$ with $\mathtt{A}_{i,j}$ defined as (i) $x_i y_i$ for $i = j$, (ii) $r_{i,j} \in_R \mathbb{K}$ for $i \neq j, j \notin J_i$, and (iii) $(r_{i,j} \oplus x_i y_j) \oplus x_j y_i$ for $j \in J_i$.

As an example, consider the multiplication gadget and the balanced multiplication

$$\mathtt{A} = \begin{bmatrix} x_0y_0 & r_{0,1} & r_{0,2} & r_{0,3} & r_{0,4} \\ r_{1,0} & x_1y_1 & r_{1,2} & r_{1,3} & r_{1,4} \\ r_{2,0} & r_{2,1} & x_2y_2 & r_{2,3} & r_{2,4} \\ r_{3,0} & r_{3,1} & r_{3,2} & x_3y_3 & r_{3,4} \\ r_{4,0} & r_{4,1} & r_{4,2} & r_{4,3} & x_4y_4 \end{bmatrix} \quad \mathtt{A}' = \begin{bmatrix} x_0y_0 & r_{1,0} & r_{2,0} & r_{0,3} & r_{0,4} \\ r_{0,1} & x_1y_1 & r_{2,1} & r_{3,1} & r_{1,4} \\ r_{0,2} & r_{1,2} & x_2y_2 & r_{3,2} & r_{4,2} \\ r_{3,0} & r_{1,3} & r_{2,3} & x_3y_3 & r_{4,3} \\ r_{4,0} & r_{4,1} & r_{2,4} & r_{3,4} & x_4y_4 \end{bmatrix}$$

Figure 7.3: Example of $\mathtt{A}$ and $\mathtt{A}'$ for $n+1=5$.

gadget for $n+1=5$ shown in Fig. 7.3. The upper matrix $\mathtt{A}$ represents the multiplication gadget defined in [RP10] and matrix $\mathtt{A}'$ describes the equivalent, but balanced multiplication gadget. The parts transposed are marked in grey. We can see that in both cases $z_i = \bigoplus_{j=0}^{n} \mathtt{A}_{i,j}$ and $z'_i = \bigoplus_{j=0}^{n} \mathtt{A}'_{i,j}$. Although the shares are calculated differently i. e. $z_i \neq z'_i$, the correctness of the gadgets holds i. e. $z = xy = \bigoplus_{i=0}^{n} z_i = \bigoplus_{i=0}^{n} z'_i$. Note that row i of $\mathtt{A}'$ has exactly two fresh random values and the remaining intermediate products come from rows $i+1$ and $i+2$.
Finally, we formally introduce the *balanced* multiplication gadget to calculate the output shares $(z_0, \ldots, z_n)$ of $z = xy$ as follows:

1. For $0 \leq i < j \leq n$, sample $r_{i,j} \in_R \mathbb{K}$.
2. Calculate $r_{j,i} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$ for $0 \leq i < j \leq n$.
3. Calculate $z_i = x_i y_i \oplus \bigoplus_{j=0; j\neq i}^{n} \delta_{i,j}$ for $0 \leq i \leq n$ where $\delta_{i,j}$ is defined as (i) $r_{i,j}$ for $j \in J_i, i < j$, (ii) $r_{i,j}$ for $j \notin J_i, i < j$, (iii) $r_{i,j}$ for $j \in J_i, i > j$, and (iv) $r_{j,i}$ for $j \notin J_i, i > j$.

Remark that the balanced multiplication gadget defined above does not bring any overhead to the scheme. The explicit description can be found in Algorithm 11. Furthermore, it is easy to see that the balance is achieved.

Lemma 7.1. *In the balanced multiplication gadget, in each row i, the intermediate products $r_{i,j}$ with $i > j$ only occur at positions $\delta_{i,j}$ with $j \in J_i$.*

Proof. Consider any row i and any position $j \notin J_i$. Then, the second or fourth cases in the construction of $\delta_{i,j}$ might occur and in both cases, a fresh random element is chosen. □

In the final step, we show that the balanced multiplication gadget indeed satisfies the SNI notion, as the gadget is secure against n attack probes.

Algorithm 11 *Balanced* Multiplication Gadget

Input: The shares $(x_0, \ldots, x_n)$ and $(y_0, \ldots, y_n)$.
Output: The vector of shares of xy as $(z_0, \ldots, z_n)$.

1: **for** $0 \leq i \leq n$
2: **for** $i < j \leq n$
3: $r_{i,j} \leftarrow rand()$ // $r_{i,j} \in_R \mathbb{K}$
4: $r_{j,i} = (x_i y_j \oplus r_{i,j}) \oplus x_j y_i$
5: **for** $0 \leq i \leq n$
6: $z_i \leftarrow x_i y_i$
7: **for** $0 \leq j \leq n$ and $j \neq i$
8: **if** $(j \in J_i$ and $i < j)$ or $(j \notin J_i$ and $i > j)$ **then**
9: $z_i \leftarrow z_i \oplus r_{j,i}$ // Denoted by $z_{i,j}$
10: **else if** $(j \in J_i$ and $i > j)$ or $(j \notin J_i$ and $j < j)$ **then**
11: $z_i \leftarrow z_i \oplus r_{i,j}$ // Denoted by $z_{i,j}$
12: **return** $(z_0, \ldots, z_n)$

Theorem 7.2 (n-SNI Security for balanced multiplication gadget). *Let G be the balanced multiplication gadget which takes $(x_i)_{0\leq i\leq n}$ and $(y_i)_{0\leq i\leq n}$ as the input shares, and outputs $(z_i)_{0\leq i<n}$. For any set of $t \leq n$ intermediate variables and any subset $\mathcal{O} \subset [z_0, \ldots, z_n]$ of output shares such that $t + |\mathcal{O}| \leq n$, there exists a subset $I \subset [0, n]$ of input indices which satisfies $|I| \leq t$, such that the t intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Proof. In order to prove the theorem, we use a similar structure as in [CGPZ16] and show that every set of intermediate variables with t elements can be simulated by two sets of input shares $(x_i)_{i\in I}$ such that $|I| \leq t$ and $(y_j)_{j\in J}$ such that $|J| \leq t$. Let $z_{i,j}$ be the j-th partial sum of z_i, i. e. $z_{i,j} = x_i y_j \oplus \bigoplus_{j'=0,j'\neq i}^{j} \delta_{i,j'}$. We divide the probes in four groups:

A1: If x_i, y_i, or $x_i y_i$ is probed, add i to I and J.

A2: If $\delta_{i,j}$ or $z_{i,k}$ is probed (for $i \neq j$), add i to I and J.

Note that after these first two groups, we have $I = J$ and will denote this common set as U.

A3: If $x_i y_j \oplus r_{i,j}$ is probed, do the following: if $i \in U$ or $j \in U$, add $\{i, j\}$ to I and J.

A4: If $x_i y_j$ is probed (for $i \neq j$), add i to I and j to J.

Clearly, $|I|$ and $|J|$ have at most one index per probe, and therefore $|I| \leq t$ and $|J| \leq t$. We will now define the simulator, first for the intermediate variables.
We now go through the different groups:

A1: To simulate x_i, y_i, or $x_i y_i$, we can simply use the input variables, as both x_i and y_i are known from I and J.

A4: To simulate $x_i y_j$ we can simply use the input variables, as both x_i and y_j are known from I and J.

For the remaining groups **A2** and **A3** (i. e. probed variables $\delta_{i,j}$, $z_{i,j}$ or $x_i y_j \oplus r_{i,j}$), we split the proof into smaller claims.

Claim 7.1. *For any i, if $i \notin U$, then $\delta_{i,j}$ is not probed and does not enter in the computation of any probed partial sum $z_{i,k}$.*

Proof. The variable $\delta_{i,j}$ is used in all partial sums $z_{i,k}$ for $k \geq j$. As $i \notin U$, no partial sum $z_{i,k}$ was probed (**A2**). □

Claim 7.2. *If $\delta_{i,j}$ or $x_i y_j \oplus r_{i,j}$ were probed, we can simulate them perfectly. Furthermore, for all $i \in U$, we can simulate $\delta_{i,j}$ perfectly.*

Proof. We consider a pair $i < j$ and distinguish all four possibilities to simulate $\delta_{i,j}$ or $x_i y_j \oplus r_{i,j}$.

- If $\{i, j\} \subseteq U$, we can sample $r_{i,j}$ uniformly and calculate $x_i y_j$, $x_i y_j \oplus r_{i,j}$, $x_j y_i$, and $r_{j,i}$ perfectly, as $\{i, j\} \subseteq I \cap J$. Hence, we can simulate $\delta_{i,j}$, $\delta_{j,i}$ and $x_i y_j \oplus r_{i,j}$.
- If $i \in U$ and $j \notin U$, we know that $\delta_{j,i}$ was not probed and is not used in any probed computation $z_{j,k}$ by Claim 7.1.

  If $\delta_{i,j}$ is a fresh random value (i. e. $\delta_{i,j} = r_{i,j}$), we can just uniformly sample $\delta_{i,j} = r_{i,\prime}$. As $\delta_{j,i}$ was not probed, the only place where $\delta_{i,j}$ might occur in is $x_i y_j \oplus r_{i,j}$. If $x_i y_j \oplus r_{i,j}$ was probed, we have $i \in I$ and $j \in J$ by construction (**A3**) and $i \in U$ and can thus simulate $x_i y_j \oplus r_{i,j}$ perfectly.

  If $\delta_{i,j}$ is not a fresh random value (i. e. $\delta_{i,j} = (x_i y_j \oplus r_{i,j}) \oplus x_j y_i$), we can also sample $\delta_{i,j}$ uniformly. As $\delta_{j,i} = r_{i,j}$ was not probed, the only place where $\delta_{j,i} = r_{i,j}$ might occur in is $x_i y_j \oplus r_{i,j}$. If $x_i y_j \oplus r_{i,j}$ is probed, we have $\{i, j\} \in I \cap J$, as $i \in U$. Hence, we know $x_j y_i$ and can thus compute and output

$$x_j y_i \oplus \delta_{i,j} = x_j y_i \oplus (x_i y_j \oplus r_{i,j}) \oplus x_j y_i = x_i y_j \oplus r_{i,j}.$$

- If $i \notin U$ and $j \in U$, we know that $\delta_{i,j}$ was not probed and is not used in any probed computation $z_{i,k}$ by Claim 7.1.

  If $\delta_{j,i}$ is a fresh random value (i. e. $\delta_{j,i} = r_{i,j}$), we can just uniformly sample $\delta_{j,i} = r_{i,j}$. As $\delta_{i,j}$ was not probed, the only place where $\delta_{j,i}$ might occur is in $x_i y_j \oplus r_{i,j}$. If $x_i y_j \oplus r_{i,j}$ was probed, we have $j \in J$ and $i \in I$ by construction (**A3**) and $j \in U$ and can thus simulate $x_i y_j \oplus r_{i,j}$ perfectly.

  If $\delta_{j,i}$ is not a fresh random value (i. e. $\delta_{j,i} = (x_i y_j \oplus r_{i,j}) \oplus x_j y_i$), we can also sample $\delta_{j,i}$ uniformly. As $\delta_{i,j} = r_{i,j}$ was not probed, the only place where $\delta_{i,j} = r_{i,j}$ might occur is in $x_i y_j \oplus r_{i,j}$. If $x_i y_j \oplus r_{i,j}$ is probed, we have $\{i, j\} \in I \cap J$, as $j \in U$. Hence, we know $x_j y_i$ and can thus compute and output

  $$x_j y_i \oplus \delta_{j,i} = x_j y_i \oplus (x_i y_j \oplus r_{i,j}) \oplus x_j y_i = x_i y_j \oplus r_{i,j}.$$

- If $i \notin U$ and $j \notin U$, neither $\delta_{i,j}$ nor $\delta_{j,i}$ were probed or used in any probed computation $z_{i,k}$ or $z_{j,k}$. If $x_i y_j \oplus r_{i,j}$ was probed, we thus know that $r_{i,j}$ is not used anywhere else. Hence, we can sample a random value uniformly for $x_i y_j \oplus r_{i,j}$. □

The only remaining internal variables to simulate are the partial sums $z_{i,k}$. Whenever such a partial sum $z_{i,k}$ was sampled, (**A2**) implies that $i \in U$. Now Claim 7.2 implies that all $\delta_{i,j}$ can be simulated perfectly and thus all $z_{i,k}$.
In the last part of the proof, we consider the simulation of the subset of output shares $z_{|\mathcal{O}}$ from $x_{|I}$ and $y_{|J}$. Claim 7.2 already shows that for $i \in U$, all z_i can be simulated.
Now, consider all indices i with $i \notin U$ (including those not in $\mathcal{O}$). We construct a subset of indices V as follows: for any probed variable $x_i y_j \oplus r_{i,j}$ corresponding to (**A3**) with $i \notin U$ and $j \notin U$, we add j to V if $i \in \mathcal{O}$ or i to V if $i \notin \mathcal{O}$. Note that whenever we do not add an index to v (either due to $i \in U$ or $j \in U$), there is a probe corresponding to (**A1**) or (**A2**) responsible for this. As we have at most t probes of intermediate variables, we have $|U| + |V| \leq t$ and thus $|U| + |V| + |\mathcal{O}| \leq n$ by assumption that $t + |\mathcal{O}| \leq n$. Hence, there is at least one index $j^\star \in \{0, \ldots, n\}$ such that $j^\star \notin U \cup V \cup \mathcal{O}$.
Now, fix any $i \in \mathcal{O}$ with $i \notin U$. We can write

$$z_i = x_i y_i \oplus \bigoplus_{j=0; j \neq i}^{n} \delta_{i,j} = \delta_{i,j^\star} \oplus (x_i y_i \oplus \bigoplus_{j=0; j \neq i; j \neq j^\star}^{n} \delta_{i,j}).$$

Claim 7.3. *Neither $\delta_{i,j^\star}$ nor $\delta_{j^\star,i}$ enter in the computation of any probed variable or another $z_{i'}$ with $i \neq i'$ and $i' \in \mathcal{O}$.*

Proof. As $i \notin U$, Claim 7.1 implies that neither $\delta_{i,j^\star}$ nor any $z_{i,k}$ were probed. As $j^\star \notin U$, Claim 7.1 implies that neither $\delta_{j^\star,i}$ nor any $z_{j^\star,k}$ were probed. Now, $\delta_{i,j^\star}$ or $\delta_{j^\star,i}$ can only occur either in $z_{j^\star}$, $x_i y_{j^\star} \oplus r_{i,j^\star}$ or in $x_{j^\star} y_i \oplus r_{j^\star,i}$. As $j^\star \notin \mathcal{O}$, we do not need to simulate $z_{j^\star}$. If $i < j^\star$, then $x_i y_{j^\star} \oplus r_{i,j^\star}$ was not probed (otherwise, we would have $j^\star \in V$, as $i \in \mathcal{O}$). Similarly, if $i > j^\star$, then $x_{j^\star} y_i \oplus r_{j^\star,i}$ was not probed (otherwise, we would have $j^\star \in V$, as $j^\star \notin \mathcal{O}$). □

Hence, we can simply simulate z_i by uniformly sampling a random value. □

Note that our proof does not actually make use of the definition of $\delta_{i,j}$. Hence, we obtain the following corollary.

Corollary 7.3. Any *partial transposition of the secure multiplication gadget from [RP10] is n-SNI.*

7.3.2.2 Balancing the RefreshMask Gadget

Algorithm 12 RefreshMask Gadget [BBD+16]

Input: The vector of shares $(x_0, \ldots, x_n)$.
Output: The vector of shares of x as $(x'_0, \ldots, x'_n)$.

1: **for** $0 \leq i \leq n$
2: $\quad x'_i \leftarrow x_i$
3: **for** $0 \leq i \leq n$
4: $\quad$ **for** $i < j \leq n$
5: $\quad\quad r \leftarrow rand()$ // $r \in_R \mathbb{K}$
6: $\quad\quad x'_i \leftarrow x'_i \oplus r$
7: $\quad\quad x'_j \leftarrow x'_j \oplus r$
8: **return** $(x'_0, \ldots, x'_n)$

In the next section we focus on balancing another important gadget for SNI notion: the RefreshMask gadget. The foundation of our gadget is the gadget defined in [BBD+16] and can be found in Algorithm 12. Remark that this gadget is an essential part of the SNI notion, due to its role in composability. Informally speaking, refresh masking gadgets are used to protect circuits where a set of inputs $(x_0, \ldots, x_n)$ is used in more than one multiplication gadget. An example of such a circuit can be found in [RP10] for

Algorithm 13 Balanced RefreshMask Gadget

Input: The vector of shares $(x_0, \ldots, x_n)$.
Output: The vector of shares of x as $(x'_0, \ldots, x'_n)$.

1: **for** $0 \leq i \leq n$
2: $\quad x'_i \leftarrow x_i$
3: **for** $0 \leq i \leq n$
4: $\quad$ **for** $0 < j \leq \lceil n/2 \rceil$
5: $\quad\quad r_{i,i+j} \leftarrow rand()$ // $r_{i,i+j} \in_R \mathbb{K}$
6: $\quad\quad x'_i \leftarrow x'_i \oplus r_{i,i+j}$ // Denoted by $a_{i,i+j}$
7: $\quad\quad x'_{i+j} \leftarrow x'_{i+j} \oplus r_{i,i+j}$ // Denoted by $b_{i+j,i}$
8: **return** $(x'_0, \ldots, x'_n)$

the function $\phi(x) = x^{254}$. Thus, the usage of RefreshMask depends on the structure of the underlying circuit.
Clearly, the total number of required randomness in Algorithm 12 is $n(n+1)/2$. Remark that the indices follow modulus operation $\mathrm{mod}(n+1)$ which we omit to improve the readability. Moreover, row i also requires $n-i$ random values. Using a similar strategy as in Section 7.3.2.1, we can reformulate this gadget and generate a balanced gadget, where each index requires the same number of randomness as given in Algorithm 13.

Theorem 7.4 (n-SNI Security for Balanced RefreshMask Gadget). *Let G be the balanced RefreshMask gadget which takes $(x_i)_{0\leq i\leq n}$ and outputs $(x'_i)_{0\leq i<n}$. For any set of $t \leq n$ intermediate variables and any subset $\mathcal{O} \subset [0,n]$ of output shares such that $t + |\mathcal{O}| \leq n$, there exists a subset $I \subset [0,n]$ of input indices which satisfies $|I| \leq t$, such that the t intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Proof. In order to prove the theorem, we show that every set of intermediate variables with $t \leq n$ elements can be simulated by a set of input shares $(x)_{i\in I}$ such that $|I| \leq t$. Let us first classify the variables. The intermediate variables are x_i, $r_{i,i+j}$, $a_{i,i+j}$ and $b_{i+j,i}$ and the outputs are x'_i (or $x'_{i,i+\lceil n/2 \rceil}$).
After this now we can define I as follows: for each probed variable x_i, $r_{i,i+j}$ and $a_{i,i+j}$ add i to I and $b_{i+j,i}$ add $i+j$ to I. It is clear that I contains at most t elements since each probed value adds at most one index to I.
Now we can define the simulator. For all $i \in I$ the simulator can sample all $r_{i,i+j}$ for for $j \in [0, \lceil n/2 \rceil]$ and compute all partials sums $a_{i,i+j}$ and $b_{i+j,i}$ and thus the output x'_i.
Last, we need to consider the simulation of the output shares x'_i such that $i \notin I$. Observe that $i \notin I$ means that any random value in the partial sum of x'_i is not probed and is

not involved in a partial sum of it. Hence we can simulate x'_i by a uniformly random value. As a result any set of t probed intermediate variables and any subset $\mathcal{O} \subset [0, n]$ can simulated by $x_{|I}$ such that $|I| \leq t$. □

After introducing suitable gadgets, where each branch needs at most $\lceil n/2 \rceil$ values from other branches, we can finally introduce the complete circuit decomposition.

7.3.3 A $(\lceil n/2 \rceil + 1, n + 1)$-Decomposition for Arithmetic Circuits

Let $\phi\colon \mathbb{K}^{\phi_{\text{in}}} \to \mathbb{K}^{\phi_{\text{out}}}$ be a function implementable by an arithmetic circuit with d gates. The branches for $n + 1$ shares are initialised by the Share algorithm that on input $x \in \mathbb{K}^{\phi_{\text{in}}}$ and random values $r_1, \ldots, r_n$ produces the input shares $x_0, \ldots, x_n$ with $x_i = r_i$ for $i = 1, \ldots, n$ and $x_0 = \bigoplus_{i=1}^{n} x_i \ominus x$. The reconstruction function $\mathsf{Rec}(y_0, \ldots, y_n)$ is defined as $\mathsf{Rec}(y_0, \ldots, y_n) = \bigoplus_{i=0}^{n} y_i$.
Depending on the gates used in the arithmetic circuit, we can define the set of functions $\mathcal{F} = \{\phi_i^{(j)} | \ \forall\ i \in [0, n] \text{ and } j \in [0, d]\}$ as follows:

- If the j-th gate corresponds to an affine function $ax \oplus b$, where $a, b \in \mathbb{K}$,

$$\phi_i^{(j)} = \begin{cases} ax_i \oplus b, & \text{for } i = 0 \\ ax_i & \text{else .} \end{cases}$$

- If the j-th gate corresponds to the addition of two sensitive variables x and y, we set $\phi_i^{(j)} = x_i \oplus y_i$.

- If the j-th gate is a multiplication of two sensitive variables: x and y, we set $\phi_i^{(j)} = x_i y_i \oplus \bigoplus_{i=0}^{n} \delta_{i,j}$ for $i \neq j$ and $\delta_{i,j}$ as above Note that the fresh random values $r_{i,j}$ that are used in $\delta_{i,j}$ (i. e. $r_{i,i+\lceil n/2 \rceil+1}, r_{i,i+\lceil n/2 \rceil+2}, \ldots$) are sampled from R_i.

  If a variable x_i is *not* used for the first time in such a multiplication, we replace x_i by

$$x_i \oplus \bigoplus_{j=1}^{\lceil n/2 \rceil} r_{i,i+j} \oplus \bigoplus_{j=1}^{\lceil n/2 \rceil} r_{i-j,i}$$

  where $r_{i,j}$ is chosen as in Algorithm 13, i. e. we first apply the balanced Refresh gadget.

Finally we can define the output as $\mathsf{Output}(w_i, R_i) = w_i^{(d)}$.

Proposition 7.1. *The decomposition*

$$\mathcal{D} = \{\mathsf{Share}, (\mathsf{Output}_i)_{0 \le i \le n}, \mathsf{Rec}\} \cup \mathcal{F}$$

as defined above is an $(\lceil n/2 \rceil + 1, n+1)$*-decomposition.*

Proof. We closely follow the proof for the $(2,3)$-decomposition presented in [GMO16] and start with the correctness of our protocol.
The correctness of the decomposition follows from the masking structure. Remark that the decomposition is based on well-known masking techniques and secure gadgets which are known to be functionality preserving. Since all gadgets are correct, the complete decomposition is correct, i. e. $Pr[\phi(x) = \Pi_\phi(x)] = 1$ over all choices of randomness.
In the second part of the proof we define the simulator S_e for an index e and on inputs ϕ and y. For the sake of simplicity we define the set of indices $[e, e+n-1]$ as E and denote the last remaining index as $\tilde{e} = e - 1$.

- Sample the random tapes $(\tilde{R}_i)_{i \in E}$
- Initialise $\tilde{w}_i^{(0)}$ by sampling a random value from $\tilde{R}_i$ for $i \in E$. Then for all linear gadgets (addition and affine) calculate the values using the corresponding functions $\phi_i^{(j)}$ for all $i \in E$. If the gadget is a multiplication gadget, we do the following:
  - For all computations $\phi_i^{(j)}$ that require the view $w_{\tilde{e}}$, we randomly sample $w_i^{(j)}$.
  - For all other views, we simply compute $\phi_i^{(j)}$, since the simulation already has the knowledge of the required views.
- Calculate $\tilde{y}_i = \mathsf{Output}(\tilde{w}_i, \tilde{R}_i)$ for all $i \in E$.
- Calculate $\tilde{y}_{\tilde{e}} = y \ominus \bigoplus_{i \in E} \tilde{y}_i$
- Output $\mathcal{O} = ((\tilde{w}_i, \tilde{R}_i)_{i \in E}, \tilde{y}_{\tilde{e}})$

We can see that $\mathcal{O}$ that is outputted by S_e has the same distribution as the real values $((w_i, R_i)_{i \in E}, y_{\tilde{e}})$ provided by Π_ϕ. Observe that all the elements of S_e are calculated as the same functions in the protocol except for the multiplication gadget, when $w_{\tilde{e}}$ is needed. In this case randomly sampling the required values $w_i^{(j)}$ is a valid approach since $w_{\tilde{e}}$ contains a random value sampled from $R_{\tilde{e}}$, which is uniformly random. We can conclude that $\mathcal{D}$ has n-privacy. □

Proposition 7.2. *Let* $\mathcal{D}$ *be the* $(\lceil n/2 \rceil + 1, n+1)$*-decomposition of* $\phi \colon \mathbb{K}^{\phi_{in}} \to \mathbb{K}^{\phi_{out}}$ *as described above. Let* Π_ϕ *be the protocol described in Fig. 7.2. Then* Π_ϕ *is* n*-SNI. The*

length of each view of Π_ϕ *is* $(\phi_{in} + N_\otimes + \phi_{out}) \log(|\mathbb{K}|) + \kappa$*, where* $N_\otimes$ *is the number of multiplication gates in the arithmetic circuit implementing* ϕ*, and* κ *is the security parameter to produce the random tapes.*

Proof. Again, we follow the proof of the corresponding Proposition 4.1 in [GMO16]. All linear gadgets are n-NI by definition, Theorem 7.2 shows that our balanced multiplication gadget is n-SNI, and Theorem 7.4 shows that our balanced refresh gadget is n-SNI. As each input to a multiplication gadget is used at most once (due to the RefreshMask gadget), Lemma 3 in [BBD+15a] implies that Π'_ϕ is n-SNI.
Finally, we need to analyze the ingredients of a view in order to reveal the size of it. By definition, the function ϕ takes ϕ_{in} inputs and produces ϕ_{out} outputs where each value can be represented by $\log(|\mathbb{K}|)$ bits. Moreover the views need a security parameter κ to generate random tapes. Note that the computation between two multiplication gates can be compressed as in [GMO16]. Hence, the size of w_i can be calculated as $(\phi_{\text{in}} + N_\otimes + \phi_{\text{out}}) \log(|\mathbb{K}|) + \kappa$. □

In the next Section we provide a version of ZKBoo that can be extended to arbitrary orders and provide probing security despite of the opened views.

7.4 $(n+1)$-ZKBoo Protocol

In this section, we provide our zero knowledge proof based on the ZKBoo protocol [GMO16] that satisfies the SNI-security notion. The main idea is using the same structure of ZKBoo, but use our new $(\lceil n/2 \rceil + 1, n+1)$ circuit decomposition. Thus, our scheme can resist $n - (\lceil n/2 \rceil + 1)$ probing attacks with $\lceil n/2 \rceil + 1$ opened views.
A brief summary of the zero-knowledge proof can be described as follows. Assume that an $(\lceil n/2 \rceil + 1, n)$-decomposition for the function ϕ is given. The prover uses the private input x to run the protocol given in Fig. 7.2 that satisfies $\phi(x) = y$, where y is a public value. After running the protocol, the prover computes the commitment $\mathbf{a}$ to views $w_0 \ldots, w_n$. In the second step, the verifier challenges the prover using an index $\mathbf{e} \in [0, n]$ and the prover opens views for all w_i with $i \in [\mathbf{e}, \mathbf{e} + \lceil n/2 \rceil]$. Remark that each output share depends on at most $\lceil n/2 \rceil + 1$ consecutive views $\mathbf{z} = w_{\mathbf{e}}, \ldots, w_{\mathbf{e}+\lceil n/2 \rceil}$. Hence, opening $\lceil n/2 \rceil + 1$ views is enough to calculate each output value $w_{\mathbf{e}}^{(j)}$. Finally, the verifier accepts if the opened views are consistent with the committed values. The summary of the protocol can be found in Fig. 7.4.

Proposition 7.3. *The* $(n+1)$*-ZKBoo protocol given in Fig. 7.4 with two parties* $\mathcal{P}$ *as prover and* $\mathcal{V}$ *as verifier is a* Σ*-protocol for the relation* $\phi(x) = y$ *with* $n+1$*-special*

Figure 7.4: The $(n+1)$-ZKBoo protocol

An $(\lceil n/2 \rceil + 1, n+1)$ decomposition of function ϕ is given. The verifier and the prover have input $y \in L_\phi$. The prover knows x such that $y = \phi(x)$. Let Π_ϕ be the protocol given in Fig. 7.2.

Commit: The prover does the following:

1. Generate random tapes R_i for $0 \leq i \leq n$.
2. Run $\Pi_\phi(x)$ with randomness $R_0, \ldots, R_n$ to obtain views w_i and outputs y_i for $0 \leq i \leq n$.
3. Commit to $c_i = \mathsf{Comm}(w_i, R_i)$ for $0 \leq i \leq n$.
4. Send $\mathbf{a} = (y_i, c_i)_{0 \leq i \leq n}$.

Prove: The verifier choose an index $\mathbf{e} \in [0, n]$ and sends it to the prover. The prover answers by opening $(c_i)_{\mathbf{e} \leq i \leq \mathbf{e}+\lceil n/2 \rceil}$ thus revealing $\mathbf{z} = (R_i, w_i)_{\mathbf{e} \leq i \leq \mathbf{e}+\lceil n/2 \rceil}$.

Verify: The verifier runs the following checks:

1. If $\mathsf{Rec}(y_0, y_1, \ldots, y_n) \neq y$, output `reject`.
2. If $\exists\ i \in [\mathbf{e}, \mathbf{e} + \lceil n/2 \rceil]$ such that $y_i \neq \mathsf{Output}_i(w_i)$, output `reject`.
3. If $\exists\ j$ such that $w_i^{(j)} \neq \phi_i^{(j)}((w_k, R_k)_{\mathbf{e} \leq k \leq \mathbf{e}+\lceil n/2 \rceil})$ for all $\mathbf{e} \leq i \leq \mathbf{e} + \lceil n/2 \rceil$, output `reject`.
4. Output `accept`.

soundness.

Proof. We follow the proof of Proposition 4.2 in [GMO16]. Clearly, the $(n+1)$-ZKBoo protocol follows the communication pattern of a Σ-protocol. As the MPC-in-the-head paradigm does not change the correctness of the protocol, if both parties are honest, then $Pr[(\mathcal{P}, \mathcal{V})(y) = \texttt{accept}] = 1$. Hence, the $(n+1)$-ZKBoo protocol is complete.
In order to prove the special soundness of the protocol, we need to analyze $n+1$ accepted conversations $\{(\mathbf{a}, \mathbf{e}, \mathbf{z_e})\}$ with $\mathbf{e} = 0, \ldots, n$. Clearly, the accepted conversations reveal (R_i, w_i) for $i = 0, \ldots, n$. Thanks to the binding property of the commitment scheme, the views corresponding to the same index for different challenges are equal. That is, for two different challenges $\mathbf{z_e}$ and $\mathbf{z_{e'}}$ the views corresponding to the same index are equal, i. e. $w_i \in \mathbf{z_e}$ and $w_i \in \mathbf{z_{e'}}$ are equal. Similarly, $R_i \in \mathbf{z_e}$ also equals $R_i \in \mathbf{z_{e'}}$. As all conversations are accepted, we have $y_i = \mathsf{Output}_i(w_i)$ for $i \in [0, n]$. Moreover, we know that every entry $w_i^{(j)}$ in w_i was computed correctly by the corresponding function $\phi_i^{(j)}$, as all branches were checked by the verifier. Hence, we can traverse the decomposition bottom-up to reconstruct all input shares $x_i = w_i^{(0)}$. Finally, we can

calculate $\mathsf{Rec}(x_0, \ldots, x_n) = x$ correctly. Hence, we have $\phi(x) = y$ and the $(n+1)$-ZKBoo-protocol thus has $n+1$-special soundness.
Note that to be able to correctly calculate the input x, all the branches must be checked. Assume that the number of accepted conversations is less than $n+1$. Although the challenges might contain all views, not all branches were checked by the verifier. While we are now able to check the branches ourself, if any branch contains an error, we are not able reconstruct x.
For the special honest-verifier ZK, we will now construct a simulator S working on input $\mathbf{e}$ and $y \in L_\phi$. Its goal is to produce a triple $(\mathbf{a}', \mathbf{e}, \mathbf{z})$ with the same probability distribution as the protocol. Due to the n-privacy property of the decomposition, there is a simulator $S_{\mathbf{e}}$ that on input ϕ and y produces an output $(\{w_i, R_i\}_{i \in \{\mathbf{e}, \mathbf{e}+1, \ldots, \mathbf{e}+(n-1)\}}, y_{\mathbf{e}-1})$ distributed as in the protocol. The simulator S now sets $w_{\mathbf{e}-1}$ and $R_{\mathbf{e}-1}$ as strings of corresponding lengths that contain only zeroes. Now, S can produce commitments $c_i = \mathsf{Comm}(R_i, w_i)$ for all $i = 0, \ldots, n$ and send $\mathbf{a} = (y_i, c_i)_{0 \leq i \leq n}$. Clearly, the triple $(\mathbf{a}, \mathbf{e}, \mathbf{z})$ has the same distribution as in a real conversation, as z_e can also be easily computed. Hence, the $(n+1)$-ZKBoo-protocol has the the special honest-verifier ZK property. □

In the last part, we analyze the *soundness error* of the $(n+1)$-ZKBoo protocol which can be directly derived from special soundness. Briefly speaking, the soundness error can be summarized as the probability of a cheating prover to trick a honest verifier to accept the protocol on a value $y \notin L_\phi$.
More formally, the soundness error δ is defined as the quantity $\max_{y \notin L_\phi, \mathcal{P}'}\{\Pr[(\mathcal{P}', \mathcal{V})(y) = \texttt{accept}]\}$, where $\mathcal{P}'$ is some cheating prover. As challenge $\mathbf{e}$ is chosen uniformly at random from a set of cardinality $n+1$, the $n+1$-special soundness implies a soundness error of at most $\delta \leq (n+1-1)/(n+1) = 1 - \frac{1}{n+1}$, as for $y \notin L$, there are at most $n+1-1$ accepting conversations for each $\mathbf{a}$.
Let $\phi\colon \mathbb{K}^{\phi_{\text{in}}} \to \mathbb{K}^{\phi_{\text{out}}}$ be a function that can be expressed by an $(\lceil n/2 \rceil + 1, n)$-decomposition with N strongly non-interfering gadgets such that $N_\otimes$ of them are balanced multiplication gadgets as defined in Section 7.3. In order to attain soundness error $2^{-\kappa}$ we need to repeat the t-ZKBoo protocol k_n times such that,

$$2^{-\kappa} \geq (1 - \frac{1}{n+1})^{k_n} \Leftrightarrow k_n \geq -\kappa \cdot [\log(1 - \frac{1}{n+1})]^{-1}. \tag{7.1}$$

Similar to [GMO16], the number of bits for the opened views is

$$-\kappa \cdot [\log(1 - \frac{1}{n+1})]^{-1} \cdot \left(\lceil \frac{n}{2} \rceil + 1\right) \cdot [\log(|\mathbb{K}|)(\phi_{\text{in}} + \phi_{\text{out}} + N_\otimes) + \kappa],$$

where κ is the desired security parameter.

Theorem 7.5. *The $(n+1)$-ZKBoo protocol satisfies the $(n-(\lceil n/2 \rceil+1), \lceil n/2 \rceil+1)$-SNI notion given in Definition 7.1.*

Proof. As shown in Proposition 7.2, the evaluation protocol of the $(\lceil n/2 \rceil + 1, n + 1)$-decomposition is n-SNI. As we open exactly $\lceil n/2 \rceil + 1$ computed shares, the $(n + 1)$-ZKBoo protocol is still $n - (\lceil n/2 \rceil + 1)$-secure. □

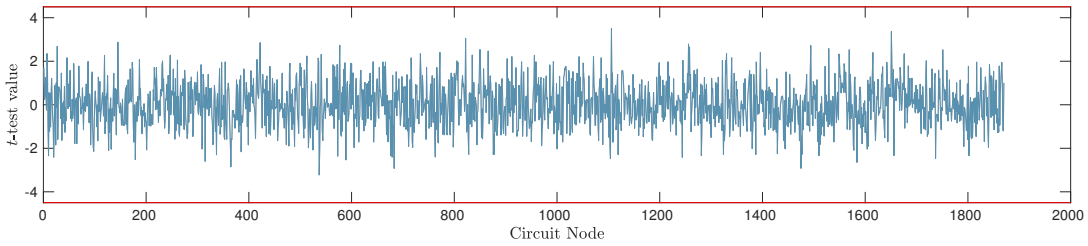
7.4.1 Experimental Results

In this section, we analyze the $(n + 1)$-ZKBoo protocol as introduced in the previous section and compare it to other instantiations from the literature that have not been adjusted to achieve SNI security. The simulation of the traces is generated by evaluating Π_ϕ using a $(3, 5)$-decomposition (i. e., $n + 1 = 5$) for a set of random inputs x and collecting the output shares of each node. Observe that the collected traces correspond to $w^{(j)}$ for all gadgets $0 \leq j \leq d$. We denote the ℓ^{th} trace (corresponding to the ℓ^{th} evaluation of the protocol) by $t_\ell = \{w_j^{(i)} \mid \text{ for all } i \in [0, n] \text{ and } j \in [0, d]\}$. Moreover, we assume that $\mathbf{e} = 0$ and collect the views of the $w_0, \ldots, w_{\lceil n/2 \rceil}$ to reflect the opened views as *free probes*.

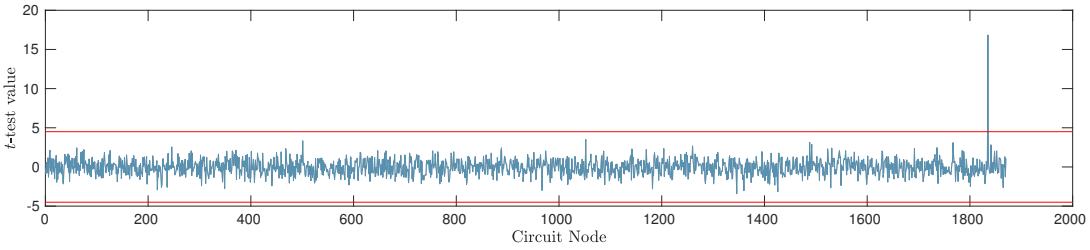
Using synthetically generated traces, we perform the test vector leakage assessment (TVLA) leakage detection method proposed by Goodwill et al. [GGJR+11].

Our analysis applies the specific t-test, where the classification of the traces relies on the opened views. As shown in Section 6.2, ZKBoo is vulnerable to side-channel attacks due to these opened views.

We analyze the 5-ZKBoo protocol which uses a $(3, 5)$-circuit decomposition. As shown in Section 7.3, the scheme is proven to be secure against first order attacks with three opened shares. We adapt the t-test and perform the classification as $\mathcal{T}_b = \{t_i | w_0^{(\alpha)} \oplus w_1^{(\alpha)} \oplus w_2^{(\alpha)} = b\}$ for $b \in \{0, 1\}$, where $w_0^{(\alpha)}$, $w_1^{(\alpha)}$ and $w_2^{(\alpha)}$ represent the opened shares during the challenge phase that correspond to the targeted gadget. The clear leakage diminishes as seen in Fig. 7.5a, where the t-value remains below 4, as expected. To compare our approach with previous unprotected approaches, we apply the same test while opening four views. Using classification $\mathcal{T}_b = \{t_i | w_0^\alpha \oplus w_1^\alpha \oplus w_2^\alpha \oplus w_3^\alpha = b\}$ we can see the resulting clear leakages in Fig. 7.5b, where the threshold value 4.5 is exceeded.



(a) TVLA-based classification using $\mathcal{T}_i$ where $i = w_0^\alpha \oplus w_1^\alpha \oplus w_2^\alpha$.



(b) TVLA-based classification using $\mathcal{T}_i$ where $i = w_0^\alpha \oplus w_1^\alpha \oplus w_2^\alpha \oplus w_3^\alpha$.

Figure 7.5: First order leakage analysis of a multiplication gadget in $(n+1)$-ZKBoo reveals no leakage with three opened shares (a), but is vulnerable with four opened shares (b). Please note the differing scales on the y-axis.

7.5 Zero-Knowledge for Post Quantum Signature Schemes

In this section, we describe the application of the $(n+1)$-ZKBoo-protocol and its performance analysis. The proposed circuit decomposition brings no overhead to the previous approaches in the sense of the masking scheme, since we are using the same gadgets with a different technique. Therefore, the number of gadgets in each branch are the same as for the circuit decomposition defined in [IKOS09] or for the MPC-in-the-head idea defined in [KKW18].

7.5.1 Picnic Scheme using $(n+1)$-ZKBoo

In this section we provide a variation of the Picnic signature scheme that can build upon our $(n+1)$-ZKBoo protocol. Picnic was introduced by Chase et al. [CDG+17]. The fundamental component of Picnic is an improved and optimized version of ZKBoo called ZKB++. The scheme uses the LowMC [ARS+15] block cipher as its underlying symmetric primitive (or ϕ in our notation). LowMC is a flexible block cipher with low AND depth especially suited for Secure Multi-Party Computation, Zero-Knowledge Proofs, or Fully Homomorphic Encryption. To make the interactive protocols ZKBoo/ZKB++ non-interactive, two methods are used: The Fiat-Shamir transformation (FS) [FS86] or

the Unruh transformation (UR) [Unr15]. We do not focus on the transformations in this work, since our idea is to modify *only* the underlying circuit decomposition.

Table 7.1 shows the number of parallel repetitions required for various decompositions, and compares them to the Picnic parameter sets. To achieve probing security for L1, L3 and L5 we need to change the underlying circuit decomposition. The required number of repetitions k_n can be calculated using Equation (7.1) and the summary can be found in Table 7.1. Due to the soundness error, the required number of repetitions to obtain the appropriate security level increases with the decomposition order. Thus, a higher number of shares implies increased soundness error at the cost of increased signature size. For example, to achieve first order protection ZKBoo using a $(2, 3)$ circuit decomposition should be replaced with 5-ZKBoo using a $(3, 5)$ circuit decomposition which results in 82% more repetitions. Similarly the cost of second order protection is 45% more repetitions compared to first order protection.

Secondly we can analyze the signature sizes and corresponding overhead. In this analysis we focus on the signature size of ZKBoo in order to make a fair comparison since ZKB++ improvements can be applied independently of the circuit decomposition structure. First we remark the signature size of ZKBoo. Let c denote the size of the commitments c_i and s denote the size of the randomness in bits used for each commitment (as in Fig. 7.4).

Table 7.1: The parameter set for the proposed circuit decomposition and the comparison between the scheme that uses standard (2,3) circuit decomposition (highlighted) with probing security order $t_{\mathcal{A}}$ and the required number of repetitions k. The LowMC parameters are key size L_n, number of s-boxes L_s, and number of rounds L_r. The hash functions in L1 are based on SHAKE-128 while L3 and L5 are based on SHAKE-256 SHA-3 functions [Dwo15] and ℓ_h represents the output length of hash functions.

Parameter Set	Decomp.	$t_{\mathcal{A}}$	κ	L_n	L_s	L_r	k	Hash/KDF	ℓ_h
Picnic-L1-FS/UR	(2,3)	0	128	128	10	20	219	SHAKE128	256
Picnic-L1-FS/UR	(3,5)	1	128	128	10	20	398	SHAKE128	256
Picnic-L1-FS/UR	(4,7)	2	128	128	10	20	576	SHAKE128	256
Picnic-L1-FS/UR	(5,9)	3	128	128	10	20	745	SHAKE128	256
Picnic-L3-FS/UR	(2,3)	0	192	192	10	30	329	SHAKE256	384
Picnic-L3-FS/UR	(3,5)	1	192	192	10	30	597	SHAKE256	384
Picnic-L3-FS/UR	(4,7)	2	192	192	10	30	864	SHAKE256	384
Picnic-L3-FS/UR	(5,9)	3	192	192	10	30	1130	SHAKE256	384
Picnic-L5-FS/UR	(2,3)	0	256	256	10	38	438	SHAKE256	512
Picnic-L5-FS/UR	(3,5)	1	256	256	10	38	796	SHAKE256	512
Picnic-L5-FS/UR	(4,7)	2	256	256	10	38	1152	SHAKE256	512
Picnic-L5-FS/UR	(5,9)	3	256	256	10	38	1507	SHAKE256	512

The ZKBoo signature size for a function $\phi\colon \mathbb{K}^{\phi_{\text{in}}} \to \mathbb{K}^{\phi_{\text{out}}}$ implemented as arithmetic circuit with $N_{\otimes}$ multiplication gates is given by [CDG+17] as:

$$\begin{aligned}|p_z| &= k_z[3(|y_i| + |c_i|) + 2((\log(|\mathbb{K}|)(\phi_{\text{in}} + \phi_{\text{out}} + N_{\otimes})) + \kappa + s)\\ &= k_z[3(\phi_{\text{out}}\log(|\mathbb{K}|) + c) + 2(\log(|\mathbb{K}|)(\phi_{\text{in}} + \phi_{\text{out}} + N_{\otimes}) + \kappa + s)]\\ &= k_z[3c + 2\kappa + 2s + \log(|\mathbb{K}|)(5\phi_{\text{out}} + 2\phi_{\text{in}} + 2N_{\otimes})],\end{aligned}$$

where k_z is the required number of repetitions as defined in Section 7.4 to achieve the desired security order κ for $n+1=3$. Using the same idea, we can calculate the signature size of $(n+1)$-ZKBoo as below. For the sake of readability, let $u = n+1$ and $v = \lceil n/2 \rceil + 1$:

$$\begin{aligned}|p| &= k_n[u(|y_i| + |c_i|) + v((\log(|\mathbb{K}|)(\phi_{\text{in}} + \phi_{\text{out}} + N_{\otimes})) + \kappa + s)\\ &= k_n[u(\phi_{\text{out}}\log(|\mathbb{K}|) + c) + v(\log(|\mathbb{K}|)(\phi_{\text{in}} + \phi_{\text{out}} + N_{\otimes}) + \kappa + s)]\\ &= k_n[uc + v\kappa + vs + \log(|\mathbb{K}|)((u+v)\phi_{\text{out}} + v\phi_{\text{in}} + vN_{\otimes})],\end{aligned}$$

where k_n is the required number of repetitions as defined in Section 7.4. Clearly the overhead of the higher decomposition is the number of opened views. Here, ZKBoo needs 2 views, while our decomposition requires $\lceil n/2 \rceil + 1$ views. Thus, replacing the underlying (2,3)-ZKBoo decomposition with a (3,5)-circuit decomposition roughly doubles the size of the signature.

In order to visualize this, we use the $|p|$-formula to compare various protocols. We assume that 128-bit security is required, corresponding to L1 ($\kappa = 128$) and ϕ function is selected as $\phi\colon \text{GF}(2)^{128} \to \text{GF}(2)^{128}$. As given in Fig. 7.6, the size of the signature naturally increases with the circuit decomposition order and the number of multiplication within ϕ. However, the size is still smaller than the signature size of $(n, n+1)$ decomposition as a result of opening only $\lceil n/2 \rceil + 1$ views instead of n views.

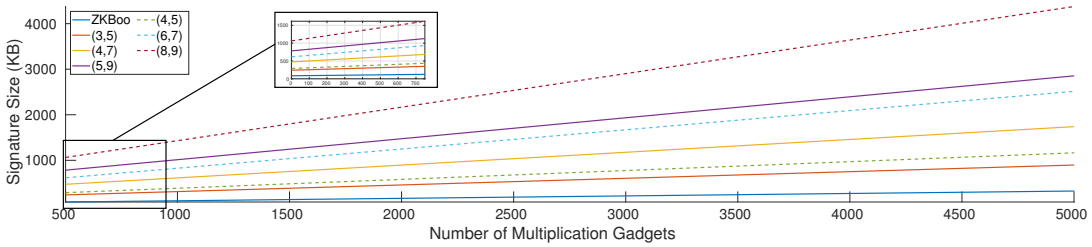


Figure 7.6: The Signature size comparison with respect to number of multiplication gadgets.

In order to compare the running times of ZKBoo with $(n+1)$-ZKBoo, we give an approximate formula involving the running time T_{rand} to generate a random tape, the running time T_{comm} to compute a single commitment, and the running time T_{mult} of a single multiplication gate. In the original ZKBoo, the prover first creates 3 random tapes and later computes the commitment to 3 shares. Within the evaluation of $\Pi_\phi(x)$, each multiplication gate of the original circuit for ϕ is replaced by 3 multiplication gates in the $(2,3)$-decomposion. As there are 3 branches, the total running time of the prover is proportional to $3 \cdot (T_{\mathsf{rand}} + T_{\mathsf{comm}} + 3N_\otimes T_{\mathsf{mult}})$, where $N_\otimes$ is the number of multiplication gates in the circuit computing ϕ. The verifier in ZKBoo just needs to verify the computation of 2 branches. Hence, its running time is proportional to $6N_\otimes T_{\mathsf{mult}}$.

In $(n+1)$-ZKBoo, the prover creates $n+1$ random tapes and computes $n+1$ commitments. Furthermore, within each of the $n+1$ branches, each multiplication gate of the original circuit for ϕ is replaced by $n+2$ multiplication gates on average in the $(\lceil n/2 \rceil + 1, n+1)$-decomposition (see Section 7.3), as $2 \cdot [(n+1)(n+2)/2]$ multiplications are computed in total. Hence, the total running time of the prover is proportional to $(n+1) \cdot (T_{\mathsf{rand}} + T_{\mathsf{comm}} + (n+2)N_\otimes T_{\mathsf{mult}})$, where $N_\otimes$ is the number of multiplication gates in the circuit computing ϕ. The verifier in $(n+1)$-ZKBoo just needs to verify the computation of $\lceil n/2 \rceil + 1$ branches. Hence, its running time is proportional to $(\lceil n/2 \rceil + 1) \cdot (n+2) N_\otimes T_{\mathsf{mult}}$.

Assuming that $N_\otimes T_{\mathsf{mult}}$ dominates T_{rand} and T_{comm}, we have a multiplicative overhead of $(n+1)(n+2)/9$ for the prover and $(\lceil n/2 \rceil + 1)(n+2)/6$ for the verifier. In the case of $n+1 = 5$, this is an overhead of $30/9 \approx 3.34$ for the prover and 3 for the verifier.

The costs of a naive solution. In a more naive approach, to withstand $n' = n - (\lceil n/2 \rceil + 1)$ probes, the prover could simply split up each of its 3 branches into $n'+1$ branches each. Each branch $j \in \{0,1,2\}$ creates the random tapes $R_j^{(0)}, \ldots, R_j^{(n')}$ used by the individual branches. Now, the MPC protocol is performed on these $3(n'+1)$ parties and the outputs $y_j^{(i)}$ and the views $w_j^{(i)}$ are computed for $j = 0,1,2$ and $i = 0,1,\ldots,n'$. The signature now contains the $3(n'+1)$ commitments $c_j^{(i)} = \mathsf{Comm}(w_j^{(i)}, R_j^{(i)})$ and the $2(n'+1)$ opened views $w_{\mathbf{e}}^{(i)}$ and $w_{\mathbf{e}+1}^{(i)}$.

Hence, the signature size of this solution would be

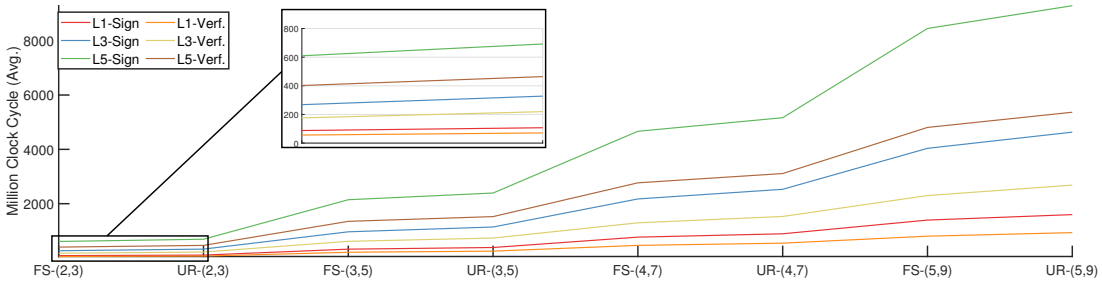
$$k_{3(n'+1)}[3(n'+1)c + 2(n'+1)\kappa + 2(n'+1)s + \log(|\mathbb{K}|)(5(n'+1)\phi_{\text{out}} + 2(n'+1)\phi_{\text{in}} + 2(n'+1)N_\otimes)].$$

Note that, compared with the signature size of our solution, *every* parameter c, κ, s, ϕ_{ou}, ϕ_{in}, and $N_{\otimes}$ has a coefficient that is at least an additive term of $n/2$ larger in the naive solution, as $3(n'+1) \approx (3/2)n$. Furthermore, the soundness error of the naive solution also relates to $(3/2)n$ parties instead of n parties. The running time of the prover would be proportional to $3(n'+1)\cdot(T_{\mathsf{rand}}+T_{\mathsf{comm}}+3(n'+1)N_{\otimes}T_{\mathsf{mult}})$ and the running time of the verifier would be proportional to $6(n'+1)^2N_{\otimes}T_{\mathsf{mult}}$. As in the case of the signature size, *every* coefficient is worse compared with our solution. Note that similar optimizations as used in the construction of ZKB++ [CDG+17] can be made here as well and will shrink the signature size, e. g. by recombining the opened views. Nevertheless, the number of commitments in the signature will still be proportionate to n', as the prover cannot recombine the unopened views without introducing a leakage. We also remark that an algorithm resistant against side-channel attacks without increased signature size is always possible by doing the complete evaluation of the algorithm via an MPC protocol. Clearly, the running time of such an approach is much larger compared to the SNI-in-the-head approach or the naive approach.

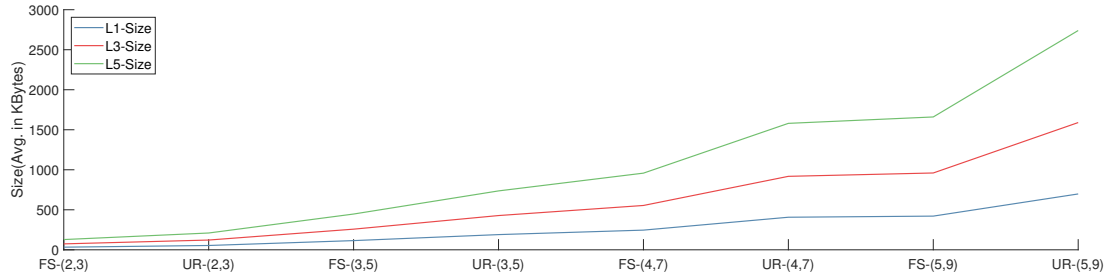
7.5.2 Performance Results

In this section we give the experimental performance analysis of our scheme. We adapted the Picnic source code of the reference implementation [Ste] to implement the $(n+1)$-ZKBoo scheme. We compare the results of our scheme with the Picnic implementation with the parameters L1, L3 and L5 using both transformation i. e. UR or FS. The benchmarking is done on AMD Ryzen Threadripper 1950X CPU@3.4 GHz. The experiments covers the total clock cycle count while signing a message or verifying a signature including the size of the signature. The results are computed by taking the average over 500 signature generation and verification. The summary of the analysis can be found in Fig. 7.7 and the exact numbers are given in Table 7.2.

As given in Table 7.2, we calculate the overhead of our scheme. The first order security increases the average number of clock cycles by a factor of 3.28-3.78 and the average size of the signature by 3.49-3.51 depending on the security parameter and transformation method. This factor increases with the security order as expected. The optimized Picnic implementation [Seb] achieves a speedup factor of 6 compared to [Ste], which we also expect for our protocol.



(a) The average number of clock cycles to sign or verify.



(b) The average size of a signature in bytes.

Figure 7.7: The benchmarking results of Picnic signature scheme using $(3, 5), (4, 7), (5, 9)$-ZKBoo decomposition with UR or FS transformations. The values are calculated by averaging the results of 500 signature generations/verifications.

7.6 Conclusion

As given in Chapter 6 the MPC-in-the-head protocols are indeed susceptible to SCA. However, as popular side-channel countermeasures also build on MPC principles, we show how to adapt MPC-in-the-head protocols to make them side-channel resistant by simply adjusting the underlying MPC protocol.

More recently, the MPC-in-the-head approach has also been applied to protocols having an offline or preprocessing phase [KKW18], which is for example used in the most recent version of Picnic called Picnic3 [KZ20]. As the preprocessing phase is not an MPC protocol, it must be secured independently. Classifying which kind of preprocessing phases are allowed by the approach of [KKW18] and obtaining a similar generic SNI-approach for this phase is a natural follow-up work and we focus this issue in the next chapter.

Table 7.2: The benchmarking results of Picnic signature scheme using $(3,5),(4,7),(5,9)$-ZKBoo decomposition with the transformations FS. The values are calculated by averaging the results of 500 signature generations/verifications. The overhead is calculated by comparing the respected values with the scheme using $(2,3)$ decomposition which corresponds to Picnic signature scheme (highlighted). The execution times are measured in millions of clock cycles and size calculated in kB.

	Benchmarking				Overhead		
Parameter	Decomp	Sign (t_s)	Verify (t_v)	Size (s)	$t_s/t_s^{(2,3)}$	$t_v/t_v^{(2,3)}$	$s/s^{(2,3)}$
Picnic-L1-FS	(2,3)	88	56	33	-	-	-
Picnic-L1-FS	(3,5)	326	212	115	3.72	3.78	3.49
Picnic-L1-FS	(4,7)	769	466	246	8.77	8.31	7.48
Picnic-L1-FS	(5,9)	1400	804	421	15.9	14.3	12.8
Picnic-L3-FS	(2,3)	268	176	74	-	-	-
Picnic-L3-FS	(3,5)	964	614	258	3.60	3.49	3.49
Picnic-L3-FS	(4,7)	2170	1300	554	8.11	7.37	7.47
Picnic-L3-FS	(5,9)	4040	2300	960	15.1	13.1	13
Picnic-L5-FS	(2,3)	610	402	128	-	-	-
Picnic-L5-FS	(3,5)	2150	1350	447	3.52	3.36	3.49
Picnic-L5-FS	(4,7)	4670	2770	958	7.65	6.88	7.48
Picnic-L5-FS	(5,9)	8450	4810	1660	13.9	12	13

8

Side-Channel Protections for Picnic Signatures

8.1 Motivation

When applied to Picnic, the SNIitH approach changes the number parties and the number of opened views, and the result is essentially a different parameter set. Accordingly, the verification algorithm has to be modified, and now depends on the masking order. For example, the signer would have to simulate a five-party MPC to achieve first order protection (i.e., $N = 5$, $t = 1$). Hence, the verifier needs to check the consistency of 3-out-of-5 views, instead of 2-out-of-3 as in the original Picnic/ZKB++ scheme. This is a major drawback in practice, since the introduction of side-channel protection would immediately break interoperability with existing verification algorithms. In order to support signers with varying levels of side-channel protections, verifiers would have to be prepared to accept multiple combinations of parameters (N, t). A more flexible design would allow different signers to set their level of side-channel protection independent of the verification software and costs. Moreover, the SNIitH approach inevitably sacrifices the *soundness* of underlying ZK proof system, since a malicious prover has a higher cheating probability when the number of views the verifier checks is smaller than $N - 1$. Therefore to maintain the same security level as in the original proof system, the SNI-in-the-head approach must simulate more MPC instances, leading to larger signature sizes, and slower signing and verification times. In this chapter we

design countermeasures without these drawbacks (as has been done for some other PQ signature schemes [BBE+18, MGTF19, BBE+19, GR19]).

In Section 8.2 we show how to mask the KKW proof system for any masking order. Our first observation is that all building blocks in both the offline and online computations can be masked with existing SNI-secure gadgets *without changing the number of opened parties*. The main technical difficulty is that the prover opens a different subset of sensitive information, depending on the challenge. Nevertheless, we are able to present a solution achieving strong non-interference of the basic building blocks of the KKW proof system. Then together with a known generic composition result [BBD+16], we prove that our masked KKW prover algorithm for the commit and response phases satisfies *non-interference with public output (NIo)* security [BBE+18] for all possible public outputs. The CPU cost of masking follows the common pattern where nonlinear operations have cost $O(T^2)$, and all other operations are linear in T, where T denotes the masking order. In addition to giving formal proofs of the (strong) non-interference of our solution, we also use the `maskVerif` tool [BBD+15b, BBD+16] to verify our conclusions. We stress that our variant outputs a proof *identical* to that of the unprotected KKW, so it is interoperable with existing verification operations.

Following our generic masking technique from Section 8.2, we present an NIo-secure masked implementation of Picnic3 for NIST security level L1 with first-order protection, and we report concrete experimental results on the low-end ARM Cortex-M4 STM32F4 platform (see Section 8.4). The complete specification of NIo-secure Picnic3 is presented in Section C.2.1. The figures were collected from a port of the optimized version of the Picnic3 reference code, with the help of the `pqm4` [KRSS] framework. However, the overhead incurred by provably secure masking is expensive – signing time is increased by 5.5x. Using this provably secure implementation as a baseline, we performed a number of optimizations described in Section 8.3, especially to improve hashing performance. Since a large part of the signing time in Picnic3 is spent hashing (e.g., 71% on the M4), we carefully analyzed which hashing operations must be masked, and found that the majority may remain unmasked. We then show that since all calls have either a non-sensitive input or output, a weaker form of masking is sufficient (under a mild assumption), further reducing the cost. Taken together, these optimizations reduce the cost of protected hashing significantly, and we observed overhead ranging from 1.8x to 2.8x (depending on the type of protections applied to hash function invocations). Since our hash function optimizations do void the provable side-channel-security of the signature scheme implementation as a whole, we experimentally verified the absence of leakage, to support our arguments that security is maintained in practice.

In this work, we use `maskVerif` to check SNI security of the basic components (namely, multi-party computation of multiplication in both online and offline phases) of the KKW proof system up to order 4. Although we further provide the verification scripts for the orders 5, 6 and 7, we did not run these, due to higher combinatorial complexity `maskVerif`. However, our manual security analysis in Section 8.2 indeed guarantees SNI security of higher-order masking. Furthermore, NIo-security of our fully masked Picnic3 (see Section C.2.1 for the specification) was verified with `maskVerif` up to order 2.
We also implemented a masked version of SHA-3, optimized with M4 assembly, as none was freely available. As SHA-3 is common to many PQ schemes and the M4 is a common embedded target, we expect this will be of independent interest. The implementation supports a range of options, from slower but provably SNI-secure, to our much faster optimized options. We experimentally verified that there was an absence of leakage in the optimized version.
Our first order masked Picnic3/SHA-3 implementations as well as formal verification scripts are publicly available under `crypto_sign/picnic3l1/masked/` of the Picnic-M4 GitHub repository.

https://github.com/dkales/picnic_m4

Notation. In the following, we fix some finite field $(\mathbb{F}, 0, 1, +, -, \cdot)$.[1] As explained above, we are working in the t-probing model which allows an attacker to obtain the value of t variables per run of the primitive. The most common technique to mitigate side-channel attacks is by *encoding* sensitive variables via an additive (or polynomial-based) secret sharing into $T > t$ parts. We say that a vector $(v_j)_{j\in[T]} \in \mathbb{F}^T$ is a T-*encoding* of $v := \sum_{j\in[T]} v_j$. For readability, we often write $\langle v \rangle$ instead of $(v_j)_{j\in[T]}$. For a subset $I \subseteq [T]$, let $\langle x \rangle_I = (x_i)_{i\in I}$ and furthermore $\overline{I} = [T] \setminus I$. Variables are shared both to protect against side-channel attacks and as part of the MPC protocol. To distinguish between these situations, we call a sharing between parties in the MPC protocol a *sharing* and an *encoding* when the goal is to protect against side-channel attacks.
Without loss of generality, we only give the security definitions for circuits that receive a single encoded input $\langle x \rangle$ and produce a single encoded output $\langle y \rangle$. In the following, we use the terms *circuit* and *gadget* interchangeably. Consider a (possibly randomized) gadget G, which on input x produces a value y according to some probability distribution G_x. To ensure that the computation of G does not leak any information, we modify it into

[1] While ISW-style masking countermeasures are known to work in a ring setting, we focus on a field case since previous KKW-style protocols [KKW18, dDOS19, BN20, KZ20] are defined for circuits over a field.

a gadget G' that takes $\langle x \rangle$ as input and outputs $\langle y \rangle$ with $\Pr[G'_{\langle x \rangle} = \langle y \rangle] = \Pr[G_x = y]$. Informally, we want to argue that the t probes made by an attacker do not reveal any information about the sensitive input x. Assume that the attacker probes the values $\langle v^{(1)} \rangle_{I_1}$, $\langle v^{(2)} \rangle_{I_2}$, ...$\langle v^{(k)} \rangle_{I_k}$ with $\sum_{j=1}^{k} |I_j| \leq t$. If $\langle x \rangle$ is a sufficiently random encoding (e. g. uniformly sampled) of x, then $\langle x \rangle_I$ does not reveal any information about x for all $I \subsetneq [T]$. Now, if there exists a set $I \subsetneq [T]$ such that all $\langle v^{(j)} \rangle_{I_j}$ can be simulated from $\langle x \rangle_I$, this implies that the $\langle v^{(j)}_{I_j} \rangle$ do not contain *any* information about x.

8.2 Masking Three-Round KKW

In this section we present our masked proof system following the three-round KKW protocol. We first strive for a provably NI-secure algorithm without specifying any particular circuit. In Section 8.3 we describe more concrete operations tailored to the LowMc circuit of Picnic3, and optimize by partially unmasking several hash computations (and discuss the security implications). The circuit in this presentation is a generic circuit C such that $C((w)_{w \in \mathsf{IN}}) = 1$, where each w is seen as an input wire value to the circuit and $|C|$ represents the number of multiplication gates in a circuit C.

In the description below, we additively secret share some variables in two dimensions: a share held by each party (indexed by $i \in [N]$)[2], further shared T times within each party (indexed by $j \in [T]$). For example, λ^x_i denotes a share of of λ^x held by the i-th party such that $\lambda^x = \sum_{i \in [N]} \lambda^x_i$; the value $\lambda^x_{i,j}$ for $j \in [T]$ denote shares such that $\lambda^x_i = \sum_{j \in [T]} \lambda^x_{i,j}$. The shares λ^x_i are as in the KKW protocol, and the extra T-wise encoding of this value is required for SNI security. Note that we only apply the notation $\langle \cdot \rangle$ (see Chapter C) on the T-wise encoding required for the masking countermeasure and never for the sharing of the KKW protocol. The functions requiring T-th order masked computation are marked in orange (e.g., $\langle y \rangle \leftarrow \mathsf{H}(\langle x \rangle)$ indicates that a hash function H is masked). Most of the randomness used in the protocol is from the random tapes of the parties; this randomness is derived from a seed, so that part of it may be efficiently communicated to the verifier (by sending the seed). Some of the masked operations will require additional randomness (e.g., to refresh a secret encoding), and this is sampled from the platform random number generator (RNG), since it is only required by the signer.

[2] We denote the set $\{1, \ldots, N\}$ by $[N]$.

Algorithm Masked KKW Prover

Inputs The prover holds a circuit C as a statement and an encoded witness $\langle \mathbf{w} \rangle = \{\langle w \rangle\}_{w \in \mathsf{IN}}$ such that $C(\mathbf{w}) = 1$. Values M, N, τ are parameters of the protocol.

Commit For each $k \in [M]$, the prover does:

1. Choose uniform $\langle \mathsf{seed}^{(k)} \rangle$ and use to generate values $\langle \mathsf{seed}_1^{(k)} \rangle, \ldots, \langle \mathsf{seed}_N^{(k)} \rangle$. Also the prover computes $\langle \mathsf{aux}^{(k)} \rangle \in \mathbb{F}^{|C|}$ as in Algorithm 14. For all $i = 1, \ldots N-1$, let $\langle \mathsf{state}_i^{(k)} \rangle = \langle \mathsf{seed}_i^{(k)} \rangle$ and let $\langle \mathsf{state}_N^{(k)} \rangle = \langle \mathsf{seed}_N^{(k)} \rangle || \langle \mathsf{aux}^{(k)} \rangle$.
2. Commit to the offline phase:
$$\langle \mathsf{com}_i^{(k)} \rangle = \mathsf{H}(\, \langle \mathsf{state}_i^{(k)} \rangle \,) \text{ and reconstruct } \mathsf{com}_i^{(k)} \text{ for all } i \in [N]$$
$$\mathsf{com\_off}^{(k)} = \mathsf{H}(\mathsf{com}_1^{(k)}, \ldots, \mathsf{com}_N^{(k)}).$$
3. Compute encodings of masked witness $\langle \hat{w}^{(k)} \rangle = \langle \lambda_1^w \rangle + \ldots + \langle \lambda_N^w \rangle + \langle w \rangle$ for each $w \in \mathsf{IN}$, where $\langle \lambda_i^w \rangle$ is the randomness used to mask the witness and is read from the random tape defined by $\langle \mathsf{state}_i^{(k)} \rangle$.
4. Simulate the online phase of the N-party protocol as in Algorithm 15 and produce $\langle \mathsf{msgs}_1^{(k)} \rangle, \ldots, \langle \mathsf{msgs}_N^{(k)} \rangle$.
5. Commit to the online phase:
$$\langle \mathsf{com\_on}^{(k)} \rangle = \mathsf{H}(\, \{\langle \hat{w}^{(k)} \rangle\}_{w \in \mathsf{IN}}, \langle \mathsf{msgs}_1^{(k)} \rangle, \ldots, \langle \mathsf{msgs}_N^{(k)} \rangle \,)$$
and reconstruct $\mathsf{com\_on}^{(k)}$.
6. The prover refreshes the encoding of witness $\langle w \rangle = \mathsf{RefreshM}(\, \langle w \rangle \,)$ for each $w \in \mathsf{IN}$.

Compute $h_{\mathsf{off}} = \mathsf{H}(\mathsf{com\_off}^{(1)}, \ldots, \mathsf{com\_off}^{(M)})$ and $h_{\mathsf{on}} = \mathsf{H}(\mathsf{com\_on}^{(1)}, \ldots, \mathsf{com\_on}^{(M)})$ and send $h^* = \mathsf{H}(h_{\mathsf{off}}, h_{\mathsf{on}})$ to the verifier.

Challenge The prover receives the following challenges from the verifier: a uniform τ-sized set $\mathcal{C} \subset [M]$ and $\mathcal{P} = \{i_k\}_{k \in \mathcal{C}}$ where each $i_k \in [N]$ is uniform.

Response For each $k \in [M] \setminus \mathcal{C}$, the prover sends reconstructed $\mathsf{seed}^{(k)}$ and $\mathsf{com\_on}^{(k)}$ for all to the verifier. For each $k \in \mathcal{C}$, the prover sends reconstructed values $\mathsf{com}_{i_k}^{(k)}$, $\{\mathsf{state}_i^{(k)}\}_{i \neq i_k}$, $\{\hat{w}^{(k)}\}_{w \in \mathsf{IN}}$ and $\mathsf{msgs}_{i_k}^{(k)}$ to the verifier.

Algorithm 14 masked_offline

Input: $(\langle \mathsf{seed}_i \rangle)_{i \in [N]}$.
Output: $\langle \mathsf{aux} \rangle$.

for each input wire w of the circuit
  read $\lambda_{i,j}^w$ from $\mathsf{seed}_{i,j}$
for each gate in C with input wires x and y, and output wire z:
  if ADD **then**
    compute $\langle \lambda_i^z \rangle \leftarrow \langle \lambda_i^x \rangle + \langle \lambda_i^y \rangle$
  if MUL **then**
    compute $\langle \lambda^x \rangle \leftarrow \sum_{i \in [N]} \langle \lambda_i^x \rangle$
    compute $\langle \lambda^y \rangle \leftarrow \sum_{i \in [N]} \langle \lambda_i^y \rangle$
    compute $\langle \lambda^{xy} \rangle \leftarrow \mathsf{SMul}(\, \langle \lambda^x \rangle, \langle \lambda^y \rangle \,)$
    for each $i \in [N-1]$ and $j \in [T]$
      read $\lambda_{i,j}^{xy}$ from $\mathsf{seed}_{i,j}$
    compute $\langle \lambda_N^{xy} \rangle \leftarrow \langle \lambda^{xy} \rangle - \sum_{i \in [N-1]} \langle \lambda_i^{xy} \rangle$
    for $j \in [T]$
      update aux_j with $\lambda_{N,j}^{xy}$
return $\langle \mathsf{aux} \rangle$.

Algorithm 15 masked_online

Input: Circuit C, $\langle \hat{w} \rangle$ for each input wire w of the circuit and $(\langle \mathsf{state}_i \rangle)_{i \in [N]}$.
Output: $(\langle \mathsf{msgs}_i \rangle)_{i \in [N]}$

for each gate in C with input wires x and y, and output wire z:
  if ADD **then**
    compute $\langle \hat{z} \rangle \leftarrow \langle \hat{x} \rangle + \langle \hat{y} \rangle$
  if MUL **then**
    for each $i \in [N]$
      read $\lambda_{i,j}^x, \lambda_{i,j}^y, \lambda_{i,j}^z, \lambda_{i,j}^{xy}$ from $\mathsf{state}_{i,j}$
      compute $\langle a_i \rangle \leftarrow \mathsf{SMul}(\, \langle \hat{x} \rangle, \langle \lambda_i^y \rangle \,)$
      compute $\langle b_i \rangle \leftarrow \mathsf{SMul}(\, \langle \hat{y} \rangle, \langle \lambda_i^x \rangle \,)$
      compute $\langle s_i \rangle \leftarrow \langle \lambda_i^z \rangle - \langle \lambda_i^{xy} \rangle - \langle a_i \rangle - \langle b_i \rangle$
      update $\mathsf{msgs}_{i,j}$ with $s_{i,j}$
    compute $\langle c \rangle \leftarrow \mathsf{SMul}(\, \langle \hat{x} \rangle, \langle \hat{y} \rangle \,)$
    compute $\langle s \rangle \leftarrow \sum_{i \in [N]} \langle s_i \rangle$
    compute $\langle \hat{z} \rangle \leftarrow \langle c \rangle + \langle s \rangle$
for each output wire z of the circuit :
  update $\mathsf{msgs}_{i,j}$ with $\lambda_{i,j}^z$
return $(\langle \mathsf{msgs}_i \rangle)_{i \in [N]}$.

Figure 8.1: Our masked version of 3-round KKW prover.

8.2.1 Masked Operations

We present our masked version of the KKW prover in Fig. 8.1. The function masked_offline in Algorithm 14 computes the offline phase Π_C^{off} from Section 5.3.2 in a straightforward way. The function masked_online in Algorithm 15 corresponds to a masked version of Π_C^{on}. The SNI-secure multiplier SMul( ) is defined in Algorithm 28. Note that SMul( ) is t-SNI with uniform output-distribution, as evident from the proof that it is t-SNI [BBD+16, Proposition 2]. For ADD gates, the only change is to work on T-encodings $\langle \hat{x} \rangle$, $\langle \hat{y} \rangle$, rather than on $\hat{x}$, $\hat{y}$ directly. Interestingly, the MUL gates can also be computed with a straight-forward adaptation by also encoding the masks λ^x, λ^y, λ^z, and λ^{xy}. Recall that $\hat{x} = x + \lambda^x$, and λ^x is shared among the parties, where party i has share λ_i^x (resp. λ_i^y, λ_i^z, and λ_i^{xy}). Each party thus stores their share λ_i^x as a T-encoding $\langle \lambda_i^x \rangle$ (resp. $\langle \lambda_i^y \rangle$, $\langle \lambda_i^z \rangle$, and $\langle \lambda_i^{xy} \rangle$). Each party's broadcast value s_i will now also consist of the T-encoding $\langle s_i \rangle$ as follows:

$$\langle s_i \rangle = \langle \lambda_i^z \rangle - \langle \lambda_i^{xy} \rangle - \text{SMul}(\, \langle \hat{x} \rangle, \langle \lambda_i^y \rangle \,) - \text{SMul}(\, \langle \hat{y} \rangle, \langle \lambda_i^x \rangle \,).$$

8.2.2 Security Analysis

We employ the definition of non-interference by [BBD+16] which guarantees security against t probes for $t < T$ as proposed by Ishai et al. [ISW03]. We use a more polished security notion known as t-non-interference (t-NI) and t-strong non-interference (t-SNI) defined in Section 4.3.

Algorithm 16 Masked KKW_MUL (offline)

Input: The masks of x: $\langle \lambda_i^x \rangle$ for $i \in [N]$,
The masks of y: $\langle \lambda_i^y \rangle$ for $i \in [N]$,
The auxiliary shares of xy: $\langle \lambda_i^{xy} \rangle$ for $i \in [N-1]$,
Output: The final auxiliary share of xy: $\langle \lambda_N^{xy} \rangle$

1: **compute** $\langle \lambda^x \rangle \leftarrow \sum_{i \in [N]} \langle \lambda_i^x \rangle$
2: **compute** $\langle \lambda^y \rangle \leftarrow \sum_{i \in [N]} \langle \lambda_i^y \rangle$
3: **compute** $\langle \lambda^{xy} \rangle \leftarrow \text{SMul}(\, \langle \lambda^x \rangle, \langle \lambda^y \rangle \,)$
4: **compute** $\langle \lambda_N^{xy} \rangle \leftarrow \langle \lambda^{xy} \rangle - \sum_{i \in [N-1]} \langle \lambda_i^{xy} \rangle$
5: **return** $\langle \lambda_N^{xy} \rangle$.

Algorithm 17 Masked KKW_MUL (online)

Input: The input shares of $\hat{x}$: $\langle \hat{x} \rangle$
The masks of x: $\langle \lambda_i^x \rangle$ for $i \in [N]$,
The input shares of $\hat{y}$: $\langle \hat{y} \rangle$,
The masks of y: $\langle \lambda_i^y \rangle$ for $i \in [N]$,
The auxiliary shares of xy: $\langle \lambda_i^{xy} \rangle$ for $i \in [N]$,
The output masks of z: $\langle \lambda_i^z \rangle$ for $i \in [N]$,
Output: The output shares of $\hat{z}$: $\langle \hat{z} \rangle$
The output shares of s_i: $\langle s_i \rangle$ for $i \in [N]$

1: **for** $i \in [N]$ // Players
2: $\langle a_i \rangle \leftarrow \text{SMul}(\, \langle \hat{x} \rangle, \langle \lambda_i^y \rangle \,)$
3: $\langle b_i \rangle \leftarrow \text{SMul}(\, \langle \hat{y} \rangle, \langle \lambda_i^x \rangle \,)$
4: $\langle s_i \rangle \leftarrow \langle \lambda_i^z \rangle - \langle \lambda_i^{xy} \rangle - \langle a_i \rangle - \langle b_i \rangle$
5: $\langle c \rangle \leftarrow \text{SMul}(\, \langle \hat{x} \rangle, \langle \hat{y} \rangle \,)$
6: $\langle s \rangle \leftarrow \sum_{i \in [N]} \langle s_i \rangle$
7: $\langle \hat{z} \rangle \leftarrow \langle c \rangle + \langle s \rangle$
8: **return** $\langle \hat{z} \rangle$ and $(\langle s_i \rangle)_{i \in [N]}$

In the security analysis we focus on a single MUL operation as extracted from Algorithms 14 and 15, denoted KKW_MUL, presented as Algorithm 16 and Algorithm 17, as the ADD

operation is linear and thus trivially NI. The lemmas are followed by the proofs.

Lemma 8.1. *Let G be the* KKW_MUL *gadget as described in Algorithm 16. Then, G is t-SNI for all $t < T$, if* SMul( ) *is t-SNI with uniform output-distribution.*

Proof. We thus need to show that for any set of $t < T$ intermediate variables and any subset $\mathcal{O} \subset [\hat{z}_1, \ldots, \hat{z}_T]$ of output shares such that $t + |\mathcal{O}| < T$, for each input variable v, there is an input set I_v with $|I_v| \leq t$ such that the t intermediate variables and the output variables $\langle \lambda_N^{xy} \rangle_{\mathcal{O}}$ can be perfectly simulated from these input sets.
Both the computation of $\langle \lambda^x \rangle$ and $\langle \lambda^y \rangle$ are straightforward and can be simply simulated, as they are linear operations. Whenever one of the terms involved in the computation of $\langle \lambda^{xy} \rangle \leftarrow$ SMul($\langle \lambda^x \rangle, \langle \lambda^y \rangle$) is probed, we add the corresponding values from the proof of the SNI-security (found e. g. in [BBD+16, Proposition 2]) of SMul( ) to the input sets I_v. The result $\langle \lambda^{xy} \rangle$ can be simulated without any input as SMul( ) is SNI. To simulate the output later on, we add all $\lambda_{i,j}^{xy}$ to the inputs I_v.
Finally, the computation of $\langle \lambda_N^{xy} \rangle$ is again linear.
For the output, suppose that $\lambda_{N,j}^{xy}$ was probed. There are two cases to consider: If λ_j^{xy} was probed, the inputs I_v already contains all $\lambda_{i,j}^{xy}$ and we can thus simulate $\lambda_{N,j}^{xy}$ perfectly. If λ_j^{xy} was not probed, λ_j^{xy} looks like a uniformly random element from $\mathbb{F}$ that is not used anywhere else, as it is produced by a t-SNI gadget with uniform output-distribution. We can thus uniformly sample a random element $r \in \mathbb{F}$ and replace $\lambda_{N,j}^{xy}$ by r. This implies strong non-interference. □

Lemma 8.2. *Let G be the* KKW_MUL *gadget as described in Algorithm 17. Then, G is t-SNI for all $t < T$, if* SMul( ) *is t-SNI with uniform output-distribution.*

Proof. We thus need to show that for any set of $t < T$ intermediate variables and any subset $\mathcal{O}$ of output shares such that $t + |\mathcal{O}| < T$, for each input variable v, there is an input set I_v with $|I_v| \leq t$ such that the t intermediate variables and the output variables indexed by $\mathcal{O}$ can be perfectly simulated from these input sets. To show this, we go through all variables of the algorithm and explain for all input variables v which indices are added to I_v.
Whenever one of the terms involved in the SMul( )-computation for a term $a_{i,j}$, $b_{i,j}$, or c_j is probed, we add the corresponding values from the proof of the strong non-interference of SMul( ) to the input sets I_v. Note that no inputs need to be added to I_v if $a_{i,j}$, $b_{i,j}$, or c_j were probed, as they are the result of a t-SNI gadget.
Whenever $s_{i,j}$ or a sub-term of $s_{i,j}$ is probed, we add the variables corresponding to $a_{i,j}$, $b_{i,j}$, $\lambda_{i,j}^z$, and $\lambda_{i,j}^{xy}$ to the input sets I_v. This clearly allows us to simulate all $s_{i^\star,j}$ and all sub-terms perfectly.

Whenever a sum $\sum_{i=1}^{i'} s_{i,j}$ (including s_j itself) is probed, we distinguish two cases. If $s_{1,j}, s_{2,j}, \ldots, s_{i',j}$ were all probed, we can simply simulate the complete sum. Otherwise, there is a term $s_{i^\star,j}$ with $i^\star \in \{1, \ldots, i'\}$ such that $s_{i^\star,j}$ was not probed. As $s_{i^\star,j}$ is the only place where $a_{i,j}$ is used, we make use of the fact that $a_{i,j}$ is constructed by an t-SNI gadget with uniform output-distribution. In other words, this means that $a_{i,j}$ looks like a uniformly random element from $\mathbb{F}$ that is not used anywhere else. We can thus uniformly sample a random element $r \in \mathbb{F}$ and replace the complete sum $\sum_{i=1}^{i'} s_{i,j}$ by r. Note that in the previous argument, we did not add anything to I_v.
Finally, whenever $\hat{z}_j$ is probed, we simply simulate s_j and c_j. As c_j is the result of a t-SNI gadget, we can simulate it without needing to add anything to the input sets I_v. As shown in the discussion about s_j, we can also simulate it without needing to add anything to the input sets I_v. This implies strong non-interference. □

To further support our security analysis, we utilized `maskVerif` [BBC+19] to confirm that both KKW_MUL offline and online gadgets are SNI-secure.
As we have shown that all components of Algorithm 14 and Algorithm 15 are SNI, the composability guaranteed by Lemma 2.1 as well as its generalization to multi-input/output SNI gadgets [CS20] implies the following theorem by adding suitable refresh gadgets (depending on the topology of circuit C that KKW is instantiated with). Note that SNI security of the refresh gadget (recalled in Algorithm 29) is already proved by Barthe et al. [BBD+16]

Theorem 8.3. *Suppose* H( ) *and* RefreshM( ) *are t-SNI secure gadgets for all $t < T$. The masked version of* KKW *presented in Fig. 8.1 is t-NIo for all $t < T$ and for public outputs* $\{\mathsf{com}_{i\neq i_k}^{(k)}\}_{k\in\mathcal{C}}$, $\{\mathsf{com}_i^{(k)}\}_{k\in[M]\setminus\mathcal{C},i\in[N]}$, $\{\mathsf{com\_off}^{(k)}\}_{k\in[M]}$ *and* $\{\mathsf{com\_on}^{(k)}\}_{k\in\mathcal{C}}$.

The public outputs stated above are not part of unprotected KKW proof elements. We thus have to validate the security of proof system in case these values are made public, which, however, is straightforward since they are indeed non-sensitive information that can be computed from response outputs (see the verification step of Fig. C.1). Furthermore, note that masking the hash function H( ) is important. There are scenarios where the inputs to H( ) are sensitive, but the outputs are not (such as Item 5 in Fig. 8.1). The computation of an unmasked hash functions might thus leak information about these sensitive inputs.

8.3 Masking Picnic

We start by analyzing the hashing operations in signature generation to determine which ones must be masked, then discuss the options for masking SHA3/SHAKE, and introduce

the half-masking technique, then estimate the overhead of masking the hash invocations. Our masked implementation of the Picnic3 signature generation function is a rather direct adaptation of the masked KKW proof protocol from Section 8.2. When compared to Section 8.2, the circuit is LowMc, and operations are done on N-bit words packed with a secret share from each of the N parties. Fig. C.2 (in Section C.2) gives an overview of the protections for each hashing operation in signature generation. A complete specification of our protected implementation, mirroring the official Picnic specification [Pic20], is given in Section C.2.

8.3.1 Implementation Security

We implemented several versions of masked Picnic3, of which we highlight two: (1) a provably NIo-secure implementation (as a direct consequence of Theorem 8.3) and (2) a performance-oriented implementation with partially unmasked non-sensitive intermediate values. For the former, we inserted the share refresh gadget (RefreshM) according to the generic composition rule stated in Lemma 2.1, and we verified with `maskVerif` that our complete specification of fully masked Picnic3 (see Section C.2.1) is indeed NIo-secure. On the other hand, we do not claim that our second implementation is NIo-secure, as there are some gaps between the analysis of Section 8.2 and this implementation, that we consciously allowed, in order to improve performance. We now explain these gaps and argue that they do not impact practical security, and in Section 8.4 we confirm the absence of leakage experimentally.

First, according to Algorithm 21, all intermediate seeds must be T-encoded until they are reconstructed at Line 29 and Line 32 and, as we argue in Section 8.3.2.1, reading t bits of a seed reduces security by at most t bits. As we assume t to be small, we accepted this risk to reduce the cost of masking SHAKE and memory required to store seeds.

By selectively masking the hash function calls (as described in Section 8.3.2) as opposed to masking all hash function calls, up to t bits of a single-use seed may leak to a side-channel attacker capable of accurately reading t bits from a single trace. (Since the seed is only ever used once, and the signature is randomized, subsequent traces have a fresh seed.) Against such an attack, the security of our L1 implementation decreases from 128 to 112 bits. As shown in Sections 8.3.4 and 8.4 this optimization gives significant performance gains, so we see this as a reasonable trade-off. Recent work by Kannwischer et al. [KPP20] describes *single-trace attacks* on the unprotected XKCP Keccak implementation. These attacks use a single trace recorded during the computation of $y = \text{SHAKE}(sk||x)$ and aim to recover all of a secret key sk, or part of y. While single-trace attacks could threaten some of the unprotected hash calls in our optimized implementation (e.g., when deriving the per-party or per-MPC instance

seeds), the results of [KPP20] do not extend to the M4, and the length constraints on sk, x, and y in our application. Future work may improve single-trace attacks, and in that case the conclusion of [KPP20] is that lightweight countermeasures will provide effective mitigation.

Half-masking (discussed in Section 8.3.3) introduces that KangarooTwelve [VWA+21] is a secure hash function. This assumption is only for security against the type of t-probing side-channel attack we consider; and half-masking can be used by individual implementations without changes to the Picnic specification. We provide benchmarks in Section 8.4 showing the performance advantage of half-masking: based on this and the fact that KangarooTwelve appears to be a relatively mild assumption, we enable half-masking by default in our implementation.

Finally, our provable security analysis assumed an SNI-secure hash implementation. Although one could use the fully SNI-secure masked Keccak as suggested by Barthe et al. [BBD+16], other previous works [BDPA10, Dae17, GSM17] achieved more efficient implementations with smaller amounts of random bits, albeit without a provable security guarantee. We implement three instances of masked Keccak (named `IND`, `DOM`, and `SNI`) with different security levels, which we explain in detail in Section 8.3.3. In Section 8.4, we compare the concrete performance and perform practical leakage analysis. From these experiments, we conclude that our implementation of `IND` – the fastest instance among three – does not leak information, which provides some assurance.

8.3.2 Side-Channel Protections for Hashing in Picnic

In this section and Algorithm 18 we give a more detailed description of the parts relevant to this paper.

Parameters. M is the number of MPC instances, N is the number of parties, τ is the number of revealed online executions and κ is the security parameter (e.g., $\kappa = 128$ for security level L1). The circuit C, defined over the binary field $\mathbb{F} = \{0, 1\}$, is also part of public parameters. Concretely, the circuit is $\mathsf{Enc}(\mathbf{w}, p) \stackrel{?}{=} c$, where Enc is the LowMc block cipher with κ-bit key and block size, $\mathbf{w}$ is an κ-bit input witness (a LowMc secret key), p and c are the plaintext and ciphertext, both κ bits long. If the input to C is a block cipher key that maps p to c, the circuit outputs 1.

Key Generation. In the presentation below, the key pair is $(pk, sk) = ((c, p), \mathbf{w})$, where both p and $\mathbf{w}$ are random κ-bit strings, and then c is computed as $c = \mathsf{Enc}(\mathbf{w}, p)$.

8.3.2.1 Hashing Operations for Signing

The concrete sign operations are described in Algorithm 18, following the Picnic specification [Pic20, Section 7.1]. When compared to the stylized description of KKW in Fig. C.1, here we include more details and list all hashing operations, since we will analyze them with respect to the probing attacks below. Some of the functions related to expanding seeds using a tree construction or creating a Merkle tree of commitments (gen_seed, get_leaves, build_tree, and open_tree) are left to the specification for simplicity.

The hash function calls are denoted by H and we omit the byte used for domain separation present in the specification. The KDF expands an arbitrary length input to an arbitrary length output. Both H and KDF are instantiated with the SHAKE XOF.

Algorithm 18 Description of Picnic signing highlighting hashing operations.

Input: signer's key pair $sk = \mathbf{w} = (w)_{w \in \mathsf{IN}}, pk$, message to be signed Msg.

1: // Derive root seed:
Sample random $R \in \{0,1\}^{2\kappa}$, $(\mathsf{seed}^*, salt) \leftarrow \mathrm{KDF}(sk||Msg||pk||\kappa||R)$
2: $\mathsf{iSeed\_tree} \leftarrow \mathsf{gen\_seed}(\mathsf{seed}^*, salt, M, 0)$ // Tree of initial seeds
3: // Initial seed for each MPC instance:
$(\mathsf{iSeed}^{(1)}, \ldots, \mathsf{iSeed}^{(M)}) \leftarrow \mathsf{get\_leaves}(\mathsf{iSeed\_tree})$
4: **for** each $k \in [M]$
5: $\quad \mathsf{seed\_tree}^{(k)} \leftarrow \mathsf{gen\_seed}(\mathsf{iSeed}^{(k)}, salt, N, j)$ // Seeds for MPC instance k
6: $\quad (\mathsf{seed}_1, \ldots, \mathsf{seed}_N) \leftarrow \mathsf{get\_leaves}(\mathsf{seed\_tree}^{(k)})$ // N per-party seeds
7: $\quad$ For each $i \in [N]$: $\mathsf{tapes}_i^{(k)} \leftarrow \mathrm{KDF}(\mathsf{seed}_i, salt, k, i)$
8: $\quad (\mathsf{aux}^{(k)}, \mathsf{tapes}_N^{(k)}) \leftarrow \mathsf{offline}(\mathsf{tapes}_1^{(k)}, \ldots, \mathsf{tapes}_N^{(k)})$
9: $\quad$ For each $i \in [N-1]$: $\mathsf{com}_i \leftarrow \mathsf{H}(\mathsf{seed}_i, salt, k, i)$
10: $\quad \mathsf{com}_N \leftarrow \mathsf{H}(\mathsf{seed}_N, \mathsf{aux}^{(k)}, salt, k, N)$
11: $\quad \mathsf{com\_off}^{(k)} \leftarrow \mathsf{H}(\mathsf{com}_1^{(k)}, \ldots, \mathsf{com}_N^{(k)})$
12: $\quad$ For each input wire w: $\hat{w}^{(k)} \leftarrow w \oplus \sum_{i \in [N]} \lambda_i^w$
13: $\quad (\mathsf{msgs}_1^{(k)}, \ldots, \mathsf{msgs}_N^{(k)}) \leftarrow \mathsf{online}((\hat{w}^{(k)})_{w \in \mathsf{IN}}, \mathsf{tapes}_1^{(k)}, \ldots, \mathsf{tapes}_N^{(k)}, pk)$
14: $\quad \mathsf{com\_on}^{(k)} \leftarrow \mathsf{H}((\hat{w}^{(k)})_{w \in \mathsf{IN}}, (\mathsf{msgs}_1^{(k)}, \ldots, \mathsf{msgs}_N^{(k)}))$
15: $\mathsf{com\_on\_tree} \leftarrow \mathsf{build\_tree}(\mathsf{com\_on}^{(1)}, \ldots, \mathsf{com\_on}^{(M)})$
16: $h \leftarrow \mathsf{H}(\mathsf{com\_off}^{(1)}, \ldots, \mathsf{com\_off}^{(M)}, \mathsf{com\_on\_tree.root}, salt, pk, Msg)$
17: Parse h as $(\mathcal{C}, \mathcal{P})$ where $\mathcal{C} \subset [M]$ and $\mathcal{P} = \{i_k\}_{k \in \mathcal{C}}, i_k \in [N]$
18: $\mathsf{com\_on\_info} \leftarrow \mathsf{open\_tree}(\mathsf{com\_on\_tree}, M, \mathcal{C})$
19: $\mathsf{iSeed\_info} \leftarrow \mathsf{reveal\_seed}(\mathsf{iSeed\_tree}, M, \mathcal{C})$
20: For each $k \in \mathcal{C}$: $\mathsf{seed\_info}^{(k)} \leftarrow \mathsf{reveal\_seed}(\mathsf{seed\_tree}^{(k)}, N, i_k)$
21: $Z \leftarrow (\mathsf{com\_on\_info}, \mathsf{iSeed\_info}, (\mathsf{seed\_info}^{(k)}, \mathsf{aux}^{(k)}, (\hat{w}^{(k)})_{w \in \mathsf{IN}}, \mathsf{com}_{i_k}^{(k)}, \mathsf{msgs}_{i_k}^{(k)})_{k \in \mathcal{C}})$.

Output: $(h, salt, Z)$ as a signature

We now consider which hashing operations must be protected against side-channel attacks, and to what degree. The Picnic specification supports randomized signatures (and recommends this option, following [AOTZ20]) by appending a random value to the KDF input when deriving the root seed. We assume this option is used throughout, as otherwise the cost of side-channel protections would be significantly higher, since all hash function calls would require masking (as opposed to only 35% shown below), and all random seeds would need to be T-encoded. First, we note that all inputs to the challenge computation are public, so this hash does not need to be masked. We now analyze the other hash function calls in order. Fig. C.2 (in Section C.2) gives an overview of the operations. When referring to individual hashing operations, we refer to the steps in Algorithm 21.

Deriving the root seed. For Line 1, the SHAKE XOF is used as a KDF to derive a root seed for signature generation. The input sk must be protected from side-channel attacks. As a first option, one could choose the root seed at random, and avoid the KDF altogether. However, deriving the root seed from the secret key and random data hedges against failures in the RNG, see the analysis for Picnic in [AOTZ20]. If sk is stored T-encoded, then we can hash all of the shares in place of sk, and append a random value. Our implementation masks this hash function call since it is relatively cheap in the context of a signature, and it makes testing easier because our implementation can produce signatures that match known test vectors.

Deriving other seeds. When generating the seeds in Line 2, Line 3, Line 5, and Line 6, protecting against the limited type of leakage we consider in this work is not necessary, since seeds are unique per-signature and are always hashed before use. Suppose an attacker A can read t bits of a leaf or intermediate seed s. With overwhelming probability each seed is only ever used in one signature, so traces from multiple signing operations will not give more information about s.

There are three possible uses of s to consider. When s is a seed from a leaf of the tree, case 1 is that s is hidden and the attacker has a commitment to it (computed in Line 9 and Line 10), and case 2 is when s is used to seed KDF (in Line 7), and A has some of the output bits. In case 3, s is a hidden intermediate seed, the attacker has one of the two child seeds, derived by hashing s.

We can model all three cases as the attacker having $C = \mathsf{H}(s)$ along with t bits of s, where H is a secure hash function. In practice H is the SHAKE XOF, which the existing analysis of Picnic already assumes is a random oracle. Then if A makes q queries to

H, they recover the missing $\kappa - t$ bits of s with probability not more than $q/2^{\kappa-t}$. This considers only a single seed and digest, which we can do since each input to H is unique, by construction (the Picnic spec uses a domain separation value, random salt, and counters to prevent multi-target attacks [DN19]). In practice $\kappa \geq 128$ and t will be 16 or less (see Section 2.1), therefore the security of our implementation is still at least 112 bits.

Computing random tapes. In Line 7 we expand the per-party seeds to random tapes. The inputs do not need to be protected (as discussed in the previous paragraph), but all output bits must be protected, since some of the random tape bits will correspond to shares of the unopened party and must be kept secret as shown in Section 6.3. We mask these calls, so the output is T-encoded (which increases the amount of memory required to store the tapes by a factor of T).

Computing commitments. In Line 9, a commitment of the form $H(\mathsf{seed} \| salt)$ is computed. Here the private input is a seed, which is not sensitive to leakage of up to t bits, as discussed above, and the output is public. Therefore, Line 9 does not require masking.
In Line 10, the last party's commitment has the additional input aux, which is sensitive to leaking of individual bits. We must mask this call, but since the output is public, we can use the half-masking technique of Section 8.3.3.
In Line 11, we hash only public values, and no masking is required. In Line 14, all inputs are sensitive to leaking individual bits (e.g., $\hat{w}$ is sensitive due to the attack described in Section 6.3.1). Because the output is public, the half-masking technique is applicable.

8.3.3 Masking SHAKE

We implement multiple methods to protect the SHA3 family of function against DPA attacks. In all of them, the Keccak-f state array A is secret shared into two arrays a, b, such that $A = a+b$. In the basic method proposed in [BDPA10], the linear operations are performed on the individual state arrays, then for the non-linear step (denoted χ) $A_i \leftarrow A_i + (A_{i+1} + 1)A_{i+2}$ the shares (a, b) are updated as $a_i \leftarrow a_i + (a_{i+1} + 1)a_{i+2} + a_{i+1}b_{i+2}$ and $b_i \leftarrow b_i + (b_{i+1} + 1)b_{i+2} + b_{i+1}a_{i+2}$, evaluated left-to-right. The cost of the linear operations are doubled, addition of constants have the same cost, and the cost of χ is doubled, plus two additional AND and XOR operations, so the computational cost of the masked round function is roughly doubled. One must also consider the cost of generating random values to create the secret shares. This method (herein called `IND`) only achieves

independence from the native variables, and the same approach can be generalized to three or more shares. In Domain-Oriented Masking (DOM) [GSM17], the AND operations between shares a and b are further protected to satisfy SNI-security with a random mask Z as $(a_{i+1}b_{i+2} + Z)$ and $(b_{i+1}a_{i+2} + Z)$, respectively. However, this is still not sufficient for the masked Keccak as a whole to be SNI secure: due to the θ-layer, which applies a linear transform to the state array A, both inputs to the AND gadget in χ depend on the same previous state bit. This is a typical pattern of insecure composition observed in [BBD+15a, §2.3]. Therefore, the third method (denoted by SNI) achieves SNI-security by additionally refreshing shares of the state array A for every invocation of χ, as already suggested by Barthe et al. [BBD+16, §8.2].

Half-Masked SHAKE. When expanding a seed to a random tape, we have shown that security is maintained when leaking a small part of the seed (t bits of fewer), so the input is not sensitive to this bounded leakage, but the output is sensitive. Conversely, when creating a commitment, the individual input bits may be sensitive but the output is public. An established assumption (for SHAKE128) is that security is preserved using only half the number of rounds, and there is a proposal called KangarooTwelve (K12) [VWA+21], that uses 12 instead of 24 rounds. Therefore, for short inputs and outputs, one can view SHAKE as two calls to K12, and mask only one of the calls. In the case of sensitive inputs, we mask the first 12 rounds: an attacker who learns the state at the 13th round is effectively given a K12 digest of the input, which sufficiently hides the input under the assumption that K12 is a secure hash function. Similarly, when only the output bits are sensitive, we mask the last 12 rounds, any state bits observed by the adversary in round 11 does not leak useful information about the output assuming that K12 is secure.

8.3.4 Estimated Overhead of Hash Function Masking in Picnic

Here we provide a rough estimate of the overhead introduced by masking the SHAKE calls in Picnic3, which will have a high impact on the cost of signing since hashing is a large portion of the signing time (e.g., at L1 it is about 57% of the signing time on x64 [KZ20], and for our ARM M4 implementation it is about 71%).

- For seed tree hashing, we have about $M + \log_2 M$ hashes to compute the round seeds, and $MN + M \log_2 N$ hashes for the per-party seeds. None of these must be masked.
- For random tape expansion, we have MN hashes, all of which must be masked.

- For commitments, we have $NM+2M+\log_2 M$ hashes and must mask $2M$ of these.

The total number of hashes is thus $3MN + 3M + 2\log_2 M + \log_2 N$ and $MN + 2M$ of these must be masked. At L1, all hash operations involve one call to Keccak-f, so all calls have approximately the same cost. Again at L1, $M = 250$, $N = 16$ so we find that about 35% of hashing must be masked. Since all masked hash operations have either non-sensitive input or output they need only be half-masked (as explained below).
Now suppose we focus on first order protection (the case $T = 2$), and assume that masked SHA-3 is about 2.73 times slower than unmasked SHA-3, and that a half-masked SHA-3 is about 1.95 times slower (these are the ratios from our implementation described in Section 8.4). Then we expect a 1.61x increase in time spent hashing in masked Picnic3, and 1.35x increase when half-masking is used.

8.4 Implementation and Experimental Evaluation

In this section we benchmark our implementation and discuss performance, then describe our experiments to ensure that our implementation is side-channel resistant in practice.

8.4.1 Implementation and Benchmarks

We implemented our masked version of Picnic signing and benchmarked it on the ARM Cortex M4, using the `pqm4` [KRSS] suite and the STMicro developer board STM32F407G-DISC1. This board has one MB of flash memory and 192KB of RAM and comes equipped with a true random number generator implemented as a hardware peripheral. The microcontroller clock frequency ranges from 24 to 168MHz, so following standard practice our benchmarks were executed at the lowest frequency to avoid the impact of memory wait states [FA17, HL19].
Our implementation is derived from the Picnic optimized implementation, which is primarily optimized for x64 platforms, and is not well-optimized for the M4. As such, our implementation results aim to bound the overhead of masking countermeasures. We also focus only on first order protection, i.e., the case $T = 2$, and implement only the L1 parameter set `picnic3-L1`. As most of our countermeasures are general, we expect them to apply equally to more optimized M4 implementations of Picnic, and (with some effort) to implementations of other MPCitH-based proof systems.
Since our masked implementation produces `picnic3-L1` signatures compatible with the specified version [Pic20], we do not repeat signature or key sizes in our benchmarks: public keys are 34 bytes, secret keys are 17 bytes, signatures are 12.4KB. All other parameters such as number of MPC parties, MPC instances, digest lengths, etc. are as

Picnic Masking	SHAKE Masking	Signing cycles	Hashing	Masking Overhead	Stack	Code	Random bytes (KB)
No	None	304	71%	1.00	32,460	121,349	0
Yes	None	460	50%	1.51	32,500	131,326	2,025
Yes	All-SNI	1663	86%	5.47	32,724	166,216	158,172
Yes	All-DOM	1289	81%	4.24	32,724	158,776	80,378
Yes	All-IND	856	72%	2.82	32,724	148.712	2,585
Yes	Selective	613	62%	2.01	32,460	148,712	2,025
Yes	Sel. Half	546	57%	1.80	32,460	148,712	2,025

Table 8.1: Benchmarks in millions of Cortex-M4 cycles showing the masking overhead for types of side-channel protections. The options for Picnic are "No" masking and $T = 2$ masking, as described in Section 8.3. For SHAKE, the "None" option indicates no masking is used for hash computations, "All-" prefix means every call is masked in one of three possible ways: SNI-secure [BBD+16], Domain-Oriented Masking (DOM) [GSM17] or using independent values (IND) [BDPA10]. "Selective" means that only sensitive calls are masked with independent values (as described in Section 8.3.2.1), and "Selective Half" means that in addition to Selectively masking and IND, we use half-masked SHAKE. The Hashing column provides the fraction of the signing time spent computing SHAKE.

specified in [Pic20]. For reference, the verification time in our implementation is 204M cycles.

In order to experimentally verify the absence of leakage, we make use of the FvR tests.

Masked Keccak. We implemented three different flavors of masked Keccak described in the last section: IND, DOM, and SNI. The implementation was built on top of the in-place 32-bit ARMv7-M assembly code found in the official Keccak code package [3] (XKCP) to operate over a double-sized state storing the two shares. We implement the same Keccak API used in Picnic by replicating functions over each share of the state, and modifying the round function to implement the non-linear operations. Because of the larger state, additional pressure was put on the registers and several intermediate variables had to be spilled onto the stack. This caused some additional performance overhead beyond the raw cost of masking. In order to prevent leakage, we took additional care to rotate registers between rounds to prevent them from loading shares of the same variable [BDPA10].

[3] https://github.com/gvanas/KeccakCodePackage

Benchmarks. In Table 8.1 we give cycle counts for our masked implementation with various options for how the hash function calls are masked. The masking cost for the non-hashing operations is 156M cycles, which represents an overhead of 1.5x over baseline. Since this is effectively doing the MPC simulation with 2-encoded values, we might expect a factor two slowdown rather than 1.5, however, this is explained by the fact that many of the operations to implement LowMc are ANDs and XORs with public constants, which are more efficient than operations on 2-encoded values. Then the cost when masking the hashing naively (by masking all operations), is given for the three Keccak masking options (`SNI`, `DOM` and `IND`) and we see that the overhead is 1203M, 829M and 396M cycles respectively. By using our analysis of Section 8.3.2.1, and selectively masking only sensitive hash function calls, the overhead for `IND` drops to 153M cycles, and all the way down to 86M cycles when we additionally use the half-masking optimization. In this most performant case, we have roughly 2/3 of the overhead accruing to the Picnic and MPC operations, and 1/3 to the hashing.

Stack usage was essentially constant for all configurations we benchmarked, since the total amount of memory required is dominated by storing the signature, the commitment and seed trees and not by the storage space for intermediate values that we must t-encode. Code size increases by 1.2x in the most performant masked implementation (selective half-masked), and by 1.4x in the fully SNI-secure version. Finally, the randomness requirements range from the baseline of $\approx$ 2MB when Keccak is masked with the `IND` method, to the much higher 80MB for the `DOM` method and 158MB for the `SNI` method, as these methods require additional refreshing of nonlinear operations within Keccak. Using the selective half-masking option reduces the randomness requirements of the `DOM` and `SNI` options significantly, since the number of hash function calls is decreased and some calls are only half-masked. By our estimate in Section 8.3.4 this would reduce randomness usage by about 65% for the `DOM` and `SNI` options. In terms of the $\approx$ 2MB of randomness used for the non-hashing operations, these are partly due to the refresh operations within the LowMc implementation (Line 11), they are required for SNI security and since they did not have a significant impact on run time, we did not investigate the option of removing them. The other significant randomness consumer in the masked LowMc implementation is the masked AND operation (Algorithm 28). Here, future work could experiment with an implementation that masks ANDs as in the `IND` method for Keccak, with the aim of reducing randomness and improving run time.

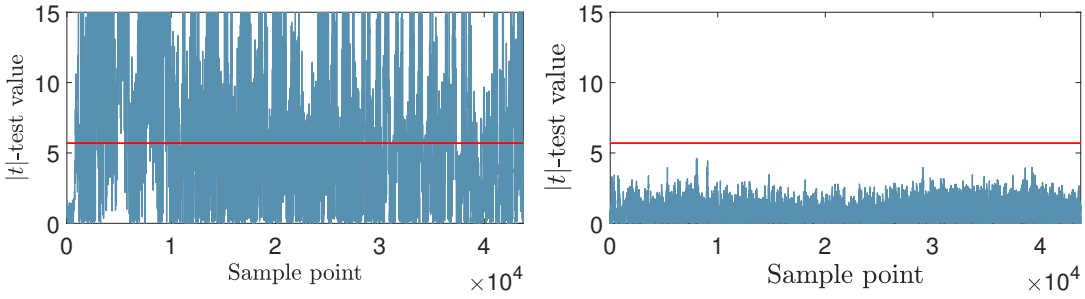


Figure 8.2: A first-order leakage detection test based on 2,000 traces on the protected Keccak implementation with fixed masking (left) and on 1,000,000 traces on the protected Keccak implementation (right). The threshold of $|t| \geq 5.7$ (as suggested by [DZD+18]) is violated throughout the implementation with fixed masking, indicating strong leakage. For the correctly masked implementation, the $|t|$-value remains below 5.7, even with 1,000,000 traces, indicating the absence of exploitable first-order leakages.

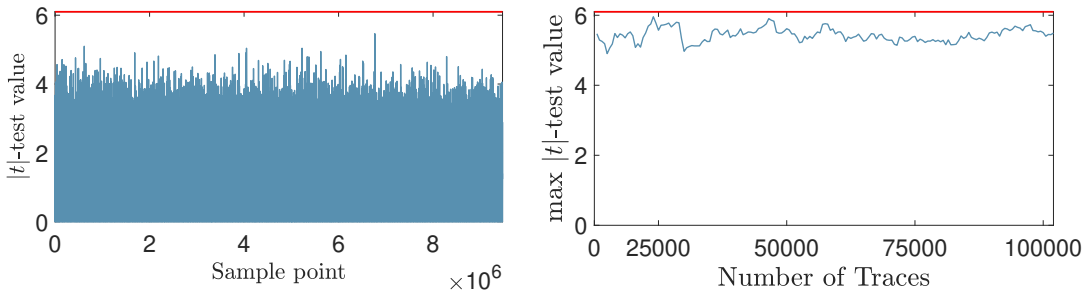


Figure 8.3: A first-order leakage detection test based on 100,000 traces on the Picnic3. The $|t|$-value remains below 6.1 (as suggested by [DZD+18]), indicating the absence of exploitable first-order leakages for the 100,000 traces. Also, the maximum $|t|$-value is bounded and becomes stable with the increased number of traces (right).

8.4.2 Experimental Leakage Analysis

To ensure our masked implementations of Keccak and Picnic are practically side-channel resistant, we performed measurements of the implementation to confirm the absence of leakage. The details of the experimental setup can be found on Chapter B.

Masked Keccak Leakage Evaluation. For Keccak we evaluate the `IND` method and follow the FvR approach to detect all possible first-order leakages. During the trace collection phase, a set of side-channel traces is collected by processing either a fixed input or a random input under the same conditions. The fixed or random choice for the input is made at random.After that, we calculate the means and standard deviations of the

two side-channel trace sets separately. The t-test indicates whether the two distributions have the same mean, i.e., they are *indistinguishable* for a first-order SCA. We apply the customary threshold values for long traces 5.7 as suggested by [DZD+18].
To show the sensitivity of the measurement setup for first-order leakages, we apply the test once to the correctly masked implementation and once to the same implementation with fixed masks. When masks are not chosen at random, the test must detect the resulting first-order leakage. The left hand side of the Fig. 8.2 shows the evaluation results of the masked Keccak implementation with fixed masks based on 2,000 measurement traces. As expected, the leakage test indicates strong leakage, with $|t|$ clearly above 5.7. When masks are chosen uniformly at random, the t-value remains below 5.7 as shown in the right hand side of the Fig. 8.2, even if the number of measurement traces is increased to 1,000,000. We thus conclude that the masked Keccak implementation is secure and provides the expected resistance to first-order attacks.

Masked Picnic Leakage Evaluation. In order to analyze the leakage for the whole first order masked Picnic3 implementation, we follow a similar methodology as for Keccak and employ the FvR approach. We collect the traces starting at the beginning of signature generation until the end of the first MPC instance, i.e., including a single preprocessing phase and simulation of an online phase(Line 1 to Line 22 in Algorithm 21). Note that after this point, everything is public, and any leakage gives no additional information beyond what is made public in the signature. To analyze our signature implementation, we choose the FvR *key* scenario, under randomized messages, as proposed in [TG16] for asymmetric cryptosystems. [4] In addition we needed to add artificial wait cycles before accessing the board's hardware TRNG to ensure that we will never need to varying amounts of time for it to become ready,as this would destroy the constant time property required for the TVLA. Note that this change is only necessary for the test setup and not required in the production code.
The sets of side-channel traces are collected by signing a random message using either a fixed key or a random key. As shown in Fig. 8.3, the t-value remains below 6.1 using 100,000 traces which indicates the absence of leakage. Moreover maximum $|t|$-value is

[4] The FvR-message scenario with a fixed key is not a good fit for randomized signature schemes like Picnic3 where the entire internal state is randomized even for fixed messages. If the randomness is fixed (by fixing R in Line 1 of Algorithm 21), the signature scheme becomes artificially deterministic. Then several *public* parts of the signature generation process, including the signature itself, will be picked up by the TVLA test. Similarly, in the deterministic case the root seed is fixed for fixed messages (and random for random ones), also creating a leakage which does not exist for randomized signatures. We still performed the analysis and verified that leakage only occurs in such expected places.

indeed bounded and have a stable pattern. Remark that an exploitable leakage as shown in Fig. 6.3 or Fig. 6.4 exceeds the threshold value within as small as 2,725 traces and has a clear increasing pattern. Thus we can conclude that the first-order masked Picnic3 implementation provides the expected SCA resistance.

Scaling to higher security levels and masking orders. Our implementation and experimental evaluation is limited to security level L1 and masking order $T = 2$. Since we expect the proportion of time spent on hashing vs. MPC simulation to be similar at levels L1 and L5 (as was the case for x64, see [KZ20, Table 7]), we expect the overhead of our masking techniques to be similar at L5 as well. When T increases, we can only make rough predictions. We expect running time overhead to increase quadratically and memory overhead to increase linearly, due to the asymptotic behavior of masking nonlinear operations, and the additional storage required for T-encoded values.
The Picnic specification [Pic20] and NIST submission includes parameter sets using both the ZKB++ proof system and the KKW system, as well as specific choices of parameters for LowMc. Since the KKW-based parameters (referred to as Picnic3) are the most efficient in terms of signature size, we choose to focus on those in this paper. In particular our masked implementation is limited to the parameter set Picnic 3-L1. Fig. C.1 describes the KKW proof system at a high level, and Algorithm 31 describes Picnic3 signing in full detail.

Comparison with other NIST PQC candidates. To the best of our knowledge, most other PQ signatures currently do *not* have publicly available masked implementations, except for lattice-based Fiat–Shamir *with aborts* [Lyu09] signatures: GLP [BBE+18], BLISS [BBE+19], Dilithium [MGTF19] (NIST PQC finalist) and qTESLA [GR19] (Round 2 candidate). While these masked signing operations do output a signature compliant with the existing verification algorithm, they rely on an additional non-standard hardness assumption for provable side-channel-security (see [BBE+18,DOTT21] for details). The issue could be circumvented by modifying the "commit" message of the underlying Σ-protocol, but this in turn breaks the interoperability of the output signature. By contrast, signatures directly derived from our generic approach to masking KKW (Section 8.2) as well as NIo-secure Picnic3 implementation maintain interoperability, and may optionally make additional assumptions for improved performance.
Performance-wise, the benchmarks on Cortex M4 given by Gérard and Rossi [GR19, Table 6] show much less overhead than ours: their first-order protected qTESLA-I incurs only 2.1x overhead in signing clock cycles and requires 343 KB of fresh randomness,

while provably secure masked Picnic3 is 5.5x slower than unprotected and consumed 158 MB of randomness. However, by trading provable security guarantee as we describe in Section 8.3, our empirically validated countermeasures achieve a lower overhead overall of 1.8x, requiring 2 MB of randomness. Giving a meaningful performance comparison with masked Dilithium [MGTF19] is hard, as they only provide benchmarks on Intel for the whole signing operation. Although their overhead for first-order protection is about 5.6x, we expect that it can be made faster on the M4 by using the platform's TRNG.

8.5 Conclusion and Future Work

In this chaper we study the side-channel security of MPCitH proof protocols and related signature schemes. We then show that masking the signing operations is a practical countermeasure for side-channel attacks, and prove our masked KKW and Picnic3 meet the standard security notion (NIo), with a mix of both manual proofs and formal verification with the `maskVerif` tool.

We implemented a masked version of the Picnic3 signature scheme for the ARM Cortex M4 as a case study, and found that the cost of masking (in terms of runtime) is high when we simply apply SNI-secure masking to all hashing operations. After careful analysis of the hashing operations, we found that the masking overhead can be quite reasonable (as low as 1.8x) under modest assumptions that we verified with practical leakage analysis of our implementation. With hardware support for side-channel protected hashing, our work shows that the overhead of masking the non-hashing parts of Picnic signing is about 1.5x, and our SNI analysis applies here.

Our flexible masked SHA-3 implementation is the first publicly available one, and will be useful to other projects as SHA-3 becomes more common. We also expect our half-masking optimization to find application in other implementations, as most hash operations have a non-sensitive input or output.

Performance improvements (while maintaining resistance to side-channel attacks) are an obvious direction for future work, both on the M4 and other embedded platforms. Reducing the amount of randomness consumed by our mitigations is also an interesting way to improve performance, together with generalizing to higher order protection efficiently.

Finally, an implementation that combines SCA resistance and resistance to fault attacks (perhaps leveraging the fault-resistance results for Picnic in [AOTZ20]) would also make a good follow-up work.

9

Conclusions

In this chapter, we give a summary of the thesis and underline the main contributions. We look back to the main questions posed in Chapter 1 and conclude this work.

9.1 Summary

This work was motivated by the need for strong, reliable and flexible countermeasures against advanced physical attacks. Although these attacks dominate the literature in the last two decades, the importance of increased countermeasures has continued to grow with the increasing number of smart devices. Moreover the increase in computation power and advances in attacks require us to employ not ad hoc methods, but *provable* secure countermeasures.
The thesis started with the adversarial models and their effect on the real world cryptographic devices and gave our main research questions. In a nutshell, the thesis is divided into two major parts. In Part I, we took an existing countermeasure and enhanced it with novel techniques to resist more advanced or combined attacks while preserving the provable security property. Then, in Part II we focused on a post-quantum signature scheme family, Picnic, and provided countermeasures for each variant in the family.

9.1.1 A Summary of Part I

Part I starts with a detailed introduction to side-channel, fault injection and combined attacks. Each line of attacks is followed by countermeasures and ways of proving or verifying the security against these attacks.
In Chapter 3, we focused on combined attacks in the in gray-box model and answered our first question:

> *Can we provide a combined countermeasure built on secure multi-party computation that can be efficiently implemented while preserving the information-theoretic security?*

In this chapter, we focused on well-known techniques which is implemented to protect against side-channel analysis: Secure Multi-party Computation and polynomial masking. Briefly speaking in the scheme we share a secret using a degree d polynomial.

Our first goal was to observe the faulty behavior of the masking scheme. We investigated the first challenge; what makes a fault detectable? The answer was hidden in the essence of the scheme: the degree of the secret sharing polynomial. We observed that a fault injection disturbs the degree of the polynomial such that the degree becomes larger than d and we define the invalid secret sharing as a secret sharing whose polynomial's degree is greater than d. Although this effect was preserved with most of the operations, secure multiplication by [RP12] could possibly eliminate this effect and result in output shares corresponding to a valid secret sharing no matter the fault. Therefore we built our scheme on this observation and used the degree as a fault detection mechanism. To waive the fault-elimination property of multiplication, we proposed a new multiplication engine that propagates faults with high probability. The fault propagation was achieved by fault preserving SMC operations.

The proposed scheme, (n, d)-SMC, works on $n = 2d + \varepsilon + 1$ shares instead of $n = 2d + 1$ shares. Therefore the redundancy was not added as form of additional gadgets but as a form of increasing the number of shares but keeping the degree as the same. Moreover the fault preserving property was added to the scheme with an extra operation within the multiplication operation which makes this approach quite efficient.

Thus the foundation of the scheme can be summarized as, once a fault is injected and becomes detectable, the fault propagates through the masked operations and stays detectable. After becoming detectable, the degree of the faulty polynomial can decrease by operations which result in a small probability of undetectable faults. However, a user can perform fault detection regularly which increases the detection rates. Accordingly, we proposed a secret-preserving fault detection mechanism. Moreover a novel recombination operation is used for both fault detection and recombination. More importantly, this gadget ensures infective computation i.e., randomizes the output if the fault is still detectable. Thus an adversary cannot derive an attack from these randomized values.

The natural next step was to analyze the security of the scheme against SCA and FA. In order to do so, we used the ISW probing model and non-interference notion. We first adjusted the t-SNI notion to work with polynomial masking. Moreover, the fault detection properties of our scheme are analyzed by the propagation notion introduced by [SMG16]. We formally gave the probability of undetectable faults for each gadgets in our scheme.

The fault detection capabilities of our scheme was given with a simulation written in

SAGE. We supported our results with a practical AES-128 implementation on ARM Cortex M0+ which was provided by a co-author. The performance analysis and an extensive leakage analysis including the higher order moments were produced by the implementation which is publicly available:

https://github.com/vernamlab/Robust-AES.

In Chapter 4, we delved into more complex adversarial model and focused on white-box adversarial model. As the attacks on these model became lethal for WBC we concentrated on the question:

> *Can we introduce a combined countermeasure based on basic masking in such a way that it achieves information-theoretic security against Differential Computation Analysis and Algebraic Differential Computation Analysis?*

In this case, the foundation of our scheme relied on another well-known side-channel countermeasure: Boolean masking. As our first step, we asked ourselves a similar question as above and tried to find the cause of DCA and ADCA individually. We observed that (as given in [BU18]) while we can counter DCA by increasing the masking order, ADCA required increase in degree of the decoding function. Therefore we achieved such a secure construction by combining linear shares with a non-linear component.
As our security notion, we used the two main notions in the literature: probing security [ISW03] and prediction security [BU18]. First we showed that these notions are incomparable i.e., we needed to prove the security using both notions. We employed the t-SNI notion to prove the security against DCA like attacks. As the caveat of our scheme, only a subset of shares is uniformly distributed, as the non-linear component is biased and an adversary can predict non-linear shares with high probability. Therefore we are losing one probing security order to obtain algebraic security.
The algebraic security of our scheme was proven using prediction security notion and we proved the security of concrete constructions for $(n, 1)$ and $(n, 2)$ masking scheme. Although the $(n, 1)$ scheme can be seen as a natural extension of the scheme by [BU18], the $(n, 2)$ masking scheme and its security definitions are introduced by extending the previous notion in a non-trivial way. Moreover we provided a novel composition proof using two fold parallel composability and sequential composability. Therefore we proved the composition of our construction for arbitrary circuits.
We supported our results with the verification tools by [BU18] (for prediction security) and `MaskVerif` for probing security. We provided a proof-of-concept AES implementation to analyze the overhead and leakage assessment. The implementation with its analysis and verification codes is publicly available:

https://github.com/UzL-ITS/white-box-masking.git.

9.1.2 Summary of Part II

In Part II, we concentrated on post-quantum cryptosystems. In Chapter 5, we gave a gentle introduction to PQC and the NIST Post-quantum project and in Chapter 6 we introduced our main motivation: the need for sound and secure designs for PQC and as target we chose the Picnic signature family as our target.
Secure Multi-party Computation is one of the most widely studied cryptographic primitives. For a long time, SMC protocols were believed to be of purely theoretic interest, but recent developments have shown that they are usable for a wide range of practically relevant applications. One well-studied version of these protocols, called MPC-in-the-Head (MPCitH), is on the brink of seeing widespread use, e. g. as part of NIST's search for a Post-Quantum Signature Suite of algorithms in the form of Picnic [CDG+17]. Due to the development of practical MPC protocols, efficient implementations of such protocols were also developed. As shown in Chapter 6 lack of protection can be abused by side-channel adversaries.Thus in Chapter 7 we aim at the question:

> *Can we improve* ZKBoo *resp. the MPC-in-the-Head paradigm in general in such a way that the unopened view is provably secure even if the opened views are reachable by an adversary?*

We showed, using the example of ZKBoo, that MPC-in-the-head protocols can be adapted to fulfil SNI by adjusting parameter choices. Thus, changes at the protocol level suffice to achieve strong non-interference and allow the protection of MPC-in-the-head protocols right at the protocol level, without deeper changes to the implementations.
We showed that MPC-in-the-head protocols lend themselves to be easily protected against side-channel attacks. The essence of the scheme again relies on Boolean masking. This time we did not changed the masking but we changed the operations. The main idea of MPCitH protocols is to prove the correctness of the operations using only a subset of calculations. Our first challenge was to adapt the secure operations to the MPCitH paradigm. To satisfy this we proposed balanced gadgets, that is, every branch depends on the same number of other branches. Therefore, the new scheme reveals only a subset of branches in such a way that the disclosed branches still provides security against SCA. We verified this via a simulation and showed that the resulting overheads are comparably low.

In the last part of this thesis, in Chapter 8, we focused on another member of Picnic signatures, Picnic3 which uses MPCitH paradigm with preprocessing. As shown in Chapter 6, KKW protocol as implemented within Picnic signature schemes, is vulnerable to side-channel attacks. Although the SNIitH approach protects the online phase of the signature, the offline phase cannot be saved with the same approach. Therefore we asked by the following equation:

> *Can we mask signature generation in signature schemes constructed with the MPC-in-the-head-with-preprocessing paradigm in a provably secure manner, without modifying the verification algorithm?*

On a high-level, in our masking countermeasure, the prover essentially *shares the shares*. Concretely, each party's share is split again into some shares and every party internally does their computation in a masked way. Accordingly, all the views are maintained in a secret-shared form, until the prover learns challenge. Once prover obtained the challenge, either prover can keep them in secret-shared form when offline phase is revealed, or prover only needs to reconstruct views of opened parties when online phase is revealed. This way, even if the adversary gets information of some share, there's always at least one share of the view that remains completely hidden. However this was not enough to provide a secure signature scheme as %70 percent of Picnic3 is the hash function: Keccak. Thus we also provided a masked Keccak implementation and designed our masked Picnic3.

We showed that our masked Keccak and Picnic3 satisfies the formal security notion (NIo) and provide a formal proof with a verification in `MaskVerif`. To ensure the security of our design and perform a benchmarking we implemented the masked Picnic3 signature scheme on ARM CORTEX M4 as our practical setup. We carried our a comprehensive leakage analysis to masked Picnic3 and Keccak to confirm the absence of leakage. Furthermore, while analyzing the overhead of our design, we observed that the cost of masking in range of 1.8-5.4 depending on the masking technique of the hash function.

The first order masked Picnic3 and SHA-3 implementation and the formal verification codes for `MaskVerif` are publicly available under `crypto_sign/picnic3l1/masked/` of the Picnic-M4 GitHub repository.

https://github.com/dkales/picnic_m4

In conclusion, this work tackles the problem of unification of side-channel countermeasures with respect to various attacks and with respect to different adversarial models. We

showed efficient ways of combining countermeasures. More importantly we proved the security of our schemes using information theoretic notions instead of ad-hoc methods. Therefore our solutions capture the essence of combined countermeasures and provide an advanced way of providing countermeasures for modern cryptography. Moreover we presented well-known attacks on post-quantum primitives which are assumed to be secure. This way we marked that SCA is still a strong method for an adversary and it cannot be ignored. Also we present the ways of protecting post-quantum primitives against the well-known attacks that can be efficiently done by masking which brings attention to SCA in the post-quantum world.

We would like to conclude this thesis as we started: there is always an adversary who tries to crack *shell* and reach the *golden treasure*, therefore we need to improve our understanding of countermeasures and harden our *shell*. We cannot rely on ad-hoc methods to protect our sensitive data, as the capabilities of an adversary rapidly increase and the attacks become cheaper which makes the attacks more accessible. We need to be ready for more advanced attacks against modern cryptography. Moreover we need to make sure that the algorithms that are going to be used in the near future are resistant to attacks that have been in the literature for over a decade.

Part III

Appendix

Appendix A

Additional Proofs and Example Constructions for Combined White-box Masking

Lemma 4.1: Correctness of Circuit Transformation $\mathtt{T}^{(n,d)}$

Proof. For simplicity, let us denote $\mathtt{Encode}$ as:

$$\mathtt{Encode}(x, \tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_{n-1}) = \mathtt{Encode}(x).$$

Next we prove the functionality preserving property of each gadget.

- $x = \mathtt{Decode}(\mathtt{RefreshMask}(\mathtt{Encode}(x))$
  $= \mathtt{Decode}(\mathtt{RefreshMask}((\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n))$
  $= \mathtt{Decode}((\tilde{x}_0 \oplus \tilde{r}_0), \ldots, (\tilde{x}_d \oplus \tilde{r}_d), (x_1 \oplus \bigoplus_{j=2}^{n} r_{1,j}), (r_{1,2} \oplus x_2 \bigoplus_{j=3}^{n} r_{2,j}),$
  $\ldots, (\bigoplus_{i=1}^{n-1} r_{i,n} \oplus x_n \oplus \mathcal{W} \oplus \mathcal{R}))$
  $= (\tilde{x}_0 \oplus \tilde{r}_0) \cdots (\tilde{x}_d \oplus \tilde{r}_d) \oplus x_1 \oplus \cdots \oplus x_n \oplus \mathcal{W} \oplus \mathcal{R}$
  $= \tilde{x}_0 \cdots \tilde{x}_d \oplus \mathcal{W}' \oplus x_1 \oplus \cdots \oplus x_n \oplus \mathcal{W} \oplus \mathcal{R}$
  $= \tilde{x}_0 \cdots \tilde{x}_d \oplus x_1 \oplus \cdots \oplus x_n$
  $= x$

- $\mathtt{Decode}(\mathtt{Xor}(\mathtt{Encode}(x), \mathtt{Encode}(y)))$
  $= \mathtt{Decode}(\tilde{x}_0 \oplus \tilde{y}_0, \ldots, \tilde{x}_d \oplus \tilde{y}_d, x_1 \oplus y_1, \ldots, x_{n-1} \oplus y_{n-1}, x_n \oplus y_n \oplus \mathcal{U})$
  where $(\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n) = \mathtt{RefreshMask}(\mathtt{Encode}(x))$ and
  $(\tilde{y}_0, \ldots, \tilde{y}_d, y_1, \ldots, y_n) = \mathtt{RefreshMask}(\mathtt{Encode}(y))$.
  $= [(\tilde{x}_0 \oplus \tilde{y}_0) \cdots (\tilde{x}_d \oplus \tilde{y}_d)] \oplus [(x_1 \oplus y_1) \oplus \cdots \oplus (x_{n-1} \oplus y_{n-1}) \oplus (x_n \oplus y_n \oplus \mathcal{U})]$
  $= [\tilde{x}_0 \cdots \tilde{x}_d \oplus \mathcal{U} \oplus \tilde{y}_0 \cdots \tilde{y}_d] \oplus [(x_1 \oplus y_1) \oplus \cdots \oplus (x_{n-1} \oplus y_{n-1}) \oplus (x_n \oplus y_n \oplus \mathcal{U})]$
  $= (\tilde{x}_0 \cdots \tilde{x}_d \oplus x_1 \oplus \cdots x_n) \oplus (\tilde{y}_0 \cdots \tilde{y}_d \oplus y_1 \oplus \cdots y_n)$
  $= x \oplus y.$

- $xy = \texttt{Decode}(\texttt{And}(\texttt{Encode}(x), \texttt{Encode(y)}))$
  $= \texttt{Decode}(\texttt{And}((\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n), (\tilde{y}_0, \ldots \tilde{y}_d, y_1, \ldots, y_n)))$
  where $(\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n) = \texttt{RefreshMask}(\texttt{Encode}(x))$ and
  $(\tilde{y}_0, \ldots, \tilde{y}_d, y_1, \ldots, y_n) = \texttt{RefreshMask}(\texttt{Encode}(y))$.
  $= \texttt{Decode}(\tilde{z}_0, \ldots, \tilde{z}_d, z_1, \ldots, z_n)$

  where the output shares can be listed as follows:

$$\tilde{z}_i = \tilde{x}_i \tilde{y}_{i'} \oplus r^{i,1} \oplus \cdots \oplus r^{i,n} \text{ for } 0 \leq i \leq d,$$
$$z_i = x_i y_i \oplus \bigoplus_{\substack{j=1 \\ j \neq i}}^{n} r_{i,j} \text{ for } 1 \leq i \leq n.$$

  Also, the values $r_{i,j}$ can be listed as:

$$r_{0,j} = \mathcal{F}(x_j, y_j) = [r_{0,j} \oplus (\tilde{x}_0 \ldots \tilde{x}_d) y_j] \oplus x_j (\tilde{y}_0 \ldots \tilde{y}_d) \text{ for } 1 \leq j \leq n,$$
$$r_{i,j} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i \text{ for } 1 \leq i < j \leq n.$$

  Therefore,

$$\begin{aligned}
\texttt{Decode}(\bar{z}) &= \tilde{z}_0 \cdots \tilde{z}_d \oplus z_1 \oplus \ldots \oplus z_n \\
&= \prod_{i=0}^{d} \left[\tilde{x}_i \tilde{y}_{i'} \oplus r^{i,1} \oplus \cdots \oplus r^{i,n}\right] \oplus \bigoplus_{i=1}^{n} \left[x_i y_i \oplus \bigoplus_{\substack{j=0 \\ j \neq i}}^{n} r_{i,j} \right] \\
&= [(\tilde{x}_0 \cdots \tilde{x}_d)(\tilde{y}_0 \cdots \tilde{y}_d) \oplus \mathcal{V}] \oplus \left[\bigoplus_{i=1}^{n} (x_i y_i \oplus \bigoplus_{\substack{j=1 \\ j \neq i}}^{n} r_{i,j}) \right] \oplus \\
&\left[\bigoplus_{j=1}^{n} ((r_{0,j} \oplus (\tilde{x}_0 \ldots \tilde{x}_d) y_i) \oplus x_i (\tilde{y}_0 \ldots \tilde{y}_d)) \right] \\
&= [(\tilde{x}_0 \cdots \tilde{x}_d)(\tilde{y}_0 \cdots \tilde{y}_d) \oplus \mathcal{V}] \oplus \left[\bigoplus_{1 \leq i,j \leq n} x_i y_j \right] \oplus \\
&\left[\mathcal{V} \oplus \bigoplus_{i=1}^{n} (\tilde{x}_0 \ldots \tilde{x}_d) y_i \oplus x_i (\tilde{y}_0 \ldots \tilde{y}_d) \right]
\end{aligned}$$

$$= (\tilde{x}_0 \cdots \tilde{x}_d)(\tilde{y}_0 \cdots \tilde{y}_d) \oplus \bigoplus_{1 \le i,j \le n} x_i y_j \oplus \bigoplus_{i=1}^{n} ((\tilde{x}_0 \ldots \tilde{x}_d) y_i \oplus x_i (\tilde{y}_0 \ldots \tilde{y}_d))$$
$$= (\tilde{x}_0 \cdots \tilde{x}_d \oplus x_1 \oplus \cdots x_n)(\tilde{y}_0 \cdots \tilde{y}_d \oplus y_1 \oplus \cdots y_n)$$
$$= xy.$$

Hence we showed that the gadgets introduced in Section 4.2 are functionally preserving gadgets. Therefore, the transformation that generates an (n, d)-masked circuit is a functionally preserving transformation. □

Example A.1. $n = 2$, $d = 1$
Here is an example construction for the $(2, 1)$*-masking scheme:*

- $\texttt{Encode}(x, x_1, \tilde{x}_0, \tilde{x}_1) = (\tilde{x}_0, \tilde{x}_1, x_1, x_2)$ *where* $x_2 = \tilde{x}_0\tilde{x}_1 \oplus x_1 \oplus x$.
- $\texttt{Decode}(\overline{x}) = \tilde{x}_0\tilde{x}_1 \oplus x_1 \oplus x_2$.
- $\texttt{Xor}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, z_1, z_2)$ *such that* $z = x \oplus y$*:*
  - $\tilde{z}_0 = \tilde{x}_0 \oplus \tilde{y}_0$,
  - $\tilde{z}_1 = \tilde{x}_1 \oplus \tilde{y}_1$,
  - $z_1 = x_1 \oplus y_1$,
  - $z_2 = x_2 \oplus y_2 \oplus \tilde{x}_1\tilde{y}_0 \oplus \tilde{x}_0\tilde{y}_1$.
- $\texttt{And}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, z_1, z_2)$ *such that* $z = xy$*;*

  Step 1: *First, calculate the multiplicative representations of the output share* z_0*:*
  - $\tilde{z}_0 = \tilde{x}_0\tilde{y}_1 \oplus r^{0,1} \oplus r^{0,2}$,
  - $\tilde{z}_1 = \tilde{x}_1\tilde{y}_0 \oplus r^{1,1} \oplus r^{1,2}$ *where* $(r^{0,1}, r^{0,2}, r^{1,1}, r^{1,2}) \leftarrow \texttt{rand}(0, 1)$

  Step 2(a): *Calculate the intermediate values* $r_{j,0}$ *which include the reconstruction of the values* x_0 *and* y_0*:*
  - $r_{1,0} = \tilde{x}_1(\tilde{x}_0 y_1 \oplus r^{0,1}\tilde{y}_0) \oplus \tilde{y}_1(\tilde{y}_0 x_1 \oplus r^{1,1}\tilde{x}_0) \oplus r^{1,1}(r^{0,1} \oplus r^{0,2})$,
  - $r_{2,0} = \tilde{x}_1(\tilde{x}_0 y_2 \oplus r^{0,2}\tilde{y}_0) \oplus \tilde{y}_1(\tilde{y}_0 x_2 \oplus r^{1,2}\tilde{x}_0) \oplus r^{1,2}(r^{0,1} \oplus r^{0,2})$.

  Step 2(b): *Calculate the intermediate values* $r_{j,0}$ *which do not include the reconstruction of the values* x_0 *and* y_0*:*
  - $r_{1,2} \leftarrow \texttt{rand}(0, 1)$,
  - $r_{2,1} = (r_{1,2} \oplus x_1 y_2) \oplus x_2 y_1$.

Step 3: *Finally, calculate the rest of the shares:*

- $z_1 = x_1 y_1 \oplus r_{1,0} \oplus r_{1,2}$,
- $z_2 = x_2 y_2 \oplus r_{2,0} \oplus r_{2,1}$.

- $\texttt{RefreshMask}(\overline{x}) = (\tilde{x}_0, \tilde{x}_1, x_1, x_2)$

  1. *First, calculate the non-linear components of the output share* x_0*:*
     - $\tilde{x}_0 = \tilde{x}_0 \oplus \tilde{r}_0$,
     - $\tilde{x}_1 = \tilde{x}_1 \oplus \tilde{r}_1$ *where* $(\tilde{r}_0, \tilde{r}_1) \leftarrow \texttt{rand}(0,1)$
  2. *Calculate the rest the linear masks:*
     - $x_1 = x_1 \oplus r_1$,
     - $x_2 = x_2 \oplus r_1$ *where* $r_1 \leftarrow \texttt{rand}(0,1)$
  3. *Select a random bit* $r_0 \leftarrow \texttt{rand}(0,1)$ *and calculate the intermediate variable with* $\mathcal{W}$ *and* $\mathcal{R}$ *:*
     - $\mathcal{W} = \tilde{r}_0(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0)$ *and* $\mathcal{R} = (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0) \oplus r_0$ *where*
     - $x_2 = x_2 \oplus \mathcal{W} \oplus \mathcal{R}$

Example A.2. *Example:* $n = 2$, $d = 2$
Here is an example construction for the $(2,2)$*-masking scheme:*

- $\texttt{Encode}(x, x_1, \tilde{x}_0, \tilde{x}_1, \tilde{x}_2) = (\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, x_1, x_2)$ *where* $x_2 = \tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus x_1 \oplus x$.
- $\texttt{Decode}(\overline{x}) = \tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus x_1 \oplus x_2$.
- $\texttt{Xor}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, z_1, z_2)$ *such that* $z = x \oplus y$
  - $\tilde{z}_i = \tilde{x}_i \oplus \tilde{y}_i$ *for* $i = \{0,1,2\}$
  - $z_1 = x_1 \oplus y_1$
  - $z_2 = x_2 \oplus y_2 \oplus \tilde{x}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{y}_2(\tilde{x}_0 \oplus \tilde{y}_0)) \oplus \tilde{y}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{x}_0(\tilde{x}_2 \oplus \tilde{y}_2))$
- $\texttt{And}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, z_1, z_2)$ *such that* $z = xy$

  Step 1: *First, calculate the multiplicative representations of the output share* z_0*:*
  - $\tilde{z}_0 = \tilde{x}_0\tilde{y}_1 \oplus r^{0,1} \oplus r^{0,2}$,
  - $\tilde{z}_1 = \tilde{x}_1\tilde{y}_2 \oplus r^{1,1} \oplus r^{1,2}$,
  - $\tilde{z}_2 = \tilde{x}_2\tilde{y}_0 \oplus r^{2,1} \oplus r^{2,2}$ *where* $r^{i,j} \leftarrow \texttt{rand}(0,1)$ *for* $i = \{0,1,2\}$, $j = \{1,2\}$.

Step 2(a): *Calculate the intermediate values $r_{j,0}$ where the combination of random nodes are defined as; $u = (r^{1,1} \oplus r^{1,2})$ and $v = (r^{2,1} \oplus r^{2,2})$.*

- $r_{1,0} = \mathcal{F}(x_1, y_1) = \tilde{x}_0 \left[\tilde{x}_2(\tilde{x}_1 y_1 \oplus r^{0,1}\tilde{y}_0) \oplus r^{1,1} v \tilde{y}_1\right] \oplus$
  $\tilde{y}_0 \left[\tilde{y}_1(\tilde{y}_2 x_1 \oplus r^{1,1}\tilde{x}_2) \oplus r^{0,1} u \tilde{x}_2\right] \oplus$
  $\tilde{x}_0\tilde{y}_1(r^{1,1}\tilde{x}_2\tilde{y}_0 \oplus r^{2,1}\tilde{x}_1\tilde{y}_2) \oplus r^{0,1}\tilde{x}_1\tilde{y}_2(v \oplus \tilde{x}_2\tilde{y}_0) \oplus$
  $\tilde{x}_2\tilde{y}_0(r^{0,1}\tilde{x}_0 \oplus r^{1,1}\tilde{y}_1) \oplus uvr^{0,1}$.
- $r_{2,0} = \mathcal{F}(x_2, y_2) = \tilde{x}_0 \left[\tilde{x}_2(\tilde{x}_1 y_2 \oplus r^{0,2}\tilde{y}_0) \oplus r^{1,2} v \tilde{y}_1\right] \oplus$
  $\tilde{y}_0 \left[\tilde{y}_1(\tilde{y}_2 x_2 \oplus r^{1,2}\tilde{x}_2) \oplus r^{0,2} u \tilde{x}_2\right] \oplus$
  $\tilde{x}_0\tilde{y}_1(r^{1,2}\tilde{x}_2\tilde{y}_0 \oplus r^{2,2}\tilde{x}_1\tilde{y}_2) \oplus r^{0,2}\tilde{x}_1\tilde{y}_2(v \oplus \tilde{x}_2\tilde{y}_0) \oplus$
  $\tilde{x}_2\tilde{y}_0(r^{0,2}\tilde{x}_0 \oplus r^{1,2}\tilde{y}_1) \oplus uvr^{0,2}$.

Step 2(b): *Calculate the intermediate values $r_{j,0}$ which do not include the reconstruction of the values x_0 and y_0:*

- $r_{1,2} \leftarrow \texttt{rand}(0, 1)$,
- $r_{2,1} = (r_{1,2} \oplus x_1 y_2) \oplus x_2 y_1$.

Step 3: *Finally, calculate the rest of the shares:*

- $z_1 = x_1 y_1 \oplus r_{1,0} \oplus r_{1,2}$,
- $z_2 = x_2 y_2 \oplus r_{2,0} \oplus r_{2,1}$.

- $\texttt{RefreshMask}(\overline{x}) = (\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, x_1, x_2)$

  1. *First, calculate the multiplicative representations of the output share x_0:*
     - $\tilde{x}_0 = \tilde{x}_0 \oplus \tilde{r}_0$,
     - $\tilde{x}_1 = \tilde{x}_1 \oplus \tilde{r}_1$,
     - $\tilde{x}_2 = \tilde{x}_2 \oplus \tilde{r}_2$, *where* $(\tilde{r}_0, \tilde{r}_1, \tilde{r}_2) \leftarrow \texttt{rand}(0, 1)$
  2. *Calculate the rest the linear shares:*
     - $x_1 = x_1 \oplus r_1$,
     - $x_2 = x_2 \oplus r_1$ *where* $r_1 \leftarrow \texttt{rand}(0, 1)$
  3. *Select a random bit* $r_0 \leftarrow \texttt{rand}(0, 1)$ *and calculate the intermediate variable with* $\mathcal{W}$ *and* $\mathcal{R}$ *:*

- $\mathcal{W} = [\tilde{r}_2(\tilde{x}_0 \oplus r_0)][\tilde{r}_1 \oplus (\tilde{x}_1 \oplus r_0)] \ \oplus [\tilde{r}_1(\tilde{x}_2 \oplus r_0)][\tilde{r}_0 \oplus (\tilde{x}_0 \oplus r_0)] \oplus$
  $[\tilde{r}_0(\tilde{x}_1 \oplus r_0)][\tilde{r}_2 \oplus (\tilde{x}_2 \oplus r_0)]$
  $\mathcal{R} = (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0)(\tilde{r}_2 \oplus r_0) \oplus$
  $r_0\tilde{r}_2(\tilde{x}_0 \oplus r_0) \oplus r_0\tilde{r}_1(\tilde{x}_0 \oplus r_0) \oplus r_0\tilde{r}_0(\tilde{x}_1 \oplus r_0) \oplus$
  $r_0\tilde{r}_2(\tilde{x}_1 \oplus r_0) \oplus r_0\tilde{r}_1(\tilde{x}_2 \oplus r_0) \oplus r_0\tilde{r}_0(\tilde{x}_2 \oplus r_0).$
- $x_2 = x_2 \oplus \mathcal{W} \oplus \mathcal{R}$

Appendix B

Practical Setups

B.1 The experimental setup for SMC AES-128

Due to the rapid development environment and the omnipresence of ARM cores in embedded applications, we employed the NUCLEO-L053R8 board from STMicroelectronics to test our robust implementation as introduced in Chapter 3. It features a 32-bit ARM Cortex-M0+ microcontroller labelled STM32L053R8T6. It can reach a clock frequency of up to 32 MHz and it is equipped with a hardware random-number generator (RNG) capable of generating one 32-bit random number every 40 cycles. The RNG must run at 48MHz. Internal Phase-Locked Loop circuits (PLLs) can be used to match this frequency.
A particular feature of this development board is that it provides two contact points to measure the actual current consumption of the ARM chip. We took advantage of it to place a low-value resistor between the pins to measure the voltage drop for our side-channel analysis. The code was initially sketched in mbed and later migrated to ARM MDK-Lite (KEIL uVision 5.21), however, the code is architecture-independent.

Field Multiplication. This is a heavily used operation across the implementation. Even a small performance variation in this operation significantly affects the whole algorithm, thus it is very important to optimize this operation and consider different trade-offs. The following paragraphs briefly describe the four variations that are available in the implementation code. The slowest version of the multiplication is based on instructions only, with the minimum memory usage and in *constant time*. The result of the field multiplication is returned after 8 iterations, as given in the Algorithm 19.
A second version of this operation is a trade-off function that combines a precomputed 256-byte look-up table (LUT) and instructions. The only difference from the previous version is that the LUT contains all possible computations of v based on items 3, 4, and 5 from instruction only-multiplication Algorithm 19 as those three lines of code only depend on operand v.
The best time-memory trade-off field multiplication [GR16], known as the Exp-Log

Algorithm 19 GF(2^8) `Multiplication (instructions only).`

```
// z = h · v
for (i = 0 ; i < 8 ; i++)
    mask = −((h >> i)&1); // (1)
    z = z ^(mask & v); // (2)
    mask = −((v >> 7)&1);// (3)
    v <<= 1;// (4)
    v ^= mask & 0x1b;// (5)
return z
```

Algorithm 20 GF(2^8) `Multiplication (mixed).`

```
// z = h · v
for (i = 0 ; i < 8 ; i++)
    mask = −((h >> i)&1); // (1)
    z = z ^(mask & v); // (2)
    v = secondOp[v];// (3, 4, 5)
return z
```

multiplication, is derived from the logarithm property $vh = g^{log_g(v)+log_g(h)}$. An appropriate generator g must be selected to precompute the logarithm and exponentiation tables so the multiplication is reduced to three table look-ups and logical and arithmetic operations, especially required to check if any of the operands is zero.

Ultimately, the fastest instance of this operation is based on two pre-computed 4-kB LUTs. To generate the tables, one of the operands is split into its most-significant nibble and least-significant nibble $v = v_m 2^4 + v_l$, then every possible permutation of each nibble is multiplied times all possible permutations of the other operand $vh = v_m h 2^4 + v_l h$ but only the most-significant nibble multiplication is reduced modulo the irreducible polynomial. To get the result of the field multiplication, only two look-ups and one addition are required, however, this method is very expensive in terms of memory usage and thus kept outside our performance analysis.

Field Squaring. The implementation code features two ways of performing field squaring. The first one, as in the multiplication case, is based on instructions only; the second one is simply a 256-byte LUT of all possible square values of the input.

The following pseudo-code describes in detail the algorithm of the code-only field

squaring which can be conducted in a single line of C code.

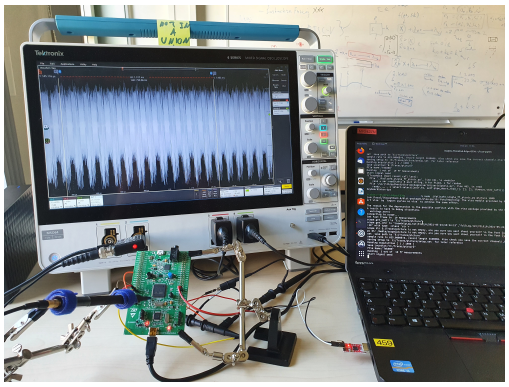
$$\begin{aligned} y^2 &= (y\&\texttt{0x01}) \oplus ((y\&\texttt{0x02}) << 1) \oplus ((y\&\texttt{0x04}) << 2) \oplus ((y\&\texttt{0x08}) << 3) \\ &\oplus (-((y\&\texttt{0x10}) >> 4)\&\texttt{0x1b}) \oplus (-((y\&\texttt{0x20}) >> 5)\&\texttt{0x6c}) \\ &\oplus (-((y\&\texttt{0x40}) >> 6)\&\texttt{0xab}) \oplus (-((y\&\texttt{0x80}) >> 7)\&\texttt{0x9a}). \end{aligned}$$

B.2 The experimental setup for Picnic

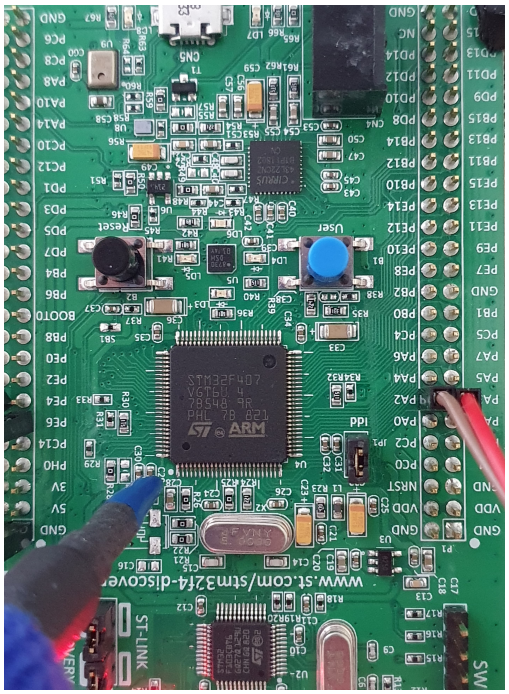
The reference implementation of Picnic is ported to the FRDM-K66F development board [fre]. The board features an NXP MK66FN2M0VMD18 Cortex-M4F MCU with 2MB flash and 256 KB SRAM, which we clocked at to 120 MHz. To measure the dynamic power consumption during Picnic, we collected 20,000 traces using a Langer EM Probe [EmP] placed 1 mm above the C37 0.1μF blocking capacitor of the FRDM-K66F board. Measurements were taken using a Tektronix MSO6 at 312.5 MHz sampling rate. Since the relevant part of the signature generation are the calls to LowMC, we placed a trigger before the start of the LowMC calls.

B.3 The experimental setup for Keccak and Picnic3

Our measurement setup comprises the STMicro developer board STM32F407G-DISC1 also used for the performance benchmarks, operated at 168MHz. We measure EM emanations using a Langer LF-U 2.5 near field probe connected to a Langer PA 303 preamplifier [EmP]. The EM probe is placed over the C29 blocking cap at a distance of approx. 1 mm. Measurements are recorded using a Tektronix MSO 6. For the Keccak implementation, we sampled at 3.125 GS/s with a 12 bit resolution and 200Mhz bandwidth. For the Picnic measurements, which are 2 orders of magnitude longer, we reduced the sampling rate to 625 MS/s in order to obtain feasible measurement time and storage sizes. Note that this still over-samples the board (168MHz) by a rate of 3.7 which is well above the minimal oversampling threshold of 2 from he Nyquist–Shannon sampling theorem.



(a) An overview of practical setup Picnic3. For our capture device we have used the Tektronix MSO6 and as our target device we have used the STM32 discovery board with an ARM Cortex M4 clocked at 168 MHz.



(b) A closer look to EM probe placement over the C29 blocking capacitor.

Figure B.1: The practical setup used for the leakage analysis and benchmarking for Picnic3

Appendix C

Details of Picnic Signatures

C.1 Complete Description of the KKW Proof System

In Fig. C.1 we present a three-round KKW proof system. We remark that commitments (i.e., generation of $\mathsf{com}_i^{(k)}$ and $\mathsf{com\_on}^{(k)}$) are de-randomized and replaced by a hash function as suggested in [KKW18, §3] (which loses HVZK but is still sufficient for provable security of signature), and the protocol is mildly generalized to work with arithmetic circuits according to [dDOS19, BN20].

C.2 Our Protected Picnic3 Implementation

In this section we give a detailed description of our masked Picnic3 implementation. Algorithm 21 has the top-level signature generation function, that calls the other algorithms in this section. Fig. C.2 gives an overview of the optimized hashing operations mentioned in Section 8.3.2.1, indicating which optimizations are applied to each one.

Notation. T is the number of shares used by our implementation, and masked values will be T-encoded. For LowMc, n is the blocksize, and the precomputed constants K_i, L_i and R_i are as defined in Section C.5. The parameter N is the number of MPC parties, and M is the number of MPC instances.

Data Types and Helper functions.

- T-encoding: an additive secret sharing in $GF(2)$. For a bit b, the T-encoding is a vector of T values $b_1, \ldots, b_T$ such that $b = \sum_{i=1}^{T} b_i$. For a bitstring s, the T-encoding is T bitstrings over $GF(2)^{|s|}$ that XOR to s. As in other parts of the paper we use $\langle b \rangle$ to indicate that b is T-encoded.
  - $\oplus_T$: XOR operation of T-encoded values. For two T-encoded inputs $\langle a \rangle$ and $\langle b \rangle$, we define $\langle c \rangle = \langle a \rangle \oplus_T \langle b \rangle$ as $c_i = a_i \oplus b_i$ for $i = 1, \ldots, T$; for one T-encoded input XOR'd by a non-encoded constant b, we define $\langle c \rangle = \langle a \rangle \oplus_T b$ as $c_1 = a_1 \oplus b$ and $c_i = a_i$ for $i = 2, \ldots, T$.

Protocol KKW

Inputs Both prover and verifier receive circuit C as a statement. The prover also holds a witness $\mathbf{w} = (w)_{w \in \mathsf{IN}}$ such that $C(\mathbf{w}) = 1$. Values M, N, τ are parameters of the protocol.

Commit

1. For each $k \in [M]$, the prover:
   a) Choose a uniform $\mathsf{seed}^{(k)}$ and use to generate values $\{\mathsf{seed}_i^{(k)}\}_{i\in[N]}$. Also the prover computes $\mathsf{aux}^{(k)} \in \mathbb{F}^{|C|}$ by running the offline phase of MPC Π_C^{off} on input $\{\mathsf{seed}_i^{(k)}\}_{i\in[N]}$. For all $i = 1, \ldots N-1$, let $\mathsf{state}_i^{(k)} = \mathsf{seed}_i^{(k)}$ and let $\mathsf{state}_N^{(k)} = \mathsf{seed}_N^{(k)} || \mathsf{aux}^{(k)}$.
   b) Commit to the offline phase:
   $$\mathsf{com}_i^{(k)} = \mathsf{H}(\mathsf{state}_i^{(k)}) \text{ for all } i \in [N]$$
   $$\mathsf{com\_off}^{(k)} = \mathsf{H}(\mathsf{com}_1^{(k)}, \ldots, \mathsf{com}_N^{(k)}).$$
   c) Compute the masked witness $\hat{w}^{(k)} = \lambda_1^w + \ldots + \lambda_N^w + w$ for each $w \in \mathsf{IN}$, where each λ_i^w corresponds to party P_i's random share to mask the witness, and is read out from $\mathsf{state}_i^{(k)}$.
   d) Simulate the online phase of the N-party protocol by running the offline phase of MPC Π_C^{on} on input $(\hat{w}^{(k)})_{w\in\mathsf{IN}}$ and $\{\mathsf{state}_i^{(k)}\}_{i\in[N]}$, to produce $\{\mathsf{msgs}_i^{(k)}\}_{i\in[N]}$.
   e) Commit to the online phase:
   $$\mathsf{com\_on}^{(k)} = \mathsf{H}(\{\hat{w}^{(k)}\}_{w\in\mathsf{IN}}, \mathsf{msgs}_1^{(k)}, \ldots, \mathsf{msgs}_N^{(k)}).$$
2. Compute $h_{\mathsf{off}} = \mathsf{H}(\mathsf{com\_off}^{(1)}, \ldots, \mathsf{com\_off}^{(M)})$ and $h_{\mathsf{on}} = \mathsf{H}(\mathsf{com\_on}^{(1)}, \ldots, \mathsf{com\_on}^{(M)})$ and send $h^* = \mathsf{H}(h_{\mathsf{off}}, h_{\mathsf{on}})$ to the verifier.

Challenge The prover receives the following challenges from the verifier: a uniform τ-sized set $\mathcal{C} \subset [M]$ and $\mathcal{P} = \{i_k\}_{k\in\mathcal{C}}$ where each $i_k \in [N]$ is uniform.

Response For each $k \in [M] \setminus \mathcal{C}$, the prover sends $\mathsf{seed}^{(k)}$ and $\mathsf{com\_on}^{(k)}$ for all to the verifier. For each $k \in \mathcal{C}$, the prover sends $\mathsf{com}_{i_k}^{(k)}$, $\{\mathsf{state}_i^{(k)}\}_{i\neq i_k}$, $\{\hat{w}^{(k)}\}_{w\in\mathsf{IN}}$ and $\mathsf{msgs}_{i_k}^{(k)}$ to the verifier.

Verification The verifier accepts if and only if all the following checks succeed:

1. Check the offline phase:
   a) For every $k \in \mathcal{C}$ and $i \neq i_k$, the verifier uses $\mathsf{state}_i^{(k)}$ to compute $\mathsf{com}_i^{(k)}$ as the prover would. Then produce $\mathsf{com\_off}^{(k)} = \mathsf{H}(\mathsf{com}_1^{(k)}, \ldots, \mathsf{com}_N^{(k)})$ using the received value $\mathsf{com}_{i_k}^{(k)}$.
   b) For every $k \in [M] \setminus \mathcal{C}$ the verifier uses $\mathsf{seed}^{(k)}$ to compute $\mathsf{com\_off}^{(k)}$ as the prover would.
   c) Finally, the verifier computes $h_{\mathsf{off}} = \mathsf{H}(\mathsf{com\_off}^{(1)}, \ldots, \mathsf{com\_off}^{(M)})$
2. Check the online phase:
   a) For $k \in \mathcal{C}$ the verifier simulates the online phase using $\{\mathsf{state}_i^{(k)}\}_{i\neq i_k}$, masked witness $(\hat{w}^{(k)})_{w\in\mathsf{IN}}$ and $\mathsf{msgs}_{i_k}^{(k)}$ to compute $\{\mathsf{msgs}_i\}_{i\neq i_k}$. Then compute $\mathsf{com\_on}^{(k)}$ as if the prover would do.
   b) The verifier computes $h_{\mathsf{on}} = \mathsf{H}(\mathsf{com\_on}^{(1)}, \ldots, \mathsf{com\_on}^{(M)})$ using the received $\mathsf{com\_on}^{(k)}$ for $k \in [M] \setminus \mathcal{C}$.
3. The verifier checks that $\mathsf{H}(h_{\mathsf{off}}, h_{\mathsf{on}}) \stackrel{?}{=} h^*$.

Figure C.1: 3-round KKW proof system for an arithmetic circuit C defined over $\mathbb{F}$.

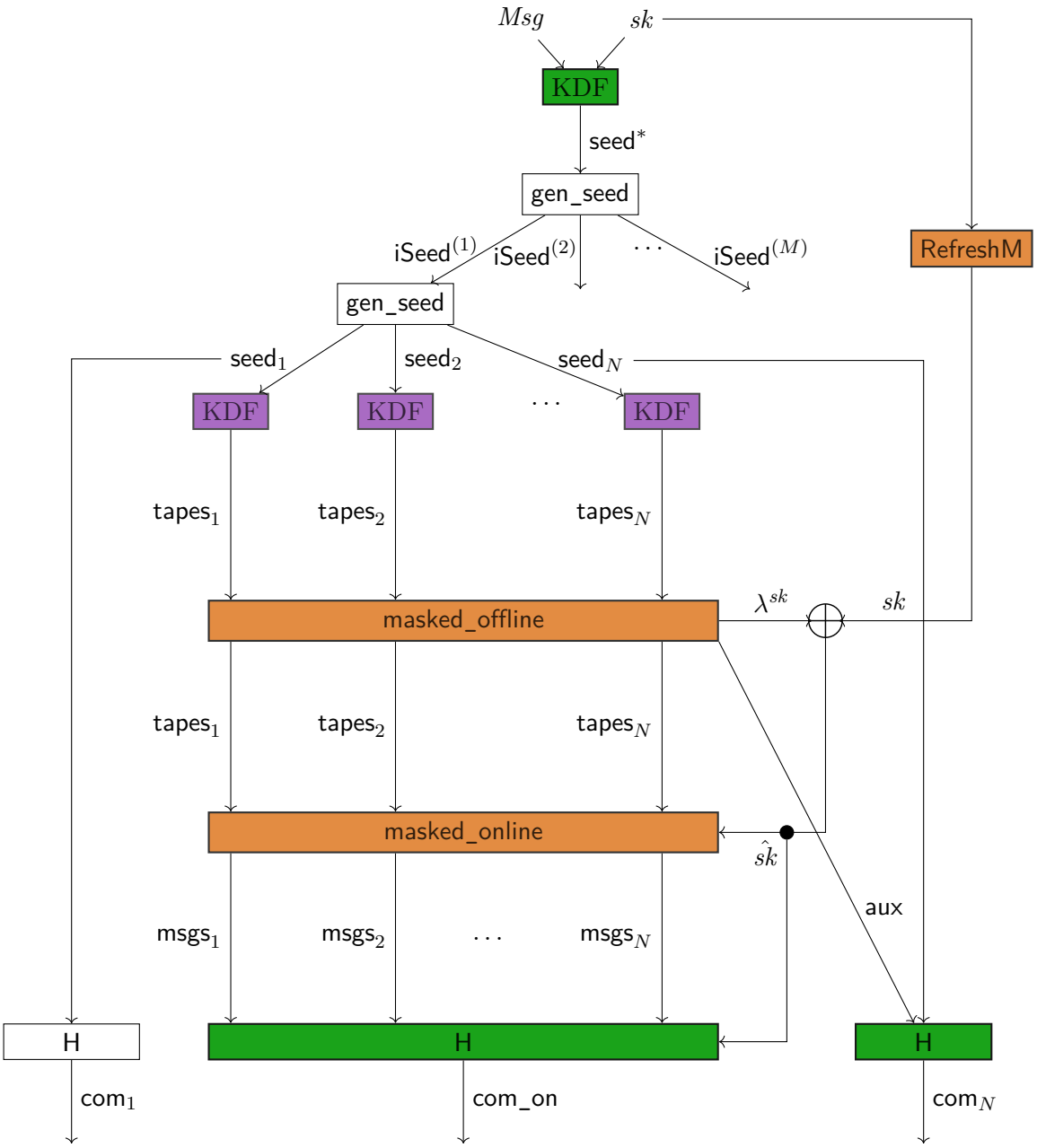


Figure C.2: Summary of our masking protections and hashing optimizations from Section 8.3.2. The figure is fully expanded for one of the M MPC instances, and shows the signer's operations for the commit phase of the protocol (i.e., before the challenge is computed). Hash functions in green are half-masked hash with sensitive inputs, hash functions in purple are half-masked with sensitive outputs, functions in orange are NI/SNI-secure gadgets, and the white functions are unprotected. The secret key (witness) is denoted sk and Msg is the message to be signed. In the figure, we omit hashing of $(\mathsf{com}_1, \ldots, \mathsf{com}_N)$ into com_off, since the inputs are public values that can be reconstructed from the signature, and therefore the hash computation is unmasked regardless of our optimization.

– SMul: AND operation of two T-encoded values. We use Algorithm 28.

For example, with 2-encoded inputs $\langle a \rangle$ and $\langle b \rangle$, this algorithm outputs $\langle c \rangle = \mathsf{SMul}(\langle a \rangle, \langle b \rangle)$ as

$$c_1 = a_1 b_1 + r$$
$$c_2 = a_2 b_2 + a_1 b_2 + r + a_2 b_1$$

where r is a fresh random bit. Note that $c = c_1 + c_2 = (a_1 + a_0)(b_1 + b_0) = ab$.

– Additional functions: Two additional helper functions from the literature are described in Section C.3. These are for refreshing the randomness of a T-encoded value and decoding (or unmasking) a T-encoded value.

- matMul_T: This is a generalization of the matMul matrix multiplication function in [Pic20, §6.4.4], modified to work on T-encoded input vectors. The matrix remains unshared. The input is a T-encoded vector v of length n, a $n \times n$ matrix M, and the output is the length-n vector vM. If $T = 2$, we have $v = (v_1, v_2)$, the output is $(v_1 M, v_2 M) = (\mathsf{matMul}(v_1, M), \mathsf{matMul}(v_2, M))$

- tapes: an object representing the N random tapes, one per party. In case we need to be explicit about individual per-party tapes, it is parsed as $\mathsf{tapes} = \mathsf{tape}_1 || \ldots, || \mathsf{tape}_N$. Each tape is expanded from a seed masked version of SHAKE that produces T-encoded outputs, and we store the T-encoded outputs. We also store a T-encoded representation of the the aux tape, the N-th party's share.

- $\mathsf{tapes\_to\_word}(\mathsf{tapes}, \mathsf{offset})$: Read one bit from each of the N tapes at the index offset, and output an N-bit word. When the tapes are T-encoded, the output word is also T-encoded.

- $\mathsf{tapes\_to\_parity}_T(n, \mathsf{tapes}, \mathsf{offset})$: Reads n bits from each tape at the index offset, the strings $s_1, \ldots, s_N$, returns a T-encoding of of $\bigoplus_{i=1}^{N} s_i$. Our implementation computes and stores a T-encoding of the parity tape (i.e., the XOR of all N tapes) and uses this to implement the tapes_to_parity function.

C.2.1 Specification of Fully Masked Picnic3

Algorithm 21 specifies the masked Picnic3 signing operations *without* the half-masked hashing optimizations that we described in Section 8.3.2. The notation is as defined elsewhere in the paper, and in the appendix on LowMc (Section C.5). All functions

marked in orange modify T-encoded values during the computation. The Unmask function takes a T-encoded value and returns the non-encoded value, by summing the shares after refresh (see Algorithm 30). We verified that Algorithm 21 is indeed NI secure with `maskVerif` for up to second order (implying it is also NIo for any public outputs).

C.2.2 Simulation of the Offline Phase

Algorithms 22 to 24 describe the protected preprocessing phase. The description is very similar to the preprocessing phase in the specification [Pic20, Section 7.4], however, the data types and helper functions are different. Primarily, all variables are T-encoded. Algorithm 23 describes our masked version of the LowMc S-box used for preprocessing (called by Algorithm 22), which in turn calls Algorithm 24 the AND operation for the preprocessing phase. Also, our presentation assumes that Algorithm 22 is used for signature generation only, since verification can use an unprotected implementation.

C.2.3 Simulation of the Online Phase

We now describe how the online phase of the MPC simulation is masked. Algorithm 25 is the MPC simulation for the online phase, implementing the LowMc circuit. For each AND gate, each party i broadcasts a bit and these are output to msgs_i, these are also T-encoded. In Algorithm 26 we describe the S-box implementation used in Algorithm 25. Finally we have Algorithm 27 that describes the online simulation of an individual AND gate. The broadcast values (written to msgs_i) and output bit are also T-encoded. Recall that SMul is implemented with the ISW multiplier (Algorithm 28).
Note that we need to refresh T-encoded st before each invocation of $\mathsf{masked\_sbox}_{\mathsf{online}}$ (Line 6 of Algorithm 25). On one hand, since every round of LowMc involves a linear transformation of st (Line 1 and Line 7), every bit of st depends on all n bits of the previous st, which corresponds to a problematic composition pattern mentioned in [BBD+15a, Diagram 1]. On the other hand, all the other gadgets are in fact characterized as an *affine gadget*, which can be security composed in an arbitrary fashion. Hence, inserting RefreshM as we do is necessary and sufficient for the entire construction to be provably NIo secure.

Storage of secret keys. We assume that the Picnic secret key (a bitstring of length n), is stored in a T-encoded representation. Picnic key pair generation may be modified to generate T-encoded secret keys, or an implementation may use regular key generation

in a trusted environment (e.g., during device manufacture), and then encode the secret key. As this is not important for performance, our implementation takes the regular key and T-encodes it at the beginning of signing. Then the input to MPC simulation is the T-encoded value $\langle \hat{sk} \rangle = \langle \lambda^{sk} \rangle \oplus_T \langle sk \rangle$, where $\langle \lambda^{sk} \rangle$ is the T-encoded random masks output by preprocessing.

Algorithm 21 masked_Sign

Input: Msg, pk and $\langle sk \rangle$ for each input wire w to the circuit.
Output: $(h, salt, Z)$

1: Sample random $R \in \{0,1\}^{2\kappa}$
2: $(\langle \mathsf{seed}^* \rangle, \langle salt \rangle) \leftarrow \mathsf{KDF}(\langle sk \rangle, Msg, pk, \lambda, R)$ and $salt \leftarrow \mathsf{Unmask}(\langle salt \rangle)$
3: $\langle \mathsf{iSeed\_tree} \rangle \leftarrow \mathsf{gen\_seed}(\langle \mathsf{seed}^* \rangle, salt, M, 0)$
4: $(\langle \mathsf{iSeed}^{(1)} \rangle, \ldots, \langle \mathsf{iSeed}^{(M)} \rangle) \leftarrow \mathsf{get\_leaves}(\langle \mathsf{iSeed\_tree} \rangle)$
5: **for** each $k \in [M]$:
6: $\quad \langle \mathsf{seed\_tree}^{(k)} \rangle \leftarrow \mathsf{gen\_seed}(\langle \mathsf{iSeed}^{(k)} \rangle, salt, N, k)$
7: $\quad (\langle \mathsf{seed}_1^{(k)} \rangle, \ldots, \langle \mathsf{seed}_N^{(k)} \rangle) \leftarrow \mathsf{get\_leaves}(\langle \mathsf{seed\_tree}^{(k)} \rangle)$
8: $\quad$ **for** each $i \in [N]$:
9: $\quad\quad \langle \mathsf{tape}_i^{(k)} \rangle \leftarrow \mathsf{KDF}(\langle \mathsf{seed}_i^{(k)} \rangle, salt, k, i)$
10: $\quad \langle \lambda^{sk} \rangle \leftarrow \mathsf{masked\_offline}(\langle \mathsf{tape}_1^{(k)} \rangle || \ldots || \langle \mathsf{tape}_N^{(k)} \rangle, pk)$ (see Algorithm 22)
11: $\quad \langle \mathsf{aux}^{(k)} \rangle \leftarrow \mathsf{get\_aux}(\langle \mathsf{tape}_N^{(k)} \rangle)$
12: $\quad$ **for** each $i \in [N]$:
13: $\quad\quad$ **if** $i \neq N$ **then**
14: $\quad\quad\quad \langle \mathsf{com}_i^{(k)} \rangle \leftarrow \mathsf{H}(\langle \mathsf{seed}_i^{(k)} \rangle, salt, k, i)$
15: $\quad\quad$ **else**
16: $\quad\quad\quad \langle \mathsf{com}_i^{(k)} \rangle \leftarrow \mathsf{H}(\langle \mathsf{seed}_i^{(k)} \rangle, \langle \mathsf{aux}^{(k)} \rangle, salt, k, i)$
17: $\quad\quad \mathsf{com}_i^{(k)} \leftarrow \mathsf{Unmask}(\langle \mathsf{com}_i^{(k)} \rangle)$
18: $\quad \mathsf{com\_off}^{(k)} \leftarrow \mathsf{H}(\mathsf{com}_1^{(k)}, \ldots, \mathsf{com}_N^{(k)})$
19: $\quad \langle sk \rangle \leftarrow \mathsf{RefreshM}(\langle sk \rangle)$
20: $\quad \langle \hat{sk}^{(k)} \rangle \leftarrow \langle sk \rangle \oplus_T \langle \lambda^{sk} \rangle$
21: $\quad (\langle \mathsf{msgs}_1^{(k)} \rangle, \ldots, \langle \mathsf{msgs}_N^{(k)} \rangle) \leftarrow \mathsf{masked\_online}(\langle \hat{sk}^{(k)} \rangle, \langle \mathsf{tape}_1^{(k)} \rangle || \ldots || \langle \mathsf{tape}_{N,j}^{(k)} \rangle, pk)$ (see Algorithm 25)
22: $\quad \langle \mathsf{com\_on}^{(k)} \rangle \leftarrow \mathsf{H}(\langle \hat{sk}^{(k)} \rangle, \langle \mathsf{msgs}_1^{(k)} \rangle, \ldots, \langle \mathsf{msgs}_N^{(k)} \rangle)$
23: $\quad \mathsf{com\_on}^{(k)} \leftarrow \mathsf{Unmask}(\langle \mathsf{com\_on}^{(k)} \rangle)$
24: $\mathsf{com\_on\_tree} \leftarrow \mathsf{build\_tree}(\mathsf{com\_on}^{(1)}, \ldots, \mathsf{com\_on}^{(M)})$
25: $h \leftarrow \mathsf{H}(\mathsf{com\_off}^{(1)}, \ldots, \mathsf{com\_off}^{(M)}, \mathsf{com\_on\_tree.root}, salt, pk, Msg)$
26: Parse h as $(\mathcal{C}, \mathcal{P})$ where $\mathcal{C} \subset [M]$ and $\mathcal{P} = \{i_k\}_{k \in \mathcal{C}}$, $i_k \in [N]$
27: $\mathsf{com\_on\_info} \leftarrow \mathsf{open\_tree}(\mathsf{com\_on\_tree}, M, \mathcal{C})$
28: $\langle \mathsf{iSeed\_info} \rangle \leftarrow \mathsf{reveal\_seed}(\langle \mathsf{iSeed\_tree} \rangle, M, \mathcal{C})$
29: $\mathsf{iSeed\_info} \leftarrow \mathsf{Unmask}(\langle \mathsf{iSeed\_info} \rangle)$
30: **for** each $k \in \mathcal{C}$:
31: $\quad \langle \mathsf{seed\_info}^{(k)} \rangle \leftarrow \mathsf{reveal\_seed}(\langle \mathsf{seed\_tree}^{(k)} \rangle, N, i_k)$
32: $\quad \mathsf{seed\_info}^{(k)} \leftarrow \mathsf{Unmask}(\langle \mathsf{seed\_info}^{(k)} \rangle)$; $\hat{sk}^{(k)} \leftarrow \mathsf{Unmask}(\langle \hat{sk}^{(k)} \rangle)$; $\mathsf{msgs}_{i_k}^{(k)} \leftarrow \mathsf{Unmask}(\langle \mathsf{msgs}_{i_k}^{(k)} \rangle)$
33: $\quad$ if $i_k = N$ then $\mathsf{aux}^{(k)} \leftarrow \perp$; otherwise $\mathsf{aux}^{(k)} \leftarrow \mathsf{Unmask}(\langle \mathsf{aux}^{(k)} \rangle)$
34: let $Z = (\mathsf{com\_on\_info}, \mathsf{iSeed\_info}, (\mathsf{seed\_info}^{(k)}, \mathsf{aux}^{(k)}, \hat{sk}^{(k)}, \mathsf{com}_{i_k}^{(k)}, \mathsf{msgs}_{i_k}^{(k)})_{k \in \mathcal{C}})$.
35: output $(h, salt, Z)$ as a signature

Algorithm 22 masked_offline (corresponds to `compute_aux` in Section 7.4 of [Pic20])

Input: The tapes $\langle \mathsf{tapes} \rangle$. The signer's public key $pk = (c, p)$.
Output: The n-bit key mask $\langle \lambda^{sk} \rangle$. The $\langle \mathsf{tapes} \rangle$ is updated inside $\mathsf{masked\_sbox}_{\mathsf{aux}}$.

$\langle \mathsf{roundkey}_0 \rangle \leftarrow \mathsf{tapes\_to\_parity}_T(n, \langle \mathsf{tapes} \rangle, 0)$
$\langle \lambda^{sk} \rangle \leftarrow \mathsf{matMul}_T(\langle \mathsf{roundkey}_0 \rangle, K_0^{-1})$
$\langle \mathsf{st\_in} \rangle \leftarrow \langle 0^n \rangle$
for each LowMc round i from r down to 1
    $\langle \mathsf{roundkey}_i \rangle \leftarrow \mathsf{matMul}_T(\langle \lambda^{sk} \rangle, K_i)$
    $\langle \mathsf{st\_out} \rangle \leftarrow \langle \mathsf{st\_in} \rangle \oplus_T \langle \mathsf{roundkey}_i \rangle$
    $\langle \mathsf{st\_out} \rangle \leftarrow \mathsf{matMul}_T(\langle \mathsf{st\_out} \rangle, L_i^{-1})$
    if $i = 1$ **then**
        $\langle \mathsf{st\_in} \rangle \leftarrow \langle \mathsf{roundkey}_0 \rangle$
    else
        $\langle \mathsf{st\_in} \rangle \leftarrow \mathsf{tapes\_to\_parity}_T(n, \langle \mathsf{tapes} \rangle, 2n(i-1))$
        $\mathsf{offset} \leftarrow 2n(i-1) + n$
    $\mathsf{masked\_sbox}_{\mathsf{aux}}(\langle \mathsf{st\_in} \rangle, \langle \mathsf{st\_out} \rangle, \langle \mathsf{tapes} \rangle, \mathsf{offset})$ (see Algorithm 23)
return $\langle \lambda^{sk} \rangle$

Algorithm 23 $\mathsf{masked\_sbox}_{\mathsf{aux}}$ (corresponds to `aux_sbox` in Section 7.4.1 of [Pic20])

Input: The input and output states $\langle \mathsf{st\_in} \rangle$ and $\langle \mathsf{st\_out} \rangle$. The tapes $\langle \mathsf{tapes} \rangle$. The tape offset offset.
Output: tapes is updated with the auxiliary bits..

for each i from 0 to $3s$, in steps of 3
    $\langle \lambda^a \rangle \leftarrow \langle \mathsf{st\_in}[i+2] \rangle$ // T bits of st_in at position $i+2$, i.e., $\langle \lambda^a \rangle = (\mathsf{st\_in}_1[i+2], \ldots, \mathsf{st\_in}_T[i+2])$
    $\langle \lambda^b \rangle \leftarrow \langle \mathsf{st\_in}[i+1] \rangle$
    $\langle \lambda^c \rangle \leftarrow \langle \mathsf{st\_in}[i] \rangle$
    $\langle \lambda^d \rangle \leftarrow \langle \mathsf{st\_out}[i+2] \rangle$
    $\langle \lambda^e \rangle \leftarrow \langle \mathsf{st\_out}[i+1] \rangle$
    $\langle \lambda^f \rangle \leftarrow \langle \mathsf{st\_out}[i] \rangle$
    $\langle \lambda^{z_{bc}} \rangle \leftarrow \langle \lambda^d \rangle \oplus_T \langle \lambda^a \rangle$
    $\langle \lambda^{z_{ca}} \rangle \leftarrow \langle \lambda^e \rangle \oplus_T \langle \lambda^a \rangle \oplus_T \langle \lambda^b \rangle$
    $\langle \lambda^{z_{ab}} \rangle \leftarrow \langle \lambda^f \rangle \oplus_T \langle \lambda^a \rangle \oplus_T \langle \lambda^b \rangle \oplus_T \langle \lambda^c \rangle$
    $\mathsf{masked\_AND}_{\mathsf{aux}}(\langle \lambda^b \rangle, \langle \lambda^c \rangle, \langle \lambda^{z_{bc}} \rangle, \langle \mathsf{tapes} \rangle, \mathsf{offset} + i + 2)$
    $\mathsf{masked\_AND}_{\mathsf{aux}}(\langle \lambda^c \rangle, \langle \lambda^a \rangle, \langle \lambda^{z_{ca}} \rangle, \langle \mathsf{tapes} \rangle, \mathsf{offset} + i + 1)$
    $\mathsf{masked\_AND}_{\mathsf{aux}}(\langle \lambda^a \rangle, \langle \lambda^b \rangle, \langle \lambda^{z_{ab}} \rangle, \langle \mathsf{tapes} \rangle, \mathsf{offset} + i)$ (see Algorithm 24)

Algorithm 24 $\mathsf{masked\_AND}_{\mathsf{aux}}$ (corresponds to `aux_AND` in [Pic20, §7.4.2])

Input: The input mask bits $\langle \lambda^x \rangle$ and $\langle \lambda^y \rangle$. The fresh output mask bit $\langle \lambda^z \rangle$. The tapes $\langle \mathsf{tapes} \rangle$. The tape offset offset.
Output: The function updates $\langle \mathsf{tape}_N \rangle$.

$\langle \mathsf{and\_helper}' \rangle \leftarrow \mathsf{tapes\_to\_parity}_T(1, \langle \mathsf{tape}_1 \rangle || \ldots || \langle \mathsf{tape}_{N-1} \rangle, \mathsf{offset})$
$\langle \lambda^{xy} \rangle \leftarrow \mathsf{SMul}(\langle \lambda^x \rangle, \langle \lambda^y \rangle)$
$\langle \mathsf{aux\_bit} \rangle \leftarrow \langle \lambda^{xy} \rangle \oplus_T \langle \mathsf{and\_helper}' \rangle \oplus_T \langle \lambda^z \rangle$
$\langle \mathsf{tape}_N[\mathsf{offset}] \rangle \leftarrow \langle \mathsf{aux\_bit} \rangle$ // Ensuring $\lambda^{xy} \oplus_T \lambda^z = \mathsf{tape}_1[\mathsf{offset}] \oplus_T \ldots \oplus_T \mathsf{tape}_N[\mathsf{offset}]$

Algorithm 25 masked_online (corresponds to `mpc_simulate` in Section 7.5 of [Pic20])

Input: The masked input $\langle \hat{sk} \rangle$. The tapes $\langle \mathsf{tapes} \rangle$. The signer's public key $pk = (c, p)$.
Output: The broadcast messages $\langle \mathsf{msgs}_1 \rangle, \ldots, \langle \mathsf{msgs}_N \rangle$

$\langle \mathsf{roundkey}_0 \rangle \leftarrow \mathsf{matMul}_T(\langle \hat{sk} \rangle, K_0)$
$\langle \mathsf{st} \rangle \leftarrow \langle \mathsf{roundkey}_0 \rangle \oplus_T p$
$\langle \mathsf{st} \rangle \leftarrow \mathsf{RefreshM}(\langle \mathsf{st} \rangle)$
Initialize empty arrays $\langle \mathsf{msgs}_1 \rangle, \ldots, \langle \mathsf{msgs}_N \rangle$
for each LowMc round i from 1 to r
    $\mathsf{masked\_sbox}_{\mathsf{online}}(\langle \mathsf{st} \rangle, \langle \mathsf{tapes} \rangle, \langle \mathsf{msgs}_1 \rangle, \ldots, \langle \mathsf{msgs}_N \rangle, 2n(i-1))$ (see Algorithm 26)
    $\langle \mathsf{st} \rangle \leftarrow \mathsf{matMul}_T(\langle \mathsf{st} \rangle, L_i)$
    $\langle \mathsf{st} \rangle \leftarrow \langle \mathsf{st} \rangle \oplus_T R_i$
    $\langle \mathsf{roundkey}_i \rangle \leftarrow \mathsf{matMul}_T(\langle \hat{sk} \rangle, K_i)$
    $\langle \mathsf{st} \rangle \leftarrow \langle \mathsf{st} \rangle \oplus_T \langle \mathsf{roundkey}_i \rangle$
    $\langle \mathsf{st} \rangle \leftarrow \mathsf{RefreshM}(\langle \mathsf{st} \rangle)$
Compare $\mathsf{st} = \mathsf{Unmask}(\langle \mathsf{st} \rangle)$ and c component of pk. If they differ, fail.
return $(\langle \mathsf{msgs}_1 \rangle, \ldots, \langle \mathsf{msgs}_N \rangle)$

Algorithm 26 $\mathsf{masked\_sbox}_{\mathsf{online}}$ (corresponds to `mpc_sbox3` in Section 7.5.1 of [Pic20])

Input: A T-encoding of n-bit LowMc state $\langle \mathsf{st} \rangle$. The tapes $\langle \mathsf{tapes} \rangle$. The broadcast message holder $\langle \mathsf{msgs}_1 \rangle, \ldots, \langle \mathsf{msgs}_N \rangle$. The tape offset offset.
Output: $\langle \mathsf{msgs}_i \rangle$ is updated with the broadcast messages of party i

for each i from 0 to $3s$, in steps of 3
    $\langle \hat{a} \rangle \leftarrow \langle \mathsf{st}[i+2] \rangle$
    $\langle \hat{b} \rangle \leftarrow \langle \mathsf{st}[i+1] \rangle$
    $\langle \hat{c} \rangle \leftarrow \langle \mathsf{st}[i] \rangle$
    $(\langle \lambda_1^a \rangle, \ldots, \langle \lambda_N^a \rangle) \leftarrow \mathsf{tapes\_to\_word}(\langle \mathsf{tapes} \rangle, \mathsf{offset} + i + 2)$
    $(\langle \lambda_1^b \rangle, \ldots, \langle \lambda_N^b \rangle) \leftarrow \mathsf{tapes\_to\_word}(\langle \mathsf{tapes} \rangle, \mathsf{offset} + i + 1)$
    $(\langle \lambda_1^c \rangle, \ldots, \langle \lambda_N^c \rangle) \leftarrow \mathsf{tapes\_to\_word}(\langle \mathsf{tapes} \rangle, \mathsf{offset} + i)$
    $\langle \widehat{bc} \rangle \leftarrow \mathsf{masked\_AND}_{\mathsf{online}}(\langle \hat{b} \rangle, \langle \hat{c} \rangle, (\langle \lambda_i^b \rangle, \langle \lambda_i^c \rangle, \langle \mathsf{tape}_i \rangle, \langle \mathsf{msgs}_i \rangle)_{i \in [N]}, \mathsf{offset} + n + i + 2)$
    $\langle \widehat{ca} \rangle \leftarrow \mathsf{masked\_AND}_{\mathsf{online}}(\langle \hat{c} \rangle, \langle \hat{a} \rangle, (\langle \lambda_i^c \rangle, \langle \lambda_i^a \rangle, \langle \mathsf{tape}_i \rangle, \langle \mathsf{msgs}_i \rangle)_{i \in [N]}, \mathsf{offset} + n + i + 1)$
    $\langle \widehat{ab} \rangle \leftarrow \mathsf{masked\_AND}_{\mathsf{online}}(\langle \hat{a} \rangle, \langle \hat{b} \rangle, (\langle \lambda_i^a \rangle, \langle \lambda_i^b \rangle, \langle \mathsf{tape}_i \rangle, \langle \mathsf{msgs}_i \rangle)_{i \in [N]}, \mathsf{offset} + n + i)$
    $\langle \mathsf{st}[i+2] \rangle \leftarrow \langle \hat{a} \rangle \oplus_T \langle \widehat{bc} \rangle$
    $\langle \mathsf{st}[i+1] \rangle \leftarrow \langle \hat{a} \rangle \oplus_T \langle \hat{b} \rangle \oplus_T \langle \widehat{ca} \rangle$
    $\langle \mathsf{st}[i] \rangle \leftarrow \langle \hat{a} \rangle \oplus_T \langle \hat{b} \rangle \oplus_T \langle \hat{c} \rangle \oplus_T \langle \widehat{ab} \rangle$

Algorithm 27 $\mathsf{masked\_AND}_{\mathsf{online}}$ (corresponds to `mpc_and3` in Section 7.5.2 of [Pic20])

Input: A T-encoding of two masked input bits $\langle \hat{x} \rangle$ and $\langle \hat{y} \rangle$. The masking bits words $(\langle \lambda_1^x \rangle, \ldots, \langle \lambda_N^x \rangle)$ and $(\langle \lambda_1^y \rangle, \ldots, \langle \lambda_N^y \rangle)$. The tapes $\langle \mathsf{tapes} \rangle$. The message holder $\langle \mathsf{msgs}_1 \rangle, \ldots, \langle \mathsf{msgs}_N \rangle$. The tape offset offset.
Output: Masked AND output $\langle \widehat{xy} \rangle$. The function updates msgs.

// $\mathsf{and\_helper}_i$ contains party i's share of $\lambda^{xy} \oplus_T \lambda^z$
$(\langle \mathsf{and\_helper}_1 \rangle, \ldots, \langle \mathsf{and\_helper}_N \rangle) \leftarrow \mathsf{tapes\_to\_word}(\langle \mathsf{tapes} \rangle, \mathsf{offset})$
for each $i \in [N]$
    $\langle a_i \rangle \leftarrow \mathsf{SMul}(\langle \hat{x} \rangle, \langle \lambda_i^y \rangle)$
    $\langle b_i \rangle \leftarrow \mathsf{SMul}(\langle \hat{y} \rangle, \langle \lambda_i^x \rangle)$
    $\langle s_i \rangle \leftarrow \langle a_i \rangle \oplus_T \langle b_i \rangle \oplus_T \langle \mathsf{and\_helper}_i \rangle$
    Append $\langle s_i \rangle$ to $\langle \mathsf{msgs}_i \rangle$
$\langle c \rangle \leftarrow \mathsf{SMul}(\langle \hat{x} \rangle, \langle \hat{y} \rangle)$
$\langle s \rangle \leftarrow \sum_{i \in [N]} \langle s_i \rangle$
$\langle \widehat{xy} \rangle \leftarrow \langle c \rangle \oplus_T \langle s \rangle$
return $\langle \widehat{xy} \rangle$

C.3 Additional Gadgets

Algorithm 28 SMul [ISW03, Theorem 1]

Input: The encodings $(x_1, \ldots, x_T)$ and $(y_1, \ldots, y_T)$.
Output: The encoding of xy as $(z_1, \ldots, z_T)$.

1: **for** $1 \leq i \leq T$
2: $\quad z_i \leftarrow x_i y_i$
3: **for** $1 \leq i \leq T$
4: $\quad$ **for** $i < j \leq T$
5: $\quad\quad r_{i,j} \leftarrow rand()$ // Not from the random tapes
6: $\quad\quad z_i \leftarrow z_i + r_{i,j}$ // Denoted by $z_{i,j}$
7: $\quad\quad r_{j,i} \leftarrow (x_i y_j - r_{i,j}) + x_j y_i$
8: $\quad\quad z_j \leftarrow z_j + r_{j,i}$ // Denoted by $z_{j,i}$
9: **return** $(z_1, \ldots, z_T)$

Algorithm 29 RefreshM [BBD+16]

Input: The encoding $(x_1, \ldots, x_T)$.
Output: The encoding $(y_1, \ldots, y_T)$ such that $y_1 + \ldots + y_T = x_1 + \ldots + x_T$

1: **for** $1 \leq i \leq T$
2: $\quad y_i \leftarrow x_i$
3: **for** $1 \leq i \leq T$
4: $\quad$ **for** $i < j \leq T$
5: $\quad\quad r_{i,j} \leftarrow rand()$ // Not from the random tapes
6: $\quad\quad y_i \leftarrow y_i + r_{i,j}$
7: $\quad\quad y_j \leftarrow y_j - r_{i,j}$
8: **return** $(y_1, \ldots, y_T)$

Algorithm 30 Unmask [BBE+19]

Input: The encoding $(x_1, \ldots, x_T)$.
Output: The shared value x such that $x = x_1 + \ldots + x_T$

1: $(x'_1, \ldots, x'_T) \leftarrow \mathsf{RefreshM}(x_1, \ldots, x_T)$
2: $x \leftarrow x'_1$
3: **for** $2 \leq i \leq T$
4: $\quad x \leftarrow x + x_i$
5: **return** x

C.4 Specification of Unprotected Picnic3

For completeness, we include the full specifications of Picnic3 signing adapted from [Pic20]. The notation is as defined elsewhere in the paper, and in the appendix on LowMc (Section C.5).

Algorithm 31 Sign

Input: Msg, pk and sk for each input wire w to the circuit.
Output: $(h, salt, Z)$

Sample random $R \in \{0,1\}^{2\kappa}$ // derive root and initial seeds
$(\mathsf{seed}^*, salt) \leftarrow \mathrm{KDF}(sk, Msg, pk, \lambda, R)$
$\mathsf{iSeed\_tree} \leftarrow \mathsf{gen\_seed}(\mathsf{seed}^*, salt, M, 0)$
$(\mathsf{iSeed}^{(1)}, \ldots, \mathsf{iSeed}^{(M)}) \leftarrow \mathsf{get\_leaves}(\mathsf{iSeed\_tree})$
for each $k \in [M]$:
    $\mathsf{seed\_tree}^{(k)} \leftarrow \mathsf{gen\_seed}(\mathsf{iSeed}^{(k)}, salt, N, k)$ // Derive random tapes from the initial seed
    $(\mathsf{seed}_1^{(k)}, \ldots, \mathsf{seed}_N^{(k)}) \leftarrow \mathsf{get\_leaves}(\mathsf{seed\_tree}^{(k)})$
    for each $i \in [N]$:
        $\mathsf{tape}_i^{(k)} \leftarrow \mathrm{KDF}(\mathsf{seed}_i^{(k)}, salt, k, i)$
    $\lambda^{sk} \leftarrow \mathsf{offline}(\mathsf{tape}_1^{(k)} || \ldots || \mathsf{tape}_N^{(k)}, pk)$ (see Algorithm 32)
    $\mathsf{aux}^{(k)} \leftarrow \mathsf{get\_aux}(\mathsf{tape}_N^{(k)})$
    for each $i \in [N]$:
        if $i \neq N$ **then**
            $\mathsf{com}_i^{(k)} \leftarrow \mathsf{H}(\mathsf{seed}_i^{(k)}, salt, k, i)$
        else
            $\mathsf{com}_i^{(k)} \leftarrow \mathsf{H}(\mathsf{seed}_i^{(k)}, \mathsf{aux}^{(k)}, salt, k, i)$
    $\mathsf{com\_off}^{(k)} \leftarrow \mathsf{H}(\mathsf{com}_1^{(k)}, \ldots, \mathsf{com}_N^{(k)})$ // Commit to preprocessing phase
    $\hat{sk}^{(k)} \leftarrow sk \oplus \lambda^{sk}$// Mask input bits
    $\mathsf{msgs}_1^{(k)}, \ldots, \mathsf{msgs}_N^{(k)} \leftarrow \mathsf{online}(\hat{sk}^{(k)}, \mathsf{tape}_1^{(k)} || \ldots || \mathsf{tape}_N^{(k)}, pk)$ (see Algorithm 35)
    $\mathsf{com\_on}^{(k)} \leftarrow \mathsf{H}(\hat{sk}^{(k)}, \mathsf{msgs}_1^{(k)}, \ldots, \mathsf{msgs}_N^{(k)})$ // Commit to MPC online phase
$\mathsf{com\_on\_tree} \leftarrow \mathsf{build\_tree}(\mathsf{com\_on}^{(1)}, \ldots, \mathsf{com\_on}^{(M)})$
$h \leftarrow \mathsf{H}(\mathsf{com\_off}^{(1)}, \ldots, \mathsf{com\_off}^{(M)}, \mathsf{com\_on\_tree.root}, salt, pk, Msg)$
Parse h as $(\mathcal{C}, \mathcal{P})$ where $\mathcal{C} \subset [M]$ and $\mathcal{P} = \{i_k\}_{k \in \mathcal{C}}$, $i_k \in [N]$
$\mathsf{com\_on\_info} \leftarrow \mathsf{open\_tree}(\mathsf{com\_on\_tree}, M, \mathcal{C})$ // Include only $\mathsf{com\_on}^{(k)}$ for $k \notin \mathcal{C}$
$\mathsf{iSeed\_info} \leftarrow \mathsf{reveal\_seed}(\mathsf{iSeed\_tree}, M, \mathcal{C})$ // Include only $\mathsf{iSeed}^{(k)}$ for $k \notin \mathcal{C}$
for each $k \in \mathcal{C}$: // Reveal online phases selected by challenge
    $\mathsf{seed\_info}^{(k)} \leftarrow \mathsf{reveal\_seed}(\mathsf{seed\_tree}^{(k)}, N, i_k)$ // Include only $\mathsf{seed}_i^{(k)}$ for $i \neq i_k$
    if $i_k = N$ then $\mathsf{aux}^{(k)} \leftarrow \perp$
let $Z = (\mathsf{com\_on\_info}, \mathsf{iSeed\_info}, (\mathsf{seed\_info}^{(k)}, \mathsf{aux}^{(k)}, \hat{sk}^{(k)}, \mathsf{com}_{i_k}^{(k)}, \mathsf{msgs}_{i_k}^{(k)})_{k \in \mathcal{C}})$.
output $(h, salt, Z)$ as a signature

Algorithm 32 offline

Input: The tapes (tapes). The signer's public key $pk = (c, p)$.
Output: The n-bit key mask λ^{sk}. The tapes is updated inside $\mathsf{sbox}_{\mathsf{aux}}$.

1: $\mathsf{roundkey}_0 \leftarrow \mathsf{tapes\_to\_parity}(n, \mathsf{tapes}, 0)$
2: $\lambda^{sk} \leftarrow \mathsf{matMul}(\mathsf{roundkey}_0, K_0^{-1})$
3: $\mathsf{st\_in} \leftarrow 0^n$
4: **for** each LowMc round i from r down to 1
5: $\quad \mathsf{roundkey}_i \leftarrow \mathsf{matMul}(\lambda^{sk}, K_i)$
6: $\quad \mathsf{st\_out} \leftarrow \mathsf{st\_in} \oplus \mathsf{roundkey}_i$
7: $\quad \mathsf{st\_out} \leftarrow \mathsf{matMul}(\mathsf{st\_out}, L_i^{-1})$
8: $\quad$ **if** $i = 1$ **then**
9: $\quad\quad \mathsf{st\_in} \leftarrow \mathsf{roundkey}_0$
10: $\quad$ **else**
11: $\quad\quad \mathsf{st\_in} \leftarrow \mathsf{tapes\_to\_parity}(n, \mathsf{tapes}, 2n(i-1))$
12: $\quad\quad \mathsf{offset} \leftarrow 2n(i-1) + n$
13: $\quad \mathsf{sbox}_{\mathsf{aux}}(\mathsf{st\_in}, \mathsf{st\_out}, \mathsf{tapes}, \mathsf{offset})$ (see Algorithm 33)
14: **return** λ^{sk}

Algorithm 33 $\mathsf{sbox}_{\mathsf{aux}}$

Input: The input and output states st_in and st_out. The tapes tapes. The tape offset offset.
Output: tapes is updated with the auxiliary bits.

1: **for** each i from 0 to $3s$, in steps of 3
2: $\quad \lambda^a \leftarrow \mathsf{st\_in}[i+2]$
3: $\quad \lambda^b \leftarrow \mathsf{st\_in}[i+1]$
4: $\quad \lambda^c \leftarrow \mathsf{st\_in}[i]$
5: $\quad \lambda^d \leftarrow \mathsf{st\_out}[i+2]$
6: $\quad \lambda^e \leftarrow \mathsf{st\_out}[i+1]$
7: $\quad \lambda^f \leftarrow \mathsf{st\_out}[i]$
8: $\quad \lambda^{z_{bc}} \leftarrow \lambda^d \oplus \lambda^a$
9: $\quad \lambda^{z_{ca}} \leftarrow \lambda^e \oplus \lambda^a \oplus \lambda^b$
10: $\quad \lambda^{z_{ab}} \leftarrow \lambda^f \oplus \lambda^a \oplus \lambda^b \oplus \lambda^c$
11: $\quad \mathsf{AND}_{\mathsf{aux}}(\lambda^b, \lambda^c, \lambda^{z_{bc}}, \mathsf{tape}_1, \ldots, \mathsf{tape}_N, \mathsf{offset} + i + 2)$
12: $\quad \mathsf{AND}_{\mathsf{aux}}(\lambda^c, \lambda^a, \lambda^{z_{ca}}, \mathsf{tape}_1, \ldots, \mathsf{tape}_N, \mathsf{offset} + i + 1)$
13: $\quad \mathsf{AND}_{\mathsf{aux}}(\lambda^a, \lambda^b, \lambda^{z_{ab}}, \mathsf{tape}_1, \ldots, \mathsf{tape}_N, \mathsf{offset} + i)$ (see Algorithm 34)

Algorithm 34 $\mathsf{AND}_{\mathsf{aux}}$

Input: The input mask bits λ^x and λ^y. The fresh output mask bit λ^z. The tapes tapes. The tape offset offset.
Output: The function updates tape_N.

1: $\mathsf{and\_helper}' \leftarrow \mathsf{tapes\_to\_parity}(1, \mathsf{tape}_1 || \ldots || \mathsf{tape}_{N-1}, \mathsf{offset})$
2: $\lambda^{xy} \leftarrow \lambda^x \wedge \lambda^y$
3: $\mathsf{aux\_bit} \leftarrow \lambda^{xy} \oplus \mathsf{and\_helper}' \oplus \lambda^z$
4: $\mathsf{tape}_N[\mathsf{offset}] \leftarrow \mathsf{aux\_bit}$ // Ensuring $\lambda^{xy} \oplus \lambda^z = \mathsf{tape}_1[\mathsf{offset}] \oplus \ldots \oplus \mathsf{tape}_N[\mathsf{offset}]$

Algorithm 35 online

Input: The masked input $\hat{sk}$. The tapes tapes. The signer's public key $pk = (c, p)$.
Output: The broadcast messages $\mathsf{msgs}_1, \ldots, \mathsf{msgs}_N$

$\mathsf{roundkey}_0 \leftarrow \mathsf{matMul}(\hat{sk}, K_0)$
$\mathsf{st} \leftarrow \mathsf{roundkey}_0 \oplus p$
Initialize empty arrays $\mathsf{msgs}_1, \ldots, \mathsf{msgs}_N$
for each LowMc round i from 1 to r
    $\mathsf{sbox}_{\mathsf{online}}(\mathsf{st}, \mathsf{tapes}, \mathsf{msgs}_1, \ldots, \mathsf{msgs}_N, 2n(i-1))$ (see Algorithm 36)
    $\mathsf{st} \leftarrow \mathsf{matMul}(\mathsf{st}, L_i)$
    $\mathsf{st} \leftarrow \mathsf{st} \oplus R_i$
    $\mathsf{roundkey}_i \leftarrow \mathsf{matMul}(\hat{sk}, K_i)$
    $\mathsf{st} \leftarrow \mathsf{st} \oplus \mathsf{roundkey}_i$
Compare st and c component of pk. If they differ, fail.
return $\mathsf{msgs}_1, \ldots, \mathsf{msgs}_N$

Algorithm 36 $\mathsf{sbox}_{\mathsf{online}}$

Input: The masked state st. The tapes tapes. The message holder $\mathsf{msgs}_1, \ldots, \mathsf{msgs}_N$. The tape offset offset.
Output: The function updates msgs.

for each i from 0 to $3s$, in steps of 3
    $\hat{a} \leftarrow \mathsf{st}[i+2]$
    $\hat{b} \leftarrow \mathsf{st}[i+1]$
    $\hat{c} \leftarrow \mathsf{st}[i]$
    $(\lambda_1^a, \ldots, \lambda_N^a) \leftarrow \mathsf{tapes\_to\_word}(\mathsf{tapes}, \mathsf{offset} + i + 2)$
    $(\lambda_1^b, \ldots, \lambda_N^b) \leftarrow \mathsf{tapes\_to\_word}(\mathsf{tapes}, \mathsf{offset} + i + 1)$
    $(\lambda_1^c, \ldots, \lambda_N^c) \leftarrow \mathsf{tapes\_to\_word}(\mathsf{tapes}, \mathsf{offset} + i)$
    $\widehat{bc} \leftarrow \mathsf{AND}_{\mathsf{online}}(\hat{b}, \hat{c}, (\lambda_i^b, \lambda_i^c, \mathsf{tape}_i, \mathsf{msgs}_i)_{i\in[N]}, \mathsf{offset} + n + i + 2)$
    $\widehat{ca} \leftarrow \mathsf{AND}_{\mathsf{online}}(\hat{c}, \hat{a}, (\lambda_i^c, \lambda_i^a, \mathsf{tape}_i, \mathsf{msgs}_i)_{i\in[N]}, \mathsf{offset} + n + i + 1)$
    $\widehat{ab} \leftarrow \mathsf{AND}_{\mathsf{online}}(\hat{a}, \hat{b}, (\lambda_i^a, \lambda_i^b, \mathsf{tape}_i, \mathsf{msgs}_i)_{i\in[N]}, \mathsf{offset} + n + i)$
    $\mathsf{st}[i+2] \leftarrow \hat{a} \oplus \widehat{bc}$
    $\mathsf{st}[i+1] \leftarrow \hat{a} \oplus \hat{b} \oplus \widehat{ca}$
    $\mathsf{st}[i] \leftarrow \hat{a} \oplus \hat{b} \oplus \hat{c} \oplus \widehat{ab}$

Algorithm 37 $\mathsf{AND}_{\mathsf{online}}$

Input: The masked input bits $\hat{x}$ and $\hat{y}$. The masking bits words $(\lambda_1^x, \ldots, \lambda_N^x)$ and $(\lambda_1^y, \ldots, \lambda_N^y)$. The tapes tapes. The message holder $\mathsf{msgs}_1, \ldots, \mathsf{msgs}_N$. The tape offset offset.
Output: Masked AND output $\widehat{xy}$. The function updates msgs.

// $\mathsf{and\_helper}_i$ contains party i's share of $\lambda^{xy} \oplus \lambda^z$
$(\mathsf{and\_helper}_1, \ldots, \mathsf{and\_helper}_N) \leftarrow \mathsf{tapes\_to\_word}(\mathsf{tapes}, \mathsf{offset})$
for each $i \in [N]$
    $s_i \leftarrow (\hat{x} \wedge \lambda_i^y) \oplus (\hat{y} \wedge \lambda_i^x) \oplus \mathsf{and\_helper}_i$
    Append s_i to msgs_i.
$\widehat{xy} \leftarrow \mathsf{parity}(s_1, \ldots, s_N) \oplus (\hat{x} \wedge \hat{y})$
return $\widehat{xy}$

C.5 LowMC

LowMC [ARS+15] is a parameterizable block cipher designed to have a small number of AND gates (low multiplicative complexity). In this work we assume the LowMc instances are those from the Picnic3 design; however, our analysis and countermeasures generalize easily to other choices of LowMc paramters.
Let n be the block size and key size, s be the number of S-boxes, and r the number of rounds. For each LowMc instance, the spec defines random and independent

- round constants $R_i \in \mathbb{F}_2^n$,
- linear layer matrices $L_i \in \mathbb{F}_2^{n \times n}$ (of full rank), and
- key matrices $K_i \in \mathbb{F}_2^{n \times n}$ for the computation of round keys.

There are r round and linear layer constants R_i, L_i, and $r+1$ key matrices K_i. The matrices are invertible, and the Picnic 3 implementation uses the inverses ${K_0}^{-1}$ and ${L_i}^{-1}$. LowMC keys are sampled uniformly at random from $\mathbb{F}_2^n$.
LowMC encryption starts by adding the first round key to the plaintext, which is followed by r rounds. Each round key is generated by multiplying the key with the key matrix K_i. A single round of LowMc is composed of an S-box layer, a linear layer, addition with constants, and addition of the round key as shown in Algorithm 38. The S-box layer applies the same 3-bit S-box on the first $3 \cdot s$ bits of the state. The S-box is defined as $S(a,b,c) = (a \oplus bc, a \oplus b \oplus ac, a \oplus b \oplus c \oplus ab)$. The other layers only consist of $\mathbb{F}_2$-vector space arithmetic, all local operations in our MPC setting.

Algorithm 38 LowMC encryption. Parameters K_i, L_i and R_i are as described in the text.

Input: Key matrices $KM_{i\in[1,r]} \in \mathbb{F}_2^{n\times k}$, Linear matrices $LM_{i\in[1,r]} \in \mathbb{F}_2^{n\times n}$, Round constants $RC_{i\in[1,r]} \in \mathbb{F}_2^n$, a plaintext $p \in \mathbb{F}_2^n$ and a secret-key $k_s \in \mathbb{F}_2^n$
Output: Ciphertext $s \in \mathbb{F}_2^n$ such that $s = \mathsf{LowMC}(p, k_s)$.

1: $s \leftarrow (KM_0 \cdot k_s) \oplus p$ // Initial Key Addition
2: **for** $1 \le i \le r$
3: $\quad s \leftarrow Sbox(s)$ // `SboxLayer`
4: $\quad s \leftarrow LM_i \cdot s$ // `LinearLayer`
5: $\quad s \leftarrow RC_i \oplus s$ // `ConstantAddition`
6: $\quad s \leftarrow (KM_i \cdot k_s) \oplus s$ // `KeyAddition`
7: **return** s

Bibliography

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. 2020 (accessed August 6, 2020).

[ABB+17] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. Bike: bit flipping key encapsulation. 2017.

[ABD+19] Erdem Alkim, Joppe W Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, Douglas Stebila, et al. Frodokem learning with errors key encapsulation. 2019.

[ABE+21] Diego F. Aranha, Sebastian Berndt, Thomas Eisenbarth, Okan Seker, Akira Takahashi, Luca Wilke, and Greg Zaverucha. Side-channel protections for picnic signatures. Cryptology ePrint Archive, Report 2021/735, 2021. https://eprint.iacr.org/2021/735.

[ABGV] Ali C Atıcı, Lejla Batina, Benedikt Gierlichs, and Ingrid Verbauwhede. Power analysis on ntru implementations for rfids: First results.

[ACC+17] Reza Azarderakhsh, M Campagna, C Costello, L Feo, B Hess, A Jalali, D Jao, B Koziel, B LaMacchia, P Longa, et al. Sike–supersingular isogeny key encapsulation. *URL: https://sike. org*, 2017.

[AKJ+18] Soojung An, Suhri Kim, Sunghyun Jin, HanBit Kim, and HeeSeok Kim. Single trace side channel analysis on ntru implementation. *Applied Sciences*, 8(11):2014, 2018.

[AOTZ20] Diego F. Aranha, Claudio Orlandi, Akira Takahashi, and Greg Zaverucha. Security of hedged Fiat-Shamir signatures under fault attacks. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 644–674. Springer, Heidelberg, May 2020.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

[ARU14] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. Quantum attacks on classical proof systems: The hardness of quantum rewinding. In *FOCS*, pages 474–483. IEEE Computer Society, 2014.

[AVFM07] Frédéric Amiel, Karine Villegas, Benoit Feix, and Louis Marcel. Passive and active combined attacks: Combining fault attacks and side channel analysis. In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fourth International Workshop on Fault Diagnosis and Tolerance in Cryptography, 2007, FDTC 2007: Vienna, Austria, 10 September 2007*, pages 92–102. IEEE Computer Society, 2007.

[AWMN20] Victor Arribas, Felix Wegener, Amir Moradi, and Svetla Nikova. Cryptographic fault diagnosis using verfi. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*, pages 229–240. IEEE, 2020.

[BBB+19] Estuardo Alpirez Bock, Joppe W Bos, Chris Brzuska, Charles Hubain, Wil Michiels, Cristofaro Mune, Eloi Sanfelix Gonzalez, Philippe Teuwen, and Alexander Treff. White-box cryptography: don't forget about grey-box attacks. *Journal of Cryptology*, 32(4):1095–1143, 2019.

[BBC+19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskverif: Automated verification of higher-order masking in presence of physical defaults. In *ESORICS (1)*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.

[BBD+15a] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, and Benjamin Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. *IACR Cryptol. ePrint Arch.*, 2015:506, 2015.

[BBD+15b] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors,

EUROCRYPT 2015, Part I, volume 9056 of *LNCS*, pages 457–485. Springer, Heidelberg, April 2015.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *ACM Conference on Computer and Communications Security*, pages 116–129. ACM, 2016.

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Heidelberg, April / May 2018.

[BBE+19] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2147–2164. ACM Press, November 2019.

[BBF+19] Estuardo Alpirez Bock, Chris Brzuska, Marc Fischlin, Christian Janson, and Wil Michiels. Security reductions for white-box key-storage in mobile payments. Cryptology ePrint Archive, Report 2019/1014, 2019. https://eprint.iacr.org/2019/1014.

[BBIJ17] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, and Martin Bjerregaard Jepsen. Analysis of Software Countermeasures for Whitebox Encryption. *IACR Trans. Symmetric Cryptol.*, 2017(1):307–328, 2017.

[BBMT18] Estuardo Alpirez Bock, Chris Brzuska, Wil Michiels, and Alexander Treff. On the Ineffectiveness of Internal Encodings-Revisiting the DCA Attack on White-Box Cryptography. In *International Conference on Applied Cryptography and Network Security*, pages 103–120. Springer, 2018.

[BCD06] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. *IACR Cryptology ePrint Archive*, 2006:468, 2006.

[BCLVV16] Daniel J Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine Van Vredendaal. Ntru prime. *IACR Cryptol. ePrint Arch.*, 2016:461, 2016.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BDF+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. *Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model*, pages 535–566. Springer International Publishing, Cham, 2017.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In Walter Fumy, editor, *Advances in Cryptology EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[BDPA10] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Building power analysis resistant implementations of keccak. Second SHA-3 Candidate Conference, 2010. https://keccak.team/files/KeccakDPA.pdf.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[BECN+06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[Ben88] Ben-Or, Michael and Goldwasser, Shafi and Wigderson, Avi. Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[Ber68] E.R. Berlekamp. *Algebraic coding theory.* McGraw-Hill series in systems science. McGraw-Hill, 1968.

[BG13] Alberto Battistello and Christophe Giraud. Fault analysis of infective AES computations. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, pages 101–107. IEEE, 2013.

[BG16] Alberto Battistello and Christophe Giraud. A Note on the Security of CHES 2014 Symmetric Infective Countermeasure. In *Constructive Side-Channel Analysis and Secure Design – COSADE 2016*, 2016.

[BGEC05] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, pages 227–240. Springer, 2005.

[BGI+18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal verification of masked hardware implementations in the presence of glitches. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking kyber: First- and higher-order implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):173–214, 2021.

[BHK+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The sphincs$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2129–2146. ACM, 2019.

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 215–236. Springer, 2016.

[BMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*, page 991–1008, August 2018.

[BN20] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 495–526. Springer, Heidelberg, May 2020.

[BOM+19] Jo Van Bulck, David F. Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1741–1758. ACM, 2019.

[BP09] Joan Boyar and Rene Peralta. New logic minimization techniques with applications to cryptology. Cryptology ePrint Archive, Report 2009/191, 2009.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.*, 10(2):163–188, 2020.

[BRVW19] Andrey Bogdanov, Matthieu Rivain, Philip S Vejre, and Junwei Wang. Higher-order DCA against standard side-channel countermeasures. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 118–141. Springer, 2019.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 513–525. Springer, 1997.

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and Countermeasures for White-box Designs. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 373–402, Cham, 2018. Springer International Publishing.

[CB08] D. Canright and Lejla Batina. *A Very Compact "Perfectly Masked" S-Box for AES*, pages 446–459. Springer, 2008.

[CBR+15] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the AES s-box. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[CCU+20] Tung Chou, Carlos Cid, Simula UiB, Jan Gilcher, Tanja Lange, Varun Maram, Rafael Misoczki, Ruben Niederhagen, Kenneth G Paterson, Edoardo Persichetti, et al. Classic mceliece: conservative code-based cryptography 10 october 2020. 2020.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM Conference on Computer and Communications Security*, pages 1825–1842. ACM, 2017.

[CDH+17] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. Ntru algorithm specifcations and supporting documentation. *URL: https://ntru.org/*, 2017.

[CEJvO03a] Stanley Chow, Philip Eisen, Harold Johnson, and Paul C. van Oorschot. A White-Box DES Implementation for DRM Applications. In Joan Feigenbaum, editor, *Digital Rights Management*, pages 1–15. Springer, 2003.

[CEJVO03b] Stanley Chow, Philip Eisen, Harold Johnson, and Paul C. Van Oorschot. White-Box Cryptography and an AES Implementation. In Kaisa Nyberg and Howard Heys, editors, *Selected Areas in Cryptography*, pages 250–270. Springer, 2003.

[CEvMS15] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Differential power analysis of a mceliece cryptosystem. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security*, pages 538–556, Cham, 2015. Springer International Publishing.

[CEvMS16] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Masking large keys in hardware: A masked implementation of mceliece. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, pages 293–309, Cham, 2016. Springer International Publishing.

[CFGR10] C. Clavier, B. Feix, G. Gagnerot, and M. Roussellet. Passive and Active Combined Attacks on AES Combining Fault Attacks and Side Channel Analysis. In *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 10–19, Aug 2010.

[CFMR+17] Antoine Casanova, Jean-Charles Faugere, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. *GeMSS: a great*

multivariate short signature. PhD thesis, UPMC-Paris 6 Sorbonne Universités; INRIA Paris Research Centre, MAMBA Team …, 2017.

[CGG+19] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking data on meltdown-resistant cpus. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 769–784. ACM, 2019.

[CGPZ16] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of sboxes via common shares. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 498–514. Springer, 2016.

[CGZ19] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel Masking with Pseudo-Random Generator. Cryptology ePrint Archive, Report 2019/1106, 2019. https://eprint.iacr.org/2019/1106.

[Cho16] Tung Chou. Qcbits: Constant-time small-key code-based cryptography. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 280–300. Springer, 2016.

[CJRR99a] Suresh Chari, Charanjit Jutla, Josyula R Rao, and Pankaj Rohatgi. A cautionary note regarding evaluation of AES candidates on smart-cards. In *Second Advanced Encryption Standard Candidate Conference*, pages 133–147. Citeseer, 1999.

[CJRR99b] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CKN01] Jean-Sébastien Coron, Paul Kocher, and David Naccache. Statistics and Secret Leakage. In Yair Frankel, editor, *Financial Cryptography*, pages 157–173, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[CPR13] Jean-Sébastien Coron, Emmanuel Prouff, and Thomas Roche. On the Use of Shamir's Secret Sharing against Side-Channel Analysis. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications*, pages 77–90. Springer, 2013.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic Decomposition for Probing Security. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 742–763, 2015.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[CRZ13] Guilhem Castagnos, Soline Renner, and Gilles Zémor. *High-order Masking by Using Coding Theory and Its Application to AES*, pages 193–212. Springer, 2013.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.

[CT05] Hamid Choukri and Michael Tunstall. Round reduction using faults. *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 5:13–24, 2005.

[Dae17] Joan Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 137–153. Springer, Heidelberg, September 2017.

[Dam10] Ivan Damgård. On Σ-protocols. http://www.cs.au.dk/~ivan/Sigma.pdf, 2010.

[DCN16] Thomas De Cnudde and Svetla Nikova. More Efficient Private Circuits II Through Threshold Implementations. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography 2016*. IEEE, 2016.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. *Unifying Leakage Models: From Probing Attacks to Noisy Leakage.*, pages 423–440. Springer, 2014.

[DDF19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. *Journal of Cryptology*, 32(1):151–177, January 2019.

[dDOS19] Cyprien de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: Using AES in picnic signatures. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 669–692. Springer, Heidelberg, August 2019.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. Sifa: Exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):547–572, Aug. 2018.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the fiat-shamir transformation in the quantum random-oracle model. In *CRYPTO (2)*, volume 11693 of *Lecture Notes in Computer Science*, pages 356–383. Springer, 2019.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology* -

AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings, volume 10831 of *Lecture Notes in Computer Science*, pages 282–305. Springer, 2018.

[DMN+12] J. M. Dutertre, A. P. Mirbaha, D. Naccache, A. L. Ribotta, A. Tria, and T. Vaschalde. Fault Round Modification Analysis of the advanced encryption standard. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 140–145, June 2012.

[DMWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a Perturbated White-Box AES Implementation. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010*, pages 292–310. Springer, 2010.

[DN19] Itai Dinur and Niv Nadler. Multi-target attacks on the Picnic signature scheme and related protocols. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 699–727. Springer, Heidelberg, May 2019.

[DN20] Siemen Dhooghe and Svetla Nikova. My gadget just cares for me - how NINA can prove security against combined attacks. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 35–55. Springer, 2020.

[DOTT21] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 99–130. Springer, Heidelberg, May 2021.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, volume 3531 of *Lecture Notes in Computer Science*, pages 164–175, 2005.

[Dwo15] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 621–641. Springer, Heidelberg, March 2013.

[DZD+18] A. Adam Ding, Liwei Zhang, Francois Durvaux, Francois-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications*, pages 105–122, Cham, 2018. Springer International Publishing.

[EmP] LF-U 2.5, H-Field Probe 100 kHz-50 MHz . https://www.langer-emv.de/en/product/lf-passive-100-khz-50-mhz/36/lf-u-2-5-h-field-probe-100-khz-up-to-50-mhz/5.

[FA17] Hayato Fujii and Diego F. Aranha. Curve25519 for the cortex-M4 and beyond. In Tanja Lange and Orr Dunkelman, editors, *LATINCRYPT 2017*, volume 11368 of *LNCS*, pages 109–127. Springer, Heidelberg, September 2017.

[FDK20] Apostolos P. Fournaris, Charis Dimopoulos, and Odysseas Koufopavlou. Profiling dilithium digital signature traces for correlation differential side channel attacks. In Alex Orailoglu, Matthias Jung, and Marc Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 281–294, Cham, 2020. Springer International Publishing.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[FHK+18] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 36, 2018.

[FMPR11] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. *Affine Masking against Higher-Order Side Channel Analysis*, pages 262–280. Springer, 2011.

[fre] FRDM-K66F: Freedom Development Platform for Kinetis. https://www.nxp.com/downloads/en/schematics/FRDM-K66F-SCH.pdf.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[Gem] Gemalto. Sentinel® LDK product brief. https://sentinel.gemalto.com/resources/software/sentinel-ldk-feature-brief/.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj M. Prabhakaran, Amit Sahai, and Eran Tromer. Circuits Resilient to Additive Attacks with Applications to Secure Computation. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 495–504, New York, NY, USA, 2014. ACM.

[Gir06] Christophe Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *Computers, IEEE Transactions on*, 55(9):1116–1120, 2006.

[GM11] Louis Goubin and Ange Martinelli. Protecting AES with Shamir's secret sharing scheme. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 79–94. Springer, 2011.

[GMK16] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, page 3, New York, NY, USA, 2016. Association for Computing Machinery.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems CHES 2001*, pages 251–261. Springer, 2001.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, pages 1069–1083. USENIX Association, 2016.

[Gol07] Oded Goldreich. *Foundations of cryptography: volume 1, basic tools*. Cambridge university press, 2007.

[GPRW19] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *Journal of Cryptographic Engineering*, Apr 2019.

[GR16] Dahmun Goudarzi and Matthieu Rivain. How Fast Can Higher-Order Masking Be in Software? Cryptology ePrint Archive, Report 2016/264, 2016.

[GR19] François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qtesla. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2019.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219. ACM, 1996.

[GRR98] Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM, 1998.

[GRW20] Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):454–482, 2020.

[GSDM+19] Hannes Gross, Ko Stoffelen, Lauren De Meyer, Martin Krenn, and Stefan Mangard. First-order masking with only two random bits. In *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*, TIS'19, page 10–23, New York, NY, USA, 2019. Association for Computing Machinery.

[GSE21] Tim Gellersen, Okan Seker, and Thomas Eisenbarth. Differential power analysis of the picnic signature scheme. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2021.

[GSF14] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: How large is the gap for AES? *Journal of Cryptographic Engineering*, 4(1):47–57, 2014.

[GSM17] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In *DSD*, pages 205–212. IEEE Computer Society, 2017.

[GST12] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In *Progress in Cryptology–LATINCRYPT 2012*, pages 305–321. Springer, 2012.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *Advances in Cryptology–CRYPTO 2014*, pages 444–461. Springer, 2014.

[GT03] Jovan D. Golić and Christophe Tymen. Multiplicative masking and power analysis of aes. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 198–212, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[HCY19] Wei-Lun Huang, Jiun-Peng Chen, and Bo-Yin Yang. Correlation power analysis on ntru prime and related countermeasures. *IACR Cryptol. ePrint Arch.*, 2019:100, 2019.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384.

[HMP10] Stefan Heyse, Amir Moradi, and Christof Paar. Practical power analysis attacks on software implementations of mceliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography*, pages 108–125, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust Pseudorandom Generators. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, pages 576–588, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 308–327. Springer, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

[KA21] Emre Karabulut and Aydin Aysu. Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks. *IACR Cryptol. ePrint Arch.*, page 772, 2021.

[Kar10] Mohamed Karroumi. Protecting White-Box AES with Dual Ciphers. In Kyung-Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology - ICISC 2010*, pages 278–291. Springer, 2010.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, 1883.

[KGB+18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. Differential power analysis of XMSS and SPHINCS. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018*, volume 10815 of *LNCS*, pages 168–188. Springer, Heidelberg, April 2018.

[KGM+20] Thilo Krachenfels, Fatemeh Ganji, Amir Moradi, Shahin Tajik, and Jean-Pierre Seifert. Real-world snapshots vs. theory: Questioning the t-probing security model, 2020. To appear at IEEE S&P 2021.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19. IEEE, 2019.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[KKT04] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Differential fault analysis attack resistant architectures for the advanced encryption standard. In Jean-Jacques Quisquater, Pierre Paradinas, Yves Deswarte, and Anas Abou El Kalam, editors, *Smart Card Research and Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France*, volume 153 of *IFIP*, pages 177–192. Kluwer/Springer, 2004.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

[KMMS21] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated generation of masked hardware. *IACR Cryptol. ePrint Arch.*, page 569, 2021.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR TCHES*, 2020(3):243–268, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8590.

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

[KSV13] Duško Karaklajić, Jörn-Marc Schmidt, and Ingrid Verbauwhede. Hardware designer's guide to fault attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2295–2306, 2013.

[KW02] Ramesh Karri and Kaijie Wu. Algorithm level re-computing using implementation diversity: a register transfer level concurrent error detection technique. *IEEE Trans. Very Large Scale Integr. Syst.*, 10(6):864–875, 2002.

[KZ20] Daniel Kales and Greg Zaverucha. Improving the performance of the Picnic signature scheme. *IACR TCHES*, 2020(4):154–188, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8680.

[LFZD14] Pei Luo, Yunsi Fei, Liwei Zhang, and A Adam Ding. Side-channel power analysis of different protection schemes against fault attacks on AES. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[LKK18] S. Lee, T. Kim, and Y. Kang. A Masked White-Box Cryptographic Implementation for Protecting Against Differential Computation Analysis. *IEEE Transactions on Information Forensics and Security*, 13(10):2602–2615, 2018.

[LN05] Hamilton E. Link and William D. Neumann. Clarifying obfuscation: improving the security of white-box DES. *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, 1:679–684 Vol. 1, 2005.

[LRDM+14] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two Attacks on a White-Box AES Implementation. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, pages 265–285. Springer, 2014.

[LRT12] Victor Lomné, Thomas Roche, and Adrian Thillard. On the Need of Randomness in Fault Attack Countermeasures-Application to AES. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on*, pages 85–94. IEEE, 2012.

[LSCH10] Mun-Kyu Lee, Jeong Song, Dooho Choi, and Dong-Guk Han. Countermeasures against power analysis attacks for the ntru public key cryptosystem. *IEICE Transactions*, 93-A:153–163, 01 2010.

[LSG+10] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 320–334, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 973–990. USENIX Association, 2018.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting post-quantum fiat-shamir. In *CRYPTO (2)*, volume 11693 of *Lecture Notes in Computer Science*, pages 326–355. Springer, 2019.

[MAB+18] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2:4–13, 2018.

[MAN+19] Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. M&m: Masks and macs against physical attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):25–50, 2019.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa,

and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 344–362. Springer, Heidelberg, June 2019.

[MIE17] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. CacheZoom: How SGX Amplifies the Power of Cache Attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 69–90. Springer, 2017.

[MM13] Amir Moradi and Oliver Mischke. On the simplicity of converting leakages from multivariate to univariate. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 1–20. Springer, 2013.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. *Correlation-Enhanced Power Analysis Collision Attack*, pages 125–139. Springer, 2010.

[MMSS18] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited - or why proofs in the robust probing model are needed. Cryptology ePrint Archive, Report 2018/490, 2018.

[MOO+14] Ciara Moore, Máire O'Neill, Elizabeth O'Sullivan, Yarkın Doröz, and Berk Sunar. Practical homomorphic encryption: A survey. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2792–2795. IEEE, 2014.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, Berlin, Heidelberg, 2007.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. *Successfully Attacking Masked AES Hardware Implementations*, pages 157–171. Springer, 2005.

[MQ18] Axel Mathieu-Mahias and Michaël Quisquater. Mixing additive and multiplicative masking for probing secure polynomial evaluation methods. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):175–208, 2018.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):209–237, 2018.

[MS] R. J. McEliece and D. V. Sarwate. on sharing secrets and reed-solomon codes.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[MSEH20] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. TPM-FAIL: TPM meets timing and lattice attacks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2057–2073. USENIX Association, 2020.

[Muk09] Debdeep Mukhopadhyay. An improved fault based attack of the advanced encryption standard. In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, pages 421–434, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. DL-LA: deep learning leakage assessment A modern roadmap for SCA evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):552–598, 2021.

[Nat20] National Institute of Standards and Technology. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*, 2020. Available at https://doi.org/10.6028/NIST.IR.8309.

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure saber KEM implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):676–707, 2021.

[Nik20] Ventzi Nikov. Threshold implementations against physical attacks an industry view, 2020. Talk at Online Workshop on Threshold Schemes for NIST-approved Symmetric Block Ciphers in a Single-Device Setting.

[NIS20a] NIST. *Post-Quantum Cryptography – Round 1 Submissions*, 2017 (accessed April 25, 2020).

[NIS20b] NIST. *Post-Quantum Cryptography – Round 2 Submissions*, 2019 (accessed April 25, 2020).

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

[NRS09] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In *Information Security and Cryptology–ICISC 2008*, pages 218–234. Springer, 2009.

[OMPR05] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-channel Analysis Resistant Description of the AES S-box. In *Proceedings of the 12th International Conference on Fast Software Encryption*, FSE'05, pages 413–423. Springer-Verlag, 2005.

[OSPG16] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical cca2-secure and masked ring-lwe implementation. Cryptology ePrint Archive, Report 2016/1109, 2016. https://eprint.iacr.org/2016/1109.

[Pat97] Jacques Patarin. The oil and vinegar signature scheme. In *Dagstuhl Workshop on Cryptography September, 1997*, 1997.

[Pay] EMV-Mobile Payment. Software-based Mobile Payment Security Requirements. https://www.emvco.com/terms-of-use/?u=wp-content/uploads/documents/EMVCo-SBMP-16-G01-V1.4_SBMP_Security_Requirements.pdf.

[Pic20] The Picnic Design Team. *The Picnic Signature Algorithm Specification*, April 2020. Version 3.0, Available at https://microsoft.github.io/Picnic/.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

[PSKH18] Aesun Park, Kyung-Ah Shim, Namhun Koo, and Dong-Guk Han. Side-channel attacks on post-quantum signature schemes based on multivariate

quadratic equations: - rainbow and uov -. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):500–523, Aug. 2018.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Advances in Cryptology–CRYPTO 2015*, pages 764–783. Springer LNCS, 2015.

[REB+08] Francesco Regazzoni, Thomas Eisenbarth, Luca Breveglieri, Paolo Ienne, and Israel Koren. Can knowledge regarding the presence of countermeasures against fault attacks simplify power attacks on cryptographic devices? In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pages 202–210. IEEE, 2008.

[RHHM17] Melissa Rossi, Mike Hamburg, Michael Hutter, and Mark E. Marson. A side-channel assisted cryptanalytic attack against qcbits. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2017.

[RJH+18] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. Side-channel assisted existential forgery attack on dilithium - a nist pqc candidate. Cryptology ePrint Archive, Report 2018/821, 2018. https://eprint.iacr.org/2018/821.

[RLK11] Thomas Roche, Victor Lomné, and Karim Khalfallah. Combined Fault and Side-Channel Attack on Protected Implementations of AES. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications*, volume 7079 of *Lecture Notes in Computer Science*, pages 65–83. Springer, 2011.

[RMB+17] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel Smart. CAPA: The Spirit of Beaver against Physical Attacks. Cryptology ePrint Archive, Report 2017/1195, 2017.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[RP11] Thomas Roche and Emmanuel Prouff. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 63–78. Springer, 2011.

[RP12] Thomas Roche and Emmanuel Prouff. Higher-order glitch free implementation of the AES using secure multi-party computation protocols. *Journal of Cryptographic Engineering*, 2(2):111–127, 2012.

[RSA83] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems (reprint). *Commun. ACM*, 26(1):96–99, 1983.

[RSS+21] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. FIVER - robust verification of countermeasures against fault injections. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):447–473, 2021.

[Rub17] Abraham Fernandez Rubio. Efficient side-channel resistant mpc-based software implementation of the aes. Master's thesis, Worcester Polytechnic Institute, 2017.

[RW19] Matthieu Rivain and Junwei Wang. Analysis and Improvement of Differential Computation Attacks against Internally-Encoded White-Box Implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):225–255, Feb. 2019.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[SBWE20] Okan Seker, Sebastian Berndt, Luca Wilke, and Thomas Eisenbarth. SNI-in-the-head: Protecting MPC-in-the-head protocols against side-channel analysis. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 1033–1049. ACM Press, November 2020.

[Seb] Sebastian Ramacher and Daniel Kales and Greg Zaverucha and Christian Paquin. Picnic signature scheme optimized implementation. https://github.com/IAIK/Picnic.

[SEL21] Okan Seker, Thomas Eisenbarth, and Maciej Liskiewicz. A white-box masking scheme resisting computational and algebraic attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):61–105, Feb. 2021.

[SFRES18] Okan Seker, Abraham Fernandez-Rubio, Thomas Eisenbarth, and Rainer Steinwandt. Extending glitch-free multiparty protocols to resist fault injection attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):394–430, Aug. 2018.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4):656–715, 1949.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sha99] Adi Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, November 23 1999. US Patent 5,991,415.

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[SKC+19] Bo-Yeon Sim, Jihoon Kwon, Kyu Young Choi, Jihoon Cho, Aesun Park, and Dong-Guk Han. Novel side-channel attacks on quasi-cyclic code-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(4):180–212, Aug. 2019.

[SKL+20] B. Y. Sim, J. Kwon, J. Lee, I. J. Kim, T. H. Lee, J. Han, H. Yoon, J. Cho, and D. G. Han. Single-trace attacks on message encoding in lattice-based kems. *IEEE Access*, 8:183175–183191, 2020.

[SLM+19] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Zombieload: Cross-privilege-boundary data sampling. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM*

SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019, pages 753–768. ACM, 2019.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations, 2015.

[SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI – Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 302–332. Springer, 2016.

[SRSWZ20] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the cca2-secure hqc kem. Cryptology ePrint Archive, Report 2020/910, 2020. https://eprint.iacr.org/2020/910.

[Ste] Steven Goldfeder and Greg Zaverucha. Picnic signature scheme reference implementation. https://github.com/microsoft/Picnic.

[TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying Fault Invariant with Randomization. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2014.

[TG16] Michael Tunstall and Gilbert Goodwill. Applying TVLA to public key cryptographic algorithms. Cryptology ePrint Archive, Report 2016/513, 2016. https://eprint.iacr.org/2016/513.

[Tol37] John Ronald Reuel Tolkien. *The hobbit, or there and back again*. George Allen & Unwin Ltd., 1937.

[TWC17] Edition 1 The WhibOx Contest. CHES 2017 Capture the Flag Challenge The WhibOx Contest. https://whibox.io/contests/2017/, 2017.

[TWC19] Edition 2 The WhibOx Contest. CHES 2019 Capture the Flag Challenge The WhibOx Contest. https://whibox.io/contests/2017/, 2019.

[TWC21] Edition 3 The WhibOx Contest. CHES 2021 Challenge - WhibOx contest. https://whibox.io/contests/2021/, 2021.

[Unr15] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *EUROCRYPT*, volume 9057 of *Lecture Notes in Computer Science*, pages 755–784. Springer, 2015.

[VBDK+20] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel resistant implementation of saber. *IACR Cryptol. ePrint Arch*, 733:2020, 2020.

[VBPS17] Jo Van Bulck, Frank Piessens, and Raoul Strackx. SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, SysTEX'17, New York, NY, USA, 2017. Association for Computing Machinery.

[Ver19] Kasper Verhulst. Power analysis and masking of saber. 2019.

[VWA+21] B. Viguier, D. Wong, G. Van Assche, Q. Dang, and J. Daemen. KangarooTwelve. IRTF Crypto Forum Internet-Draft, Feb 2021. https://tools.ietf.org/html/draft-irtf-cfrg-kangarootwelve-05.

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *Selected Areas in Cryptography*, pages 264–277. Springer, 2007.

[WWME20] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. Sevurity: No security without integrity : Breaking integrity-free memory encryption with minimal assumptions. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1483–1496. IEEE, 2020.

[WWSE21] Luca Wilke, Jan Wichelmann, Florian Sieck, and Thomas Eisenbarth. undeserved trust: Exploiting permutation-agnostic remote attestation. In *IEEE Security and Privacy Workshops, SP Workshops 2021, San Francisco, CA, USA, May 27, 2021*, pages 456–466. IEEE, 2021.

[WZW13] An Wang, Xuexin Zheng, and Zongyue Wang. Power analysis attacks and countermeasures on ntru-based wireless body area networks. *KSII Transactions on Internet and Information Systems*, 7:1094–1107, 05 2013.

[XL09] Y. Xiao and X. Lai. A Secure Implementation of White-Box AES. In *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6, 2009.

[Yi18] Haibo Yi. Under quantum computer attack: Is rainbow a replacement of rsa and elliptic curves on hardware?, Feb 2018.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.

[YL17] Haibo Yi and Weijian Li. On the importance of checking multivariate public key cryptography for side-channel attacks: The case of entts scheme. *The Computer Journal*, 60:1–13, 02 2017.

[YO13] Maki Yoshida and Satoshi Obana. Detection of Cheaters in Non-interactive Polynomial Evaluation. Cryptology ePrint Archive, Report 2013/032, 2013.

[ZCD+20] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. Picnic. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[ZYD+20] F. Zhang, B. Yang, X. Dong, S. Guilley, Z. Liu, W. He, F. Zhang, and K. Ren. Side-channel analysis and countermeasure design on arm-based quantum-resistant sike. *IEEE Transactions on Computers*, 69(11):1681–1693, 2020.