FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Side-Channel Security of Embedded Devices

by

Ing. Petr Socha

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Doctoral study programme: Informatics
Department of Digital Design

Prague, May 2023

Supervisor:
Dr.-Ing. Martin Novotný
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Co-Supervisor:
Ing. Vojtěch Miškovský, Ph.D.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract and contributions

This dissertation thesis focuses on the physical security of cryptographic implementations in embedded devices, especially resistance to side-channel attacks such as differential power analysis. Attention is paid to both symmetric encryption algorithms (PRESENT, AES, Serpent) and the post-quantum asymmetric Rainbow signature scheme. Security of symmetric cryptography is mainly investigated on a Field Programmable Gate Arrays (FPGA) hardware platform, where countermeasures are proposed for design and synthesis on register-transfer and system levels. Asymmetric cryptography is mainly investigated on software platforms, where an attack on the reference Rainbow implementation is discussed and a multi-platform countermeasure against such attacks is proposed. All of the proposed countermeasures are evaluated in terms of time and space requirements and their effectiveness is evaluated using leakage assessment methodology.

The main contributions of the dissertation thesis are based on peer-reviewed and published papers, here presented as follows:

1. Survey of the current state of the art in the field of side-channel analysis.
2. Design, implementation, and evaluation of countermeasures, based on a dynamic reconfiguration of combinational logic, for AES and Serpent on FPGA, and designed for register transfer level synthesis.
3. Design, implementation, and evaluation of countermeasures for PRESENT, AES, and Serpent on FPGA, described at system level and designed for high-level synthesis.
4. Design, implementation, and evaluation of an attack on the 32-bit reference implementation of Rainbow on an ARM-based microcontroller.
5. Proposal of a countermeasure for multivariate quadratic signature schemes based on the principle of equivalent keys, its formal examination, implementation for Rainbow, and evaluation on an ARM-based microcontroller.

Keywords:
Embedded Systems, Cryptography, Side-Channel Security, Internet of Things.

Abstrakt a přínosy (Czech)

Tato dizertační práce se zabývá fyzickou bezpečností kryptografických implementací ve vestavných zařízeních, konkrétně odolností proti útokům postranními kanály, mezi něž patří například rozdílová odběrová analýza. Pozornost je věnována jak symetrickým šifrovacím algoritmům (PRESENT, AES, Serpent), tak asymetrickému podpisovému schéma Rainbow. Bezpečnost symetrické kryptografie je zkoumána především na hardwarové platformě založené na programovatelném hradlovém poli (FPGA), kde jsou navržena protiopatření jak pro návrh na úrovni meziregistrových přenosů, tak na systémové úrovni. Asymetrická kryptografie je zkoumána především na softwarové platformě, kde je navržen útok na referenční implementaci Rainbow a následně je navrženo multiplatformní protiopatření proti takovým útokům. Všechna navržená protiopatření jsou vyhodnocena z hlediska časové a prostorové náročnosti a jejich účinnost je ověřena pomocí vhodné metodologie.

Hlavní přínosy této dizertační práce vycházejí z recenzovaných a publikovaných článků, zde prezentovaných následovně:

1. Shrnutí současného stavu poznání v oboru analýzy postranních kanálů.
2. Návrh, implementace a vyhodnocení protiopatření, založených na dynamické rekonfiguraci kombinační logiky, pro AES a Serpent na FPGA, určených pro syntézu na úrovni meziregistrových přenosů.
3. Návrh, implementace a vyhodnocení protiopatření pro PRESENT, AES a Serpent na FPGA, popsaných na systémové úrovni a určených pro vysokoúrovňovou syntézu.
4. Návrh, implementace a vyhodnocení útoku na 32bitovou referenční implementaci Rainbow na mikrokontroléru s jádrem ARM.
5. Návrh protiopatření pro multivarietní kvadratická podpisová schémata založeného na principu ekvivalentních klíčů, jeho formální analýza a implementace pro Rainbow a vyhodnocení na mikrokontroléru s jádrem ARM.

Klíčová slova:
Vestavné systémy, kryptografie, bezpečnost postranních kanálů, internet věcí.

Acknowledgement

First of all, I would like to thank my supervisor Martin Novotný for his help during my studies, his valuable insights, for involving me in many different projects, and for introducing me to many interesting people. I would also like to thank my co-supervisor Vojtěch Miškovský for his insights and cooperation, and for convincing me to do my research in the first place. I would also like to thank my student and colleague David Pokorný for a productive collaboration. Many thanks to my colleague Jan Onderka for helpful discussions, proofreading of my work, and for providing a sober perspective. Many thanks also to Hana Kubátová and the entire Department of Digital Design for providing a friendly and helpful environment for my research.

A big thank you goes to my grandfather Josef, who taught me to build my own electrical circuits and program my first 386 PC when I was just a little boy, thus predetermining my whole life. Special thanks to my music teacher Karel Bareš, who taught me calmness, perseverance, and prudence. I must not forget my friends, Martin, Honza, Zdeněk, Petr, Milan, Filip, Marek, and others, who had a great influence on me at one time or another in my life, and who gave me the strength to carry on. Finally, my biggest thanks go to my parents Jana and Petr for their unwavering support in everything I do.

The research presented in this thesis has been partially supported by the following grants:

- "Fault-Tolerant and Attack-Resistant Architectures Based on Programmable Devices: Research of Interplay and Common Features" (GA16-05179S) of The Czech Science Foundation (2016-2018).

- "DRASTIC: Dynamically Reconfigurable Architectures for Side-channel analysis protecTIon of Cryptographic implementations" of the Central Europe Leuven Strategic Alliance (CELSA) (2017-2019).

- "Tools for AI-enhanced Security Verification of Cryptographic Devices" (VJ02010010) of the Ministry of the Interior of the Czech Republic (2022-2025).

- "Dependable and attack-resistant architectures for programmable devices" (2017-2019, SGS17/213/OHK3/3T/18), "Design, programming and verification of embedded systems" (2020-2022, SGS20/211/OHK3/3T/18), "Design, programming and verification of intelligent embedded systems" (2023-2025, SGS23/208/OHK3/3T/18) of the Student Grant Competition of CTU in Prague.

Dedication

To all my loves, past, present, and to come.

Contents

List of Figures

List of Tables

Abbreviations

Related to Cryptography and Side-Channel Analysis

AES	Advanced Encryption Standard
CPA	Correlation Power Analysis
DDLA	Differential Deep Learning Analysis
DLLA	Deep Learning Leakage Assessment
DM	Distinguishing Margin
DPA	Differential Power Analysis
GE	Guessing Entropy
HD	Hamming Distance
HW	Hamming Weight
ID	Identity
iKSA	Inter-class Kolmogorov-Smirnov Analysis
KSA	Kolmogorov-Smirnov Analysis
MIA	Mutual Information Analysis
PPA	Partitioning Power Analysis
RSA	Rivest–Shamir–Adleman cryptosystem
SR	Success Rate
TA	Template Attack

Related to Digital Design and Electronics

AC	Alternating Current
ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
CLK	Clock
CMOS	Complementary Metal–Oxide–Semiconductor
DC	Direct Current
EM	Electromagnetic
FPGA	Field-Programmable Gate Array
GND	Ground
HLS	High-Level Synthesis
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
NMOS	N-channel Metal–Oxide–Semiconductor
PMOS	P-channel Metal–Oxide–Semiconductor
RAM	Random Access Memory
RTL	Register-Transfer Level
SABL	Sense Amplifier-Based Logic
SDDL	Simple Dynamic Differential Logic
SNR	Signal-to-Noise Ratio
VHDL	VHSIC (Very High-Speed Integrated Circuits Program) Hardware Description Language
WDDL	Wave Dynamic Differential Logic
XOR	Exclusive OR

Miscellaneous

CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
GPU	Graphics Processing Unit
IoT	Internet of Things
ML	Machine Learning
MLP	Multilayer Perceptron
NIST	National Institute of Standards and Technology
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
SVM	Support Vector Machine

Mathematical notation and the used formal model is presented in subsection 2.1.2.

Chapter 1

Introduction

1.1 Motivation and Problem Statement

In the past few decades, computer systems and communication networks have become an essential part of our everyday lives. Various computing devices are used not only as tools for many professionals, but also for entertainment. These devices include embedded devices, such as payment cards, biometric passports, smart cars, trains or whole cities, and even medical devices like pacemakers. Being surrounded by devices connected to the Internet, our private lives are endangered more than ever [156].

Special attention must be given to ensure security of computer systems and their users. Various measures are employed to achieve confidentiality, integrity, availability, and non-repudiation of data with efficiency, ease of use, and cost in mind. Nowadays, widely used algorithms such as AES [45, 120] or RSA [144] are typically considered secure from the cryptanalytic point of view. However, their implementations may leak sensitive information through side channels of the used cryptographic device, potentially compromising the entire system.

Side-channel attacks exploit the data-dependent side channels, such as power consumption of the cryptographic device [84, 83] or its electromagnetic radiation [136], in order to extract secret information such as cipher keys. Such attacks pose a severe threat to both hardware and software cryptographic implementations, especially in the IoT environment where the attacker may easily gain physical access to the device, leaving it vulnerable to tampering. Various countermeasures have been proposed to prevent such attacks. Masking is a widely used technique based on randomization of the processed data [39, 109, 122, 63], making it difficult to exploit the leakage. Another common approach is hiding, which aims to conceal the exploitable leakage in either side-channel signal amplitude or time [167, 168, 64, 108]. Recent real-world attack examples show that uncompromising protection and testing of embedded cryptographic implementations is necessary [95].

1.2 Goals of the Dissertation Thesis

Together with my supervisors, we aimed to contribute to effective, efficient, and easy-to-use countermeasures against passive non-invasive side-channel attacks. Our goals can be summarized as follows:

1. Development of effective, efficient (in the terms of area and time), and easy-to-use side-channel countermeasures for symmetric cryptography (most notably Rijndael/AES) tailored specifically for FPGA hardware platforms.
2. Side-channel analysis of the Rainbow signature scheme and development of effective and efficient side-channel countermeasures for such multivariate quadratic signature schemes.

1.3 Structure of the Dissertation Thesis

This doctoral dissertation thesis is organized into 5 chapters as follows:

- *Chapter 1: Introduction*: Describes the motivation behind our efforts together with our goals.
- *Chapter 2: Background and State of the Art*: Introduces the reader to the necessary theoretical background and surveys the current state of the art.
- *Chapter 3: Symmetric Cryptography*: Describes our countermeasure development for PRESENT, Rijndael/AES, and Serpent on FPGA. Countermeasures for both register-transfer level and high level synthesis are proposed and thoroughly evaluated.
- *Chapter 4: Asymmetric Cryptography*: Describes our side-channel attack on the Rainbow signature scheme reference implementation and our development of a multi-platform countermeasure based on a concept of equivalent keys, suitable for multivariate-quadratic signature schemes.
- *Chapter 5: Conclusion*: Summarizes the results of our research, suggests possible topics for further research, and concludes the thesis.

Chapter 2

Background and State of the Art

This chapter was published as a review article in the Sensors journal [A.1] in 2022.

This chapter presents theoretical background and state of the art in the area of non-invasive passive side-channel attacks. It is structured into 7 sections as follows:

1. *Introduction to Side-Channel Security*: Introduces side-channel leakage origin, measurement setup, formal model of the leakage, and leakage functions.
2. *Non-Profiled Attacks*: Describes non-invasive passive non-profiled attacks.
3. *Profiled Attacks*: Describes non-invasive passive profiled attacks.
4. *Side-Channel Related Metrics*: Describes both experimental and theoretical attack-related metrics.
5. *Countermeasures Against Attacks*: Describes hiding and masking countermeasures.
6. *Attacks on Protected Implementations*: Describes extensions of the presented attacks for attacking implementations with countermeasures.
7. *Leakage Assessment*: Describes methods for evaluations of side-channel leakage.

2.1 Introduction to Side-Channel Security

Side channels of digital systems that may be used to compromise the system include power consumption [83, 49, 29, 40], electromagnetic radiation [136], combinational logic delay [150, 183], timing [84], and more. Some of these side channels are mutually dependent, for example, the relationship between current intensity and the magnetic field can be shown, e.g., by Biot–Savart law [160], and the combinational logic delay can be convincingly modeled as inversely proportional to the voltage drop [128]. This dissertation

thesis focuses on dynamic power consumption side channel, but presented concepts may be relevant for other side channels as well.

Side-channel attacks may be classified in many different ways [160], such as invasive/non-invasive or active/passive. Invasive attacks require depackaging the chip in order to access internal components, such as data buses, while non-invasive attacks only exploit the external access. Active attacks tamper with proper functionality of the device (e.g., by introducing faults), while passive attacks only make use of observation of the device during its undisturbed operation. This dissertation thesis focuses on non-invasive passive attacks only.

Side-channel attacks can also be classified as either horizontal or vertical. Horizontal attacks exploit leakage during a single algorithm execution, while vertical attacks exploit leakage from multiple executions. For example, considering a hardware implementation of RSA algorithm that uses naïve Square&Multiply exponentiation, either only Square is performed, or both Square&Multiply operations are performed during computation, for every exponent bit, depending on the bit being zero or one. This not only influences execution time, but, in some cases, it also allows the attacker to directly read the secret key from a single measured power/EM trace by graphing the trace, as the two operations form distinctive patterns [84]. This kind of a horizontal side-channel attack is called Simple Power Analysis. Unlike this simple example, this dissertation thesis focuses on vertical side-channel attacks, where the information is typically contained in the instantaneous signal amplitude as further described below.

A CMOS inverter model is depicted in Figure 2.1. Three different dissipation sources can be observed in such a CMOS structure [137]:

- static leakage current,
- short-circuit current, and
- capacitance charge and discharge.

When the inverter input presents a stable voltage corresponding to 0 or 1, one of the transistors is open and the other one is closed. In this case, only static leakage current is present. When the input changes, short-circuit current can be observed for a brief period of time when both transistors are open. Furthermore, the modeled load capacitance C_L has to be charged to the proper voltage when the input changes its value. Therefore, based on the instantaneous current consumption, it can be easily distinguished whether a transition happened or not. This fact is exploited by the most common leakage models as described later in this section. The consumption during a transition is demonstrated in Figure 2.2.

Because the P-channel MOSFET majority carriers have lower mobility and the minority carriers have lower lifetime, in contrast to the N-channel MOSFET [71, 57], the P-channel MOSFET is typically built larger than the N-channel MOSFET [71], resulting in different characteristics, most importantly on-resistance and propagation delay (for non-inverter gates) [138]. Due to differences between N-channel and P-channel MOSFETs, the output value after transition can also be distinguished by the instantaneous current [100].

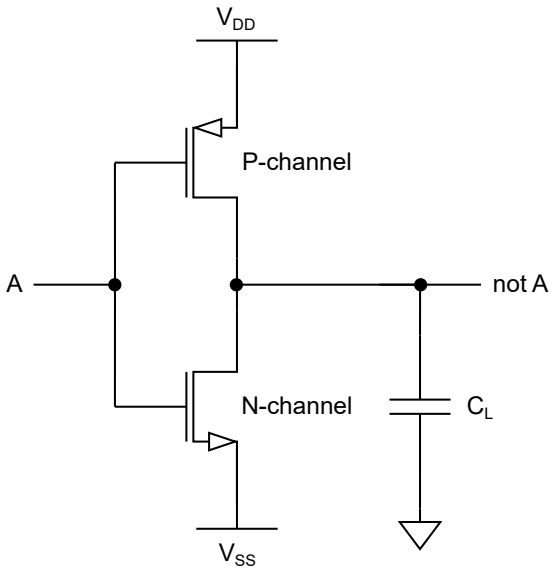


Figure 2.1: A CMOS inverter model.

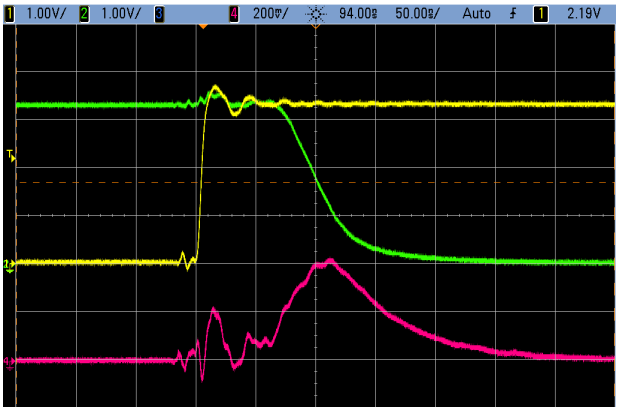


Figure 2.2: A CMOS inverter current consumption. The yellow line is the inverter input, the green line is the inverter output. The pink line is the current consumption, where peaks during the transition are clearly observable.

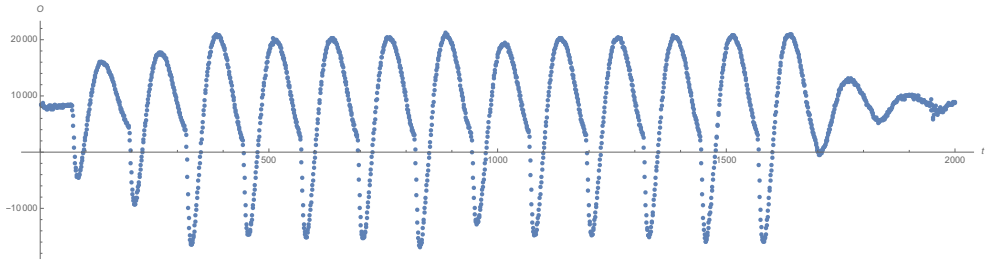


Figure 2.3: Rijndael/AES Encryption FPGA Power Trace.

This simple example illustrates data dependency of the instantaneous power consumption, which is the main cause of the power-related side-channel information leakage in CMOS-based integrated circuits. Vertical attacks exploiting this kind of leakage typically require multiple side-channel measurements, unlike the previously described simple power analysis attack.

2.1.1 Measurements

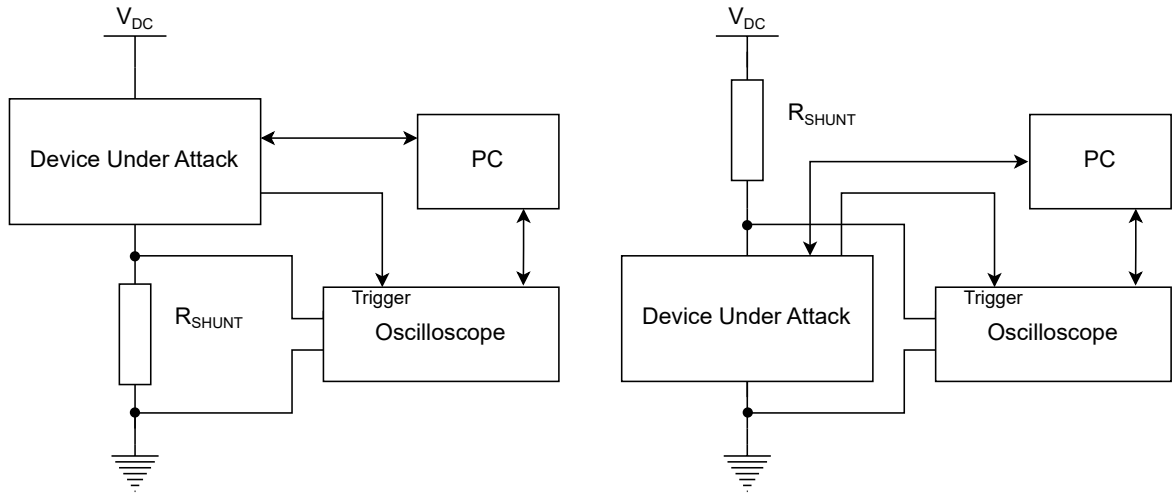
The device's side channel is typically observed during a cryptographic operation, resulting in a measurement record, so-called trace, i.e., a vector of samples. For example, a single trace of dynamic power consumption during Rijndael/AES encryption in an FPGA is visualized in Figure 2.3. As mentioned earlier, multiple aligned traces, such as this one, are typically required for a successful attack, although single-trace attacks are sometimes also possible. This subsection briefly describes different measurement methods.

2.1.1.1 Power Consumption

Power consumption of the cryptographic device is typically measured using an oscilloscope which samples voltage across a shunt resistor. The current can then be obtained, knowing both resistance and voltage, using Ohm's law $I = \frac{U}{R}$. However, raw ADC values corresponding to the voltage can be directly used in a typical attack scenario, since the current and the voltage are assumed to be linearly dependent, as long as the oscilloscope setup parameters are consistent during all measurements.

Various measurement setups are described in [115], the differences being primarily in the shunt resistor placement:

- **Shunt resistor in GND path**, with the voltage across the resistor being sampled by the oscilloscope, as shown in Figure 2.4a,
- **Shunt resistor in V_{DC} path**, with the voltage across the cryptographic device being sampled by the oscilloscope, as shown in Figure 2.4b, also observing voltage drops of the power regulator.



(a) GND path measurement (b) V_{DC} path measurement

Figure 2.4: Example of a power measurement setup.

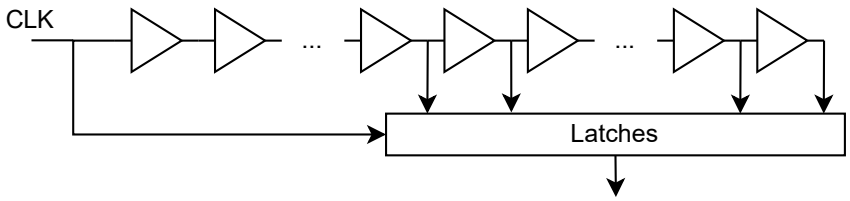
The latter setup offers an advantage when the device under attack has multiple power networks, because it allows the attacker to measure the just cryptographic core consumption. When the voltage is measured using a shunt resistor in the ground path (Figure 2.4a), the measured voltage typically contains more noise such as noise caused by the device's peripheral drivers. When measuring in the V_{DC} path (Figure 2.4b), the DC shift must be removed (unless measuring in a differential mode), which can be ensured either by using oscilloscope's AC mode, or by using external DC blocker. Other choices for power measurement include differential or current probes. However, these are not recommended unless necessary, as they present an additional source of environmental noise [115].

In a real-world attack, any decoupling capacitors near the cryptographic core must be removed as they might filter out the relevant voltage changes. Correct power measurement setup is crucial for successful side-channel analysis. Parameters such as the environmental noise, sampling rate, or synchronization jitter have a direct impact on the attack success [127].

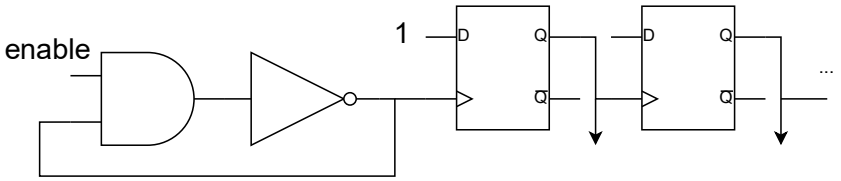
2.1.1.2 Electromagnetic Radiation

Similarly to power measurement, an oscilloscope connected to a near-field probe may be used for measurement of electromagnetic radiation [136]. As mentioned earlier, there is a close relationship between the power consumption and the radiation [160].

Attacking electromagnetic radiation offers more degrees of freedom compared to the power analysis. The attacker can examine a particular part of the chip only, and she can choose from a wide variety of probes. Consequently, EM analysis may provide a very powerful tool at the cost of more intricate and more costly employment. Further discussion of EM side channels is outside of the scope of this dissertation thesis.



(a) Delay chain + latches.



(b) Ring oscillator + asynchronous counter.

Figure 2.5: Combinatorial logic delay monitors.

In addition to directly measuring electromagnetic radiation of the device under attack, data-dependent leakage may also be unintentionally broadcast by a radio transmitter present on the same chip (such as SoC bluetooth/WiFi transmitters with built-in encryption) [36]. In mixed-signal systems on chip, the leakage from the digital part of the chip couples through substrate to the high-frequency analog radio transmitter [2]. This class of attacks is called screaming channels and it allows the attacker to successfully reveal cipher keys from traces obtained by a radio receiver from even 15 m distance [35].

2.1.1.3 Combinational Logic Delay

Combinational logic delay inside a chip can be satisfactorily modeled as inversely proportional to the voltage drop of the internal power network [128], which is data-dependent due to the switching activity. In FPGA chips, the delay can be measured internally using a delay-chain monitor [61, 150], shown in Figure 2.5a, or using a ring oscillator monitor [183], shown in Figure 2.5b. Acquired delay traces can be used in side-channel analysis in a similar fashion as the power traces.

Furthermore, crosstalk between two long wires inside the chip can be detected [140] using a ring oscillator monitor as a receiver. In a multitenant FPGA chip setting, where independent customers share the same FPGA accelerator, e.g., in a cloud environment, these monitors open possibility for remote and even automated large-scale side-channel attacks.

2.1.2 Formal Model

This subsection presents a formal model of side-channel leakage and corresponding terminology as described in [58, 163]. The presented model is used for attack descriptions in

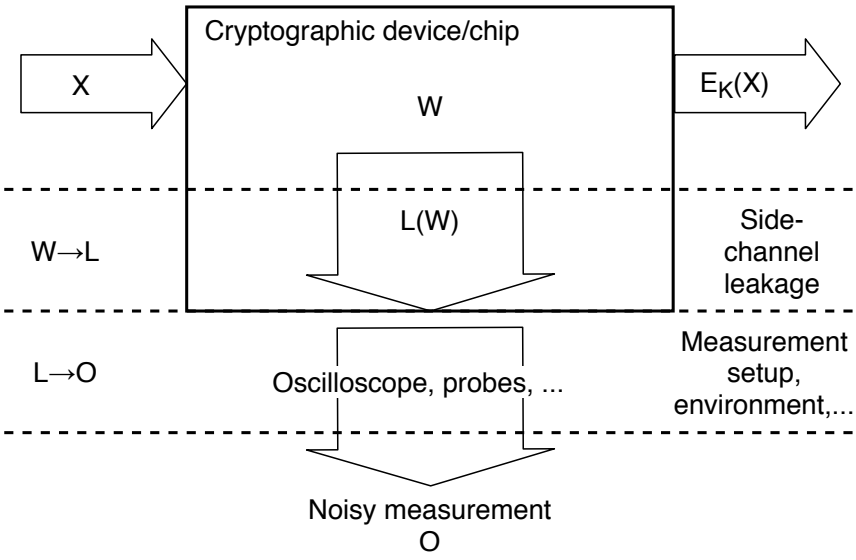


Figure 2.6: Illustration of channels involved in side-channel analysis.

following sections.

Consider a physical device performing a cryptographic operation $\mathrm{E}_k(x)$, depending on a secret (sub)key $k \in \mathcal{K}$, where $\mathcal{K} = \mathbb{B}^m = \{0,1\}^m$, $x \in \mathcal{X}$. The unknown (sub)key is then modeled as a random variable $\mathbf{K} : \Omega \to \mathcal{K}$, the processed data as a random variable $\mathbf{X} : \Omega \to \mathcal{X}$.

Key-dependent state transitions (bit flips) occur inside the device during the execution of E_k. These state transitions are described as word pairs $(v_1, v_2) \in \mathcal{W}$, where $\mathcal{W} = \mathbb{B}^n \times \mathbb{B}^n$, v_1 is previous state, v_2 is new state. Unknown transitions (word pairs) are modeled as a random variable $\mathbf{W} : \Omega \to \mathcal{W}$.

State transitions $\mathbf{W}$ induce side-channel leakage $\mathbf{L}$ on space $\mathcal{L}$, modeled by a side-channel leakage function $\mathsf{L}(\mathbf{W})$. Leakage $\mathbf{L}$ is measured through the noisy physical observable $\mathbf{O}$ on space $\mathcal{O}$.

The model describes a cascade of two channels $\mathbf{W} \to \mathbf{L} \to \mathbf{O}$. This cascade is comprised of a leakage channel $\mathbf{W} \to \mathbf{L}$ through which information on processed words $\mathbf{W}$ leaks in $\mathbf{L}$, and observation channel $\mathbf{L} \to \mathbf{O}$ through which the attacker obtains noisy information on $\mathbf{L}$. The described channels are illustrated in Figure 2.6.

Observing $\mathbf{O}$ then means measuring $q \in \mathbb{N}^+$ traces $o_{x_i}(t)$, $i = 1, 2, ..., q$, of device's side channel (e.g., power consumption) $\mathbf{O}(t)$, while processing known data x_i. In case of Rijndael/AES, o_{x_i} might be a trace similar to the one in Figure 2.3 and x_i might be a corresponding plaintext or ciphertext block.

Side-channel attack is then defined as determining the (sub)key k by reconstructing the words $\mathbf{W}$ and using information on $\mathbf{L}$ contained in $\mathbf{O}$. For example, the attack may be performed in these steps:

1. The real leakage function L is unknown, so the attacker assumes a hypothetical

leakage function $\hat{\mathsf{L}}$ (described in subsection 2.1.3).

2. The attacker makes a guess $\hat{k} \in \mathcal{K}$ on the real (sub)key k.

3. Based on the known data $\mathbf{X}$, she computes an intermediate value $f_{\hat{k}}(\mathbf{X})$ within the $\mathrm{E_k}$ computation.

4. The intermediate value implies a guess $\mathbf{W}_{\hat{k}}$, which in turn implies a guess $\hat{\mathbf{L}}_{\hat{k}} = \hat{\mathsf{L}}(\mathbf{W}_{\hat{k}})$.

5. Finally, the attacker checks if the guess $\hat{\mathbf{L}}_{\hat{k}}$ is compatible with the observed $\mathbf{O}$.

This attack scenario assumes that the real (sub)key k is fully enumerable in a reasonable time and space. As shown in section 2.2 and section 2.3, side-channel attacks typically target a single subkey, e.g., an octet in case of Rijndael/AES.

2.1.3 Leakage Function

Side-channel attacks can be classified into two groups according to approach to the hypothetical leakage function $\hat{\mathsf{L}}$:

- **Non-profiled attacks**, where the attacker only makes use of an explicit leakage function, which is effective for a range of devices (e.g., based on CMOS technology) instead of being tailored for a specific one.

- **Profiled attacks**, which consist of a profiling step, where the attacker examines a duplicate of the device under attack and she creates her own leakage approximation. Furthermore, her approximation inherently takes noise contained in $\mathbf{O}$ into account, making her empirical model more effective. This model is used for the attack, while an explicit leakage function may or may not be used during the process.

Besides the differences regarding approach to the hypothetical leakage function, these two types of attacks also assume a differently powerful attacker: for a profiled attack, an exact duplicate of the device under attack is required, whereas it is not for a non-profiled attack.

This subsection briefly introduces widely used explicit leakage functions necessary for non-profiled attacks which are discussed thereafter in section 2.2. Profiled attacks are discussed later in section 2.3.

2.1.3.1 Hamming distance and Hamming weight

Hamming distance leakage function for $v_1, v_2 \in \mathbb{B}^n$ is defined as a number of bit positions at which the words v_1, v_2 differ [29]:

$$\hat{\mathsf{L}}^{\mathrm{HD}}(v_1, v_2) = \mathrm{HD}(v_1, v_2) \in \mathcal{L} = \{0, 1, ..., n\}, \tag{2.1}$$

The function corresponds to the number of bit flips in an n-bit wide register during (v_1, v_2) transition. It is a generally applicable model suitable for attacking CMOS logic. Hamming distance is equal to Hamming weight of XOR of the operands: $\mathrm{HD}(v_1, v_2) = \mathrm{HW}(v_1 \oplus v_2)$, where Hamming weight HW is defined as a number of bits in the word that are set to one.

When v_1 is a zero vector, the Hamming distance leakage function reduces to Hamming weight leakage function [111]

$$\hat{\mathrm{L}}^{\mathrm{HW}}(v_2) = \mathrm{HW}(v_2) \in \mathcal{L} = \{0, 1, ..., n\}, \tag{2.2}$$

for $v_2 \in \mathbb{B}^n$. This is often the case when attacking software implementations in microcontrollers [83, 125].

Assuming Hamming weight $\mathrm{HW}(x), x \in \mathbb{B}^n$ and uniformly distributed values of n bits in a word $x \in \mathbb{B}^n$, following properties hold for Hamming weight (and consequently Hamming distance):

$$\mathrm{HW}(x) \sim \mathrm{Binom}(n, \frac{1}{2}), \tag{2.3}$$

$$\mathbb{E}(\mathrm{HW}(x)) = \frac{n}{2},\ \mathrm{Var}(\mathrm{HW}(x)) = \frac{n}{4}. \tag{2.4}$$

Furthermore, the Binomial distribution can be satisfactorily approximated by the Normal distribution for $p = \frac{1}{2}$ [77] and therefore

$$\mathrm{HW}(x) \approx \mathcal{N}(\frac{n}{2}, \sqrt{\frac{n}{4}}). \tag{2.5}$$

Generalized distance model Less commonly, different weights may be assigned for $0 \to 1$ and $1 \to 0$ transitions, resulting in a generalized distance leakage function. E.g., weight 1.5 may be used instead of 1 for the $1 \to 0$ transition to provide a more effective attack on some platforms [96].

2.1.3.2 Identity

Identity leakage function [58]

$$\hat{\mathrm{L}}^{\mathrm{id}}(f_{\hat{k}}(x_i)) = f_{\hat{k}}(x_i) \in \mathcal{L}, \tag{2.6}$$

for $x_i \in \mathcal{X}$, is equal to the targeted intermediate value within the $\mathrm{E_k}(x_i)$ computation. It is the most general leakage function in the sense that it puts no assumptions on the cryptographic device or technology.

2.2 Non-Profiled Attacks

Non-profiled attacks can be divided into:

- **Parametric/Moment-based attacks**, which exploit statistical moments (such as mean or variance). Typical examples include Differential Power Analysis [83] or Correlation Power Analysis [49, 29].
- **Non-parametric/Information-theoretic attacks**, which exploit the entire underlying statistical distribution. A typical example is Mutual Information Analysis [58].
- **Machine learning-based attacks**, namely the Deep Learning Power Analysis [166].

These attacks are presented in more depth in this section.

Unless stated otherwise, all of the attack descriptions in this section assume the attacker has already acquired $q \in \mathbb{N}^+$ traces $o_{x_i}(t)$, $i = 1, 2, ..., q$, of device's side channel $\mathbf{O}(t)$ (e.g., power consumption), while processing known data x_i (e.g., plaintext), where bits in x_i are uniformly distributed. Usage of uniform plaintext gives a good confidence about uniformity of intermediate values during the computation, since a cipher where properties such as diffusion are expected is typically targeted.

The q measured traces can be modeled as q samples from a multivariate random variable $\mathbf{O}(t)$, where the dimension of the variable corresponds to a number of sampling points within single trace. All the attacks presented in this section, except for the last one, are univariate, i.e., only a single point in time is examined, which is desirable when the sensitive intermediate value manifests itself at a single time instant. In this section, unless stated otherwise, it is assumed that the interesting time instant $t = \tau$ is known and only the single relevant sampling point is considered. The q measured traces are therefore considered an univariate random variable $\mathbf{O}(\tau)$.

In practice, when the time instant is unknown, the attack is performed at every time instant t independently. The final attack evaluation thus typically requires more attention and skill due to a larger false-result chance. Alignment of the traces is required when synchronization of the measurements (e.g., using a trigger signal) is not possible.

2.2.1 Differential Power Analysis (DPA)

The Differential Power Analysis [83] attack is performed in these steps:

1. Assume a single bit ($n = 1$) Hamming weight (or distance) leakage function $\hat{\mathsf{L}}$.
2. Enumerate (sub)key guesses $\hat{k} \in \mathcal{K}$.
3. Compute an intermediate value $v_2 = f_{\hat{k}}(x_i), \forall \hat{k}, x_i$ (and the previous state v_1 if Hamming distance is used) and consider only a single bit (e.g., the LSB).
4. For every guess $\hat{k}$, partition measurements o_{x_i} into two groups $O_0^{\hat{k}}, O_1^{\hat{k}}$ according to the leakage function $\hat{\mathsf{L}}$:

$$O_0^{\hat{k}} = \{o_{x_i} \mid \hat{\mathsf{L}}(v_1, f_{\hat{k}}(x_i)) = 0\}, \tag{2.7}$$

$$O_1^{\hat{k}} = \{o_{x_i} \mid \hat{\mathsf{L}}(v_1, f_{\hat{k}}(x_i)) = 1\}. \tag{2.8}$$

5. Select the guess $\hat{k}$ for which the groups' $O_0^{\hat{k}}, O_1^{\hat{k}}$ means differ the most.
   - For wrong guesses $\hat{k}$, the traces for which $\mathsf{L} = 0$ and the traces for which $\mathsf{L} = 1$ are theoretically uniformly distributed in both groups.
   - For the right guess $\hat{k}$, the groups $O_0^{\hat{k}}, O_1^{\hat{k}}$ should be distinguishable by their mean value, due to the bias caused by the fixed bit.

In the last step, the original Kocher's DPA [83] selects the guess $\hat{k}$ for which the absolute difference of means between the two groups is greatest. More formally, the hypothesis about equal means may be examined by using Welch's t-test or a similar statistic.

Example: Attacking first round of Rijndael/AES

1. Assume Hamming weight or identity single-bit leakage (Hamming weight is equivalent to identity for $n = 1$).
2. Select an enumerable key-dependent intermediate value during Rijndael/AES encryption: $f_{\hat{k}}(x_i) := \mathrm{Sbox}(x_i \oplus \hat{k})$. Sbox is an 8-bit bijection, $f_{\hat{k}}(x_i)$ is therefore computable for each byte independently of the other bytes.
3. Enumerate byte subkey guesses $\hat{k} \in \mathcal{K} = \{0, 1, \ldots, 255\}$, compute the intermediate value $v_2 = f_{\hat{k}}(x_i), \forall \hat{k}, x_i$ and choose, e.g., the LSB.
4. Use the Hamming weight of the LSB as leakage function and partition traces o_{x_i} to groups $O_0^{\hat{k}}, O_1^{\hat{k}}$.
5. Select the subkey guess $\hat{k}$ for which the two groups differ the most.

Because DPA builds its hypothesis on a single bit value, its assumptions regarding the leakage are very general. This may be one of the reasons for false results, so-called "ghost peaks". The choice of the bit in $f_{\hat{k}}(x_i)$ has direct impact on the attack success. These facts are a motivation for multi-bit DPA as described further [20, 37].

2.2.2 Multi-bit DPA and Partitioning Power Analysis (PPA)

Bevan's approach to multi-bit extension of the DPA is performing the original DPA independently for different bits of intermediate value $f_{\hat{k}}(x_i)$ and summing all the independent differences of means [20]. Then the subkey guess with the greatest summed difference is selected. This approach reduces the number of traces necessary as well as the chance of a false result [20].

Messerges's multi-bit extension of the original DPA suggests using n-bit Hamming weight or Hamming distance leakage model [111], therefore utilizing the whole $f_{\hat{k}}(x_i)$ value. This time, two sets $O_<^{\hat{k}}$, $O_\geq^{\hat{k}}$ are defined so that

$$O_<^{\hat{k}} = \{o_{x_i} \mid \hat{\mathsf{L}}(f_{\hat{k}}(x_i)) < \frac{n}{2}\}, \tag{2.9}$$

$$O_{\geq}^{\hat{k}} = \{o_{x_i} \,|\, \hat{\mathsf{L}}(f_{\hat{k}}(x_i)) \geq \frac{n}{2}\}, \tag{2.10}$$

and their difference is examined similarly to the original DPA.

Partitioning Power Analysis [4, 90] is a generalization of the multi-bit DPA. Assuming an n-bit $f_{\hat{k}}(x_i)$ intermediate value and a Hamming weight or Hamming distance leakage function, the traces o_{x_i} are partitioned into $(n+1)$ sets $O_0^{\hat{k}}, \ldots O_n^{\hat{k}}$ so that

$$O_j^{\hat{k}} = \{o_{x_i} \,|\, \hat{\mathsf{L}}(f_{\hat{k}}(x_i)) = j\}. \tag{2.11}$$

The distinguishing statistic (which is a difference of means in the original DPA) is then defined by using weights $a_j \in \mathbb{R}$ as

$$\sum_{j=0}^{n} a_j \cdot \mu_{O_j^{\hat{k}}}, \tag{2.12}$$

where $\mu_{O_j^{\hat{k}}}$ are means of the aforementioned groups.

The original DPA is a special case of 1-bit PPA where $a_0 = -1, a_1 = 1$. Bevan's 4-bit DPA is a special case of 4-bit PPA where $a_0 = -\frac{1}{8}, a_1 = -\frac{1}{4}, a_2 = 0, a_3 = \frac{1}{4}, a_4 = \frac{1}{8}$. Messerges's n-bit DPA is a special case of n-bit PPA where $a_j = -1$ for $0 \leq j < \frac{n}{2}$, and $a_j = 1$ for $\frac{n}{2} \leq j \leq n$ [90].

2.2.3 Correlation Power Analysis (CPA)

The Correlation Power Analysis [49, 29] attack is performed in these steps:

1. Assume a Hamming weight or Hamming distance leakage function $\hat{\mathsf{L}}$.
2. Enumerate (sub)key guesses $\hat{k} \in \mathcal{K}$.
3. Compute an intermediate value $v_2 = f_{\hat{k}}(x_i), \forall \hat{k}, x_i$ (and the previous state v_1 if using Hamming distance).
4. For every key guess $\hat{k}$, pairs $(o_{x_i}, \hat{\mathsf{L}}(v_1, f_{\hat{k}}(x_i)))$ represent samples from joint distribution $(\mathbf{O}, \hat{\mathbf{L}}_{\hat{k}})$. [1]
5. Compute Pearson correlation coefficient $\rho_{\hat{k}} = \frac{\mathrm{Cov}(\mathbf{O}, \hat{\mathbf{L}}_{\hat{k}})}{\sigma_{\mathbf{O}} \sigma_{\hat{\mathbf{L}}_{\hat{\mathbf{k}}}}}$ for every $\hat{k}$.
6. Select the guess $\hat{k}$ for which the value of $|\rho_{\hat{k}}|$ is the highest.

Assuming there is a linear dependence between the predicted leakage and the physical observation, a significant correlation $\rho_{\hat{k}}$ should appear for the right guess $\hat{k}$, while for a wrong guess, the $\rho_{\hat{k}}$ should converge to zero.

[1] In other words, every trace is paired with the predicted Hamming weight/distance.

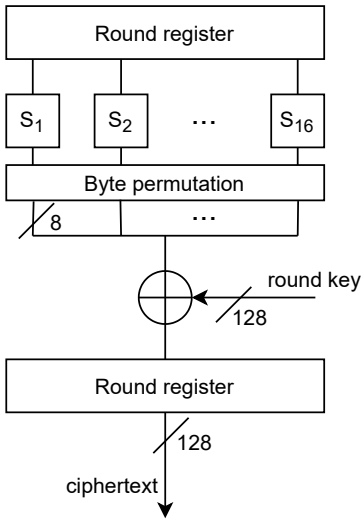


Figure 2.7: Architecture of Rijndael/AES last round. The scheme is unrolled for illustration purposes only, both "Round register" blocks depict the same hardware register.

Example: Attacking last round of Rijndael/AES

1. Assume the architecture illustrated in Figure 2.7 and a Hamming distance leakage (both v_1, v_2 must be derived). Let $\mathbf{Y} = \mathrm{E_k}(\mathbf{X})$, i.e., ciphertext.

2. Let $v_2 = y_i$. The previous register state is then $v_1 = f_{\hat{k}}(y_i) = \mathrm{Sbox}^{-1}(\mathrm{Perm}^{-1}(y_i \oplus \hat{k}))$. Both values are once again enumerable for each byte independently of the other bytes due to the fact that the MixColumns operation is not performed in the last round.

3. Enumerate byte subkey guesses $\hat{k} \in \mathcal{K} = \{0, 1, ..., 255\}$, and compute the intermediate value $f_{\hat{k}}(y_i), \forall \hat{k}, y_i$.

4. Compute the leakage function $\hat{\mathsf{L}}^{\mathrm{HD}}(f_{\hat{k}}(y_i), y_i), \forall \hat{k}, y_i$.

5. Compute Pearson correlation coefficient $\rho_{\hat{k}} = \frac{\mathrm{Cov}(\mathbf{O}, \hat{\mathbf{L}}_{\hat{k}})}{\sigma_{\mathbf{O}} \sigma_{\hat{\mathbf{L}}_{\hat{\mathbf{k}}}}}$ for every $\hat{k}$.

6. Select the guess $\hat{k}$ for which the value of $|\rho_{\hat{k}}|$ is the highest.

Unlike previously described DPA attacks, which use a partitioning approach, the CPA attack uses a comparative approach. However, CPA with a single-bit leakage function is equivalent to the original DPA. Interestingly, CPA is equivalent to normalized PPA with weights implicitly given by the distribution of bits in $f_{\hat{k}}(x_i)$. For a uniform distribution of bits in $f_{\hat{k}}(x_i)$, CPA is equivalent to the Bevan's multi-bit DPA [90].

The CPA attack assumes a linear relationship between the predicted leakage and the physical observation. However, this requirement can be relaxed to monotonicity by using

the Spearman coefficient instead of the Pearson coefficient [14]. Similar to DPA, CPA exploits statistical moments such as mean or covariance, and therefore requires a "normally" distributed observation channel. The success of the attack largely depends on the quality of the leakage approximation and present noise.

2.2.4 Mutual Information Analysis (MIA)

The Mutual Information Analysis [58] attack is performed in these steps:

1. Assume an arbitrary leakage function $\hat{\mathsf{L}} \in \mathcal{L}$ (with some restrictions, as explained in subsubsection 2.2.4.1).
2. Enumerate (sub)key guesses $\hat{k} \in \mathcal{K}$.
3. Compute an intermediate value $v_2 = f_{\hat{k}}(x_i), \forall \hat{k}, x_i$ (and the previous state v_1 if using Hamming distance).
4. Let $L_0, \ldots, L_l$ be subsets of $\mathcal{L}$ so that the set $\{L_0, \ldots, L_l\}$ is a partitioning of $\mathcal{L}$. The elements $L_j, j = 0, \ldots, l$, are called atoms.
5. Associate inputs x_i that leak L_j under key guess $\hat{k}$ to $L_j^{\hat{k}}$:
$$L_j^{\hat{k}} = \{x_i \,|\, \hat{\mathsf{L}}(v_1, f_{\hat{k}}(x_i)) \in L_j\}. \tag{2.13}$$
Each partition $\{L_0^{\hat{k}}, ..., L_l^{\hat{k}}\}$ induces a subdivision of measurements o_{x_i}.
6. Define conditional distributions $\{\mathbb{P}_{\mathbf{O} \,|\, L_j^{\hat{k}}}\}_{j=0}^{l}$ by using the subdivision of $\mathbf{O}$, and let $\mathbb{P}_{\mathbf{O}}$, $\mathbb{P}_{\mathbf{L}_{\hat{k}}}$ be probability distributions of $\mathbf{O}$, $\hat{\mathbf{L}}_{\hat{k}}$.
7. Select $\hat{k}$ with the highest mutual information $\mathbf{I}(\hat{\mathbf{L}}_{\hat{k}}; \mathbf{O})$.

Unlike DPA or CPA, the Mutual Information Analysis exploits mutual information, which is defined as
$$\mathbf{I}(\mathbf{X}; \mathbf{Y}) = D_{KL}(\mathbb{P}_{\mathbf{X},\mathbf{Y}} \,||\, \mathbb{P}_{\mathbf{X}} \otimes \mathbb{P}_{\mathbf{Y}}), \tag{2.14}$$
where D_{KL} is Kullback-Leibler divergence, i.e., a statistical distance describing probability distribution difference. Mutual information is directly related to entropy H:
$$\mathbf{I}(\mathbf{X}; \mathbf{Y}) = \mathrm{H}(\mathbf{X}) - \mathrm{H}(\mathbf{X}|\mathbf{Y}) = \mathrm{H}(\mathbf{X}) + \mathrm{H}(\mathbf{Y}) - \mathrm{H}(\mathbf{X}, \mathbf{Y}) = \mathbf{I}(\mathbf{Y}; \mathbf{X}), \tag{2.15}$$
where $\mathrm{H}(\mathbf{X}|\mathbf{Y})$ is conditional entropy, $\mathrm{H}(\mathbf{X}, \mathbf{Y})$ is joint entropy. Mutual information can be intuitively interpreted as the amount of information obtained about $\mathbf{X}$ by observing $\mathbf{Y}$ or, in other words, the reduction of uncertainty in $\mathbf{X}$ obtained by observing $\mathbf{Y}$.

The mutual information computation may go as follows:

1. Using measurements o_{x_i} belonging to $L_j^{\hat{k}}$, estimate the conditional distribution $\mathbb{P}_{\mathbf{O} \,|\, L_j^{\hat{k}}}$ and the conditional entropy $\tilde{\mathrm{H}}(\mathbf{O}|\hat{\mathbf{L}}_{\hat{k}} = j)$.

2. Compute the conditional entropy $\tilde{\mathrm{H}}(\mathbf{O}|\hat{\mathbf{L}}_{\hat{k}})$ by using $\{\tilde{\mathrm{H}}(\mathbf{O}|\hat{\mathbf{L}}_{\hat{k}} = j)\}_{j=0}^{l}$.

3. By using all of the measurements o_{x_i}, estimate the distribution $\mathbb{P}_{\mathbf{O}}$ and the entropy $\tilde{\mathrm{H}}(\mathbf{O})$.

4. Compute the mutual information $\tilde{\mathbf{I}}(\hat{\mathbf{L}}_{\hat{k}}; \mathbf{O}) = \tilde{\mathrm{H}}(\mathbf{O}) - \tilde{\mathrm{H}}(\mathbf{O}|\hat{\mathbf{L}}_{\hat{k}})$.

Example: Attacking AES/Rijndael with minimum assumptions

1. Assume an identity leakage function $\hat{\mathrm{L}}(f_{\hat{k}})$, e.g., three MSBs of $f_{\hat{k}}(x_i) = \mathrm{Sbox}(x_i \oplus \hat{k})$

2. Enumerate byte subkey guesses $\hat{k} \in \mathcal{K} = \{0, 1, \ldots, 255\}$, and compute the intermediate value $f_{\hat{k}}(x_i), \forall \hat{k}, y_i$.

3. For every subkey guess, associate each o_{x_i} with an atom of $\{L_i^{\hat{k}}\}_{i=0}^{7}$ based on its input's predicted leakage.

4. For every subkey guess, estimate the densities $\mathbb{P}_{\mathbf{O}}$ and $\mathbb{P}_{\mathbf{O}|\hat{\mathbf{L}}_{\hat{k}}}$ and compute the mutual information $\tilde{\mathbf{I}}(\hat{\mathbf{L}}_{\hat{k}}; \mathbf{O})$.

5. Select $\hat{k}$ with the highest mutual information $\tilde{\mathbf{I}}(\hat{\mathbf{L}}_{\hat{k}}; \mathbf{O})$.

Crucial aspect of Mutual Information Analysis is the estimation of probability densities $\mathbb{P}_{\mathbf{O}}$ and $\mathbb{P}_{\mathbf{O}|\hat{\mathbf{L}}_{\hat{k}}}$. Some of the choices include:

- histogram, i.e., a non-parametric discrete estimate;
- kernel density estimate, i.e., a non-parametric continuous estimate; and
- finite mixture model, i.e., a (semi-)parametric continuous estimate.

The quality of the estimate has a direct influence on both attack success and computational complexity [173, 15].

A histogram provides a simple and efficient estimate with the most critical parameter being the number of bins [173]. The best estimate would require as many bins as there are values in the domain; however, it would be problematic to get enough values in every bin for it to be statistically significant. Less bins result in less information, but also lower susceptibility to noise. The original MIA [58] proposes using $l + 1$ bins, i.e., as many bins as there are atoms in the $\mathcal{L}$ partitioning.

A kernel density estimate provides better attack results than the histogram [173] at the cost of its higher computational complexity. In this case, the kernel and bandwidth are the most critical parameters. Popular kernel choices include Epanechnikov (which is mean-square-error optimal) and Gaussian (for its convenience). Bandwidth has a similar role as bins in histograms, and it can be intuitively seen as a "smoothing parameter". Generally speaking, the attacker aims to select the bandwidth as small as allowed by

the data. A "rule-of-thumb" bandwidth estimator can be used alongside the Gaussian kernel [157].

A finite mixtures model assumes the underlying distribution to be a mixture of distributions, whose parameters are estimated, e.g., by using the expectation-maximization algorithm. Typically, a mixture of Gaussians is assumed [92].

Mutual information analysis puts no hard assumptions on the leakage function or the underlying distributions and provides sound results even with a simple identity leakage function. It provides a generic and powerful side-channel distinguisher (although it is less efficient in scenarios well-suited for DPA/CPA) [173, 15, 175].

2.2.4.1 Leakage Function in Partitioning Attacks

Mutual Information Analysis allows for an arbitrary leakage function, giving the attacker a great degree of freedom. Although Hamming weight or Hamming distance leakage functions may be used when it is possible to predict both v_1, v_2, their usage inherently leads to a loss of information. Identity leakage function, which is much more generic, may also be used. If the Hamming weight/distance estimate is possible, identity is shown to be less efficient, but still effective [173].

The leakage function must be selected so that a different $\hat{k}$ must not yield a permutation of $\hat{\mathbf{L}}_{\hat{k}}$. For example, assume that using the identity of Rijndael/AES bijective S-box output. Different $\hat{k}$ then leads to a permutation of $\{L_0^{\hat{k}}, \ldots, L_l^{\hat{k}}\}$, which means that the mutual information is constant and independent of $\hat{k}$ [58]. This limitation can be easily overcome, e.g., by choosing only seven least significant bits of the Sbox output, or by using a Hamming weight/distance [58].

This problem does not only pertain to MIA, but to every partitioning attack (all the presented non-profiled attacks except CPA). When $f_{\hat{k}}(x_i)$ is an injective function, an attack using trivial partitioning where each value belongs in its distinct class will always fail [162, 178].

2.2.5 Kolmogorov–Smirnov Analysis (KSA)

The Kolmogorov–Smirnov Analysis [173, 177] attack is performed in these steps:

1.-6. The first six steps are same as for Mutual Information Analysis in subsection 2.2.4. Define the conditional distributions $\{\mathbb{P}_{\mathbf{O}\,|\,L_j^{\hat{k}}}\}_{j=0}^{l}$ and the distribution $\mathbb{P}_{\mathbf{O}}$.

7. For every $\hat{k}$, compute the average Kolmogorov–Smirnov distance between $\mathbb{P}_{\mathbf{O}}$ and $\mathbb{P}_{\mathbf{O}\,|\,L_j^{\hat{k}}}$, optionally further normalized by $\frac{1}{|O_{L_j^{\hat{k}}}|}$, where $|O_{L_j^{\hat{k}}}|$ is a size of the measurements set belonging to atom $L_j^{\hat{k}}$:

$$\mathop{\mathbb{E}}_{j}\left(\frac{1}{|O_{L_j^{\hat{k}}}|} D_{KS}(\mathbb{P}_{\mathbf{O}} \,||\, \mathbb{P}_{\mathbf{O}\,|\,L_j^{\hat{k}}})\right). \tag{2.16}$$

8. Select the key guess $\hat{k}$ with the largest average KS-distance.

The Kolmogorov–Smirnov distance between $\mathbb{P}_{\mathbf{X}}$ and $\mathbb{P}_{\mathbf{Y}}$ is defined as

$$D_{KS}(\mathbb{P}_{\mathbf{X}} \,||\, \mathbb{P}_{\mathbf{Y}}) = \sup_{x} |F_{\mathbf{X}}(x) - F_{\mathbf{Y}}(x)|, \tag{2.17}$$

where $F_{\mathbf{X}}$ is a cumulative density function of $\mathbf{X}$. The Kolmogorov–Smirnov Analysis is heavily inspired by the Mutual Information Analysis. However, instead of estimating probability density function, the easier-to-obtain cumulative density function is used.

Alternatively, interclass Kolmogorov–Smirnov Analysis (iKSA) [98] distinguishes the key guess using the distance between the conditional distributions:

$$\frac{1}{2}\,\mathop{\mathbb{E}}_{j,j'}(D_{KS}(\mathbb{P}_{\mathbf{O}\,|\,L_j^{\hat{k}}} \,||\, \mathbb{P}_{\mathbf{O}\,|\,L_{j'}^{\hat{k}}})). \tag{2.18}$$

Other choices for comparison of the distributions include Cramér–von Mises criterion or different F-divergences [173].

The Kolmogorov–Smirnov Analysis shares some important characteristics with MIA, as both attacks can be used with an identity leakage function, and therefore without precise knowledge about the implementation and leakage. It can provide better results for weak signals than MIA due to its noise robustness [177], and since it utilizes CDF instead of PDF, it may be more convenient.

2.2.6 Differential Deep Learning Analysis (DDLA)

The Differential Deep Learning Analysis [166] attack is performed in these steps:

1. Assume an arbitrary leakage function $\hat{\mathsf{L}} \in \mathcal{L}$.
2. Enumerate (sub)key guesses $\hat{k} \in \mathcal{K}$.
3. Compute an intermediate value $v_2 = f_{\hat{k}}(x_i), \forall \hat{k}, x_i$ (and the previous state v_1 if using the Hamming distance).
4. Create labeled training datasets $\{(o_{x_i}, \hat{\mathsf{L}}(v_1, f_{\hat{k}}(x_i)))\}_{\hat{k}}, \forall \hat{k}, o_{x_i}$. Note that the same limitations as described in subsubsection 2.2.4.1 apply.
5. Perform Deep Learning classifier training for every dataset.
6. Select the key guess $\hat{k}$ with the best DL training metrics.

Unlike the previously described attacks in this section, the Differential Deep Learning Analysis is typically used in a multivariate fashion, not univariate. In other words, the attack is not performed at a single sampling point/all the sampling points in the trace independently. Instead, the classifier is fed with the multivariate vectors corresponding to the entire encryption.

The Differential Deep Learning Analysis is a partitioning attack, like all the previously presented attacks except CPA. A key-dependent partitioning of the data is created and then the distinguishability of the partitions is examined by using the classifier. For the correct key guess, the classifier should be able to learn distinctive features of differently labeled data. When a wrong guess is made, the traces are randomly distributed across labels, and therefore the training metrics should be significantly worse than for the correct guess. Different training metrics are proposed for the final selection of the key, e.g., by using sensitivity analysis [166].

Various deep-learning architectures, such as a multilayer perceptron or a convolutional network may be used for the classifier. Translation-invariance property of convolutional networks can be exploited to attack desynchronized traces [34, 166], whereas previously described attacks would require synchronization of the traces during preprocessing, e.g., by using autocorrelation, due to their univariate nature. A distinct disadvantage of using the machine-learning based blackbox approach is limited explainability of the results [171].

2.3 Profiled Attacks

Profiled attacks assume the attacker has a fully-controlled identical copy of the device under attack at her disposal. She is capable of observing the device's side channels during execution of the identical cryptographic implementation. Moreover, she is able to feed the implementation with arbitrary inputs and keys. Her attack is tailored for a specific device and therefore more effective and efficient than a non-profiled attack.

Profiled attack consists of two phases:

1. **Profiling phase**, during which an empirical model of the leakage is created using the identical copy of the device under attack.

2. **Attack phase**, during which observations of the device under attack are evaluated using the previously profiled model.

Unlike non-profiled attacks presented in section 2.2, all the presented profiled attacks are multivariate, i.e., full traces $\mathbf{O}(t)$ are considered, where the dimension (t) corresponds to a number of sampling points within a single trace.

2.3.1 Template Attack (TA)

The Template attack [40, 143] is performed in these steps:

1. Consider an arbitrary leakage function $\hat{\mathsf{L}} \in \mathcal{L}$.

   Profiling phase

2. Measure a profiling set of traces $o_{(x_i,k_i)}(t)$ using desired (typically, but not necessarily random uniform) inputs (x_i, k_i).

3. Compute an intermediate value $v_2 = f_{k_i}(x_i), \forall k_i, x_i$ (and the previous state v_1 if using Hamming distance).

4. Let $L_0, \ldots, L_l$ be subsets of $\mathcal{L}$ so that the set $\{L_0, \ldots, L_l\}$ is a partitioning of $\mathcal{L}$. The elements $L_j, j = 0, \ldots, l$, are called atoms.

5. Associate measurements $o_{(x_i,k_i)}$ whose inputs (x_i, k_i) leak L_j to O_j:

$$O_j = \{o_{(x_i,k_i)} \,|\, \hat{\mathsf{L}}(v_1, f_{k_i}(x_i)) \in L_j\}. \tag{2.19}$$

6. Select points of interest t_i within the measurements $o_{(x_i,k_i)}(t)$, e.g., by using sum of differences of average traces of each O_j set or by using principal component analysis. From this moment on, restrict the measurements to these points only.

7. Create empirical models, so-called templates, T_j, e.g., Gaussian probability estimates, characterizing leakage induced by atoms L_j using traces in set O_j.

Attack phase

8. Measure an attack set of traces $o_{x_i}(t)$ using desired plaintexts x_i.

9. Enumerate (sub)key guesses $\hat{k} \in \mathcal{K}$.

10. Compute an intermediate value $v_2 = f_{\hat{k}}(x_i), \forall \hat{k}, x_i$ (and the previous state v_1 if using Hamming distance).

11. Associate measurements whose inputs x_i leak L_j under key guess $\hat{k}$ to $O_j^{\hat{k}}$:

$$O_j^{\hat{k}} = \{o_{x_i} \,|\, \hat{\mathsf{L}}(v_1, f_{\hat{k}}(x_i)) \in L_j\}. \tag{2.20}$$

12. Compute probabilities $Pr(\mathbf{O} = o_{x_i} \,|\, \mathbf{L} \in L_j), o_{x_i} \in O_j^{\hat{k}}$ that measurements in $O_j^{\hat{k}}$ leak L_j using the templates T_j.

13. Select the key guess $\hat{k}$ with the highest overall probability (product of posterior probabilities) of the predicted leakage.

The Template attack provides the attacker with a very powerful and universal tool. The empirical templates T_j are typically multivariate Gaussian models [40]. A creation of the Gaussian template is demonstrated in the following examples.

Example: Attacking Rijndael/AES using Hamming weight and Gaussian templates

1. Assume Hamming weight leakage function $\hat{\mathsf{L}}$ and intermediate value $f_{\hat{k}}(x_i) = \text{Sbox}(x_i \oplus \hat{k})$.

Profiling phase

2. Measure a profiling set of traces $o_{(x_i,k_i)}(t)$, using random uniform inputs (x_i, k_i).

3. Partition measurements $o_{(x_i,k_i)}(t)$ into nine groups according to the Hamming weight of S-box output:
$$O_j = \{o_{(x_i,k_i)}(t) \mid \hat{\mathsf{L}}(f_{k_i}(x_i)) = j\}. \tag{2.21}$$

4. Select sampling points of interest $t_1, \ldots, t_m$ using sum of differences of average traces:

   a) Compute the average measurement $M_j(t)$ for every group O_j.

   b) Compute the sum of the absolute pairwise differences of these average power traces: $\sum_{i,j} |M_i(t) - M_j(t)|$ and select the most deviate points, preferably in different clock cycles.

   Reduce the dimensionality of $\mathbf{O}(t)$ and of the average traces $M_j(t)$ to these selected points only.

5. Define noise measurements as
$$N_j = \{n_{(x_i,k_i)}(t) \mid n_{(x_i,k_i)}(t) = o_{(x_i,k_i)}(t) - M_j(t) \wedge o_{(x_i,k_i)}(t) \in O_j\}, \tag{2.22}$$
and consider N_j samples from variable $\mathbf{N}_j$.

6. Compute the noise covariance matrices Σ_j between all the points of the interest for every group N_j:
$$\Sigma_j = \begin{pmatrix} \mathrm{Var}(\mathbf{N}_j(t_1)) & \ldots & \mathrm{Cov}(\mathbf{N}_j(t_1), \mathbf{N}_j(t_m)) \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}(\mathbf{N}_j(t_m), \mathbf{N}_j(t_1)) & \ldots & \mathrm{Var}(\mathbf{N}_j(t_m)) \end{pmatrix} \tag{2.23}$$

7. $T_j = (M_j, \Sigma_j)$ is a Gaussian template characterizing leakage $\mathsf{L} = j$, i.e., Hamming weight of the S-box output.

Attack phase

8. Measure/capture an attack set of measurements $o_{x_i}(t)$, by using random uniform plaintexts x_i.

9. Enumerate byte subkey guesses $\hat{k} \in \mathcal{K} = \{0, 1, \ldots, 255\}$ and compute the intermediate value $v_2 = f_{\hat{k}}(x_i), \forall \hat{k}, x_i$.

10. Partition the traces $o_{x_i}(t)$ into nine groups according to the predicted Hamming weight, for every subkey guess:
$$O_j^{\hat{k}} = \{o_{x_i}(t) \mid \hat{\mathsf{L}}(f_{\hat{k}}(x_i)) = j\}. \tag{2.24}$$

11. For every measurement $o_{x_i}(t)$ in group $O_j^{\hat{k}}$, evaluate the probability of it belonging in the designated group by evaluating the template $T_j = (M_j, \Sigma_j)$:

    a) Compute hypothetical noise vector $n_{x_i}(t) = o_{x_i}(t) - M_j(t)$.

    b) Compute the probability of observing $n_{x_i}(t)$ by using the multivariate Gaussian probability distribution:

$$p_j(n_{x_i}(t)) = \frac{1}{\sqrt{(2\pi)^N |\Sigma_j|}} \exp(-\frac{1}{2}\ n_{x_i}(t)^\top\ \Sigma_j^{-1}\ n_{x_i}(t)\ ), \tag{2.25}$$

    where N is number of points of the interest, $|\Sigma_j|$ is the determinant of Σ_j, and Σ_j^{-1} is its inversion.

12. Select the key guess $\hat{k}$ with maximum overall probability of the measurements being partitioned in the correct groups.

Example: Attacking Rijndael/AES using a single measurement

1. Assume identity leakage function $\hat{\mathsf{L}}$ and intermediate value $f_{\hat{k}}(x_i) = \hat{k}$. Assume an atom L_j for every subkey value, i.e., 256 atoms.

   Profiling phase

2. Measure a large amount of traces using uniform plaintexts and keys and create 256 Gaussian templates.

   Attack phase

3. Measure a trace using a uniform plaintext and evaluate it against all the templates. Select the key guess with the highest probability.

Computing the probabilities as described above may lead to numerical instabilities, which can be solved by using logarithms of probabilities instead [100].

The creation of the model (i.e., the templates) requires a large amount of measurements in comparison to the actual attack. Efficient and effective templates may require further evaluation due to potential overfitting: templates too specific for the attacker's copy might not work on the device under attack [40].

Various ways to improve the efficiency of the template attack are described in [42], such as methods for the dimensionality reduction/selection of points of the interest, usage of pooled covariance matrices, combining multiple traces, etc.

Many use cases and scenarios are possible by using the extend-and-prune approach, i.e., starting with small parts of information and increasingly extending the attack. Thanks to the profiling phase, the attack phase is very effective and efficient.

2.3.2 Machine Learning-based Attacks

Machine Learning (ML) algorithms are algorithms that learn to solve a problem without being explicitly programmed to do so. In this context, "to learn" can be perceived as "to build an empirical model using training data", whereas "to solve" can be perceived as "to evaluate real data using the model". A profiled side-channel attack as described in this section can be reduced to a classifying task, which is thoroughly studied in the context of supervised machine learning [87].

Profiled machine learning-based attacks are typically performed in a similar fashion as the Template attack, and can often be classified as one. They also share many of its advantages and disadvantages. The crucial difference is in the choice of the empirical model: instead of Gaussian, a machine learning-based classifier is used.

A Support Vector Machine (SVM) is a machine learning algorithm commonly used for classification, based on creating an optimal hyperplane between different classes. It was shown to be more efficient than Gaussian Template attack in some aspects [72, 68, 93], performing better on noisy measurements and requiring a smaller profiling set. However, selection of the algorithm parameters, such as the kernel function, may have a significant impact on its performance [72]. Both binary and multi-class SVM classifiers were successfully used to attack Rijndael's S-box output [13]. Other common classifier choices are Decision trees, Random forests [93, 94], and others.

Neural network-based deep learning classifiers are a popular choice in side-channel security [17, 66]. The non-profiled variant of the attack is presented in subsection 2.2.6. Both multilayer perceptron [103, 102] and convolutional neural network [97, 34] architectures are suitable for a profiling attack. Whereas a multilayer perceptron classifier must be fed with aligned power traces, the location and scale invariant convolutional neural network can extract the features itself and therefore it is capable of processing misaligned or jittered measurements without prior preprocessing [34]. It is capable of exploiting both univariate and multivariate leakage, as well as utilizing both Hamming weight/distance or identity training labels [89].

Hyperparameters of the neural network model include the network architecture (number of nodes in a layer, number of layers, activation function) and learning parameters (number of epochs, batch size, optimization algorithm, learning rate). Unfortunately, there does not seem to be the best model for every scenario ("no free lunch" theorem). Finding a suitable model is a nontrivial task; however, it is crucial for a successful attack. Class imbalance may present a significant obstacle [132], especially when Hamming weight is used (e.g., only a single word value $x \in \mathbb{B}^n$ leads to $\mathrm{HW} = 0$, in contrast to $\mathrm{HW} = \frac{n}{2}$). Even though the neural network can be fed the whole unprocessed and even misaligned traces, it holds that the higher is the dimension of the data, the higher is the attack complexity and the larger training sets are required [94]. Similarly to the Template attack, both underfitting and overfitting of the model during the learning phase may lead to an unsuccessful attack: the former cannot generalize the observations, whereas the latter learns non-relevant details and noise [89].

2.4 Side-Channel Attack Related Metrics

Several metrics related to side-channel attacks are presented in this section. Experimental metrics Success rate and Guessing entropy are presented in subsection 2.4.1. Theoretical metrics Confusion coefficient, Distinguishing margin, and their relationship to differential cryptnalysis are presented in subsection 2.4.2.

Let $\hat{k} \in \mathcal{K}$ be a (sub)key guess during an attack and let $k^* \in \mathcal{K}$ be the real (secret) (sub)key. Define a distinguisher $\mathcal{D}^{\hat{k}}(o_{x_1}, \ldots, o_{x_q}; x_1, \ldots, x_q)$ as an absolute value of the statistic that is used to distinguish the correct key during the attack. For example, a difference of means or t-value in case of DPA (subsection 2.2.1), a correlation coefficient in case of CPA (subsection 2.2.3), a mutual information in case of MIA (subsection 2.2.4), a Kolmogorov–Smirnov distance in case of KSA (subsection 2.2.5), probabilities in case of DDLA (subsection 2.2.6) and TA (subsection 2.3.1). Assume that the higher the value of $\mathcal{D}_{\hat{k}}$ is, the higher is the probability of the correct key $\hat{k}$.

2.4.1 Success Rate and Guessing Entropy

Success rate [164] and Guessing entropy [104, 86] are experimental metrics allowing a comparison of different attacks on the same implementation. A simplified definition of these metrics is presented in this subsection, omitting two parameters: time complexity τ and memory complexity m [164].

Assume all the key guesses $\hat{k}$ are sorted according to the value of $\mathcal{D}^{\hat{k}}$ in descending order, i.e., the most probable candidate is positioned first, the second most probable candidate is positioned second, etc. Let $\#(\hat{k})$ be a position of the guess $\hat{k}$. Success rate is then defined as the probability of the correct key guess k^* being on the first position:

$$\mathrm{SR}(q) = \Pr(\#(k^*) = 1), \tag{2.26}$$

where q is the number of measurements available. In other words, it is the probability of the attack revealing the correct key. The n-th order Success rate is defined as

$$\mathrm{Succ}^n(q) = \Pr(\#(k^*) \leq n). \tag{2.27}$$

Guessing entropy is a related metric defined as the expected position of the correct key within the previously mentioned sorted guesses:

$$\mathrm{GE}(q) = \mathbb{E}(\#(k^*)), \tag{2.28}$$

where q is a number of measurements available. Whereas Success rate characterizes the probability of the attack being successful, Guessing entropy characterizes the amount of the remaining work of the attacker when the attack fails to reveal the correct key.

In order for the values of Success rate or Guessing entropy to be trustworthy, a large number of independent experiments must be performed [164].

2.4.2 Confusion Coefficient and Distinguishing Margin

Consider a single bit intermediate value $f_k(x)$ (as in a DPA attack). Let k^* be the correct key and $k \in \mathcal{K}$ be any key hypothesis. The confusion coefficient $\kappa(k^*, k)$ is then defined as a probability of the bit $f_k(x)$ having a different value given two different keys [54]:

$$\kappa(k^*, k) = \Pr(f_{k^*}(x) \neq f_k(x)). \tag{2.29}$$

It reaches minimum when the two keys are the same: $\kappa(k, k) = 0$, and maximum $\kappa(k^*, k) = 1$ iff $\exists k \neq k^*, \forall x : f_{k^*}(x) = \overline{f_k(x)}$. Assuming $f_k(x) = \mathrm{S}(x \oplus k)$, the Confusion coefficient is directly linked to cryptanalytical metrics of Boolean S-box $S : \mathbb{B}^n \to \mathbb{B}^m$, namely to its differential uniformity Δ_S:

$$\Delta_S = \max_{a \in \mathbb{B}^m, k \in \mathbb{B}^n} |\{x \in \mathbb{B}^n \,|\, S(x) \oplus S(x \oplus k) = a\}|. \tag{2.30}$$

Considering $m = 1$ ($f : \mathbb{B}^n \to \mathbb{B}^m$ being equivalent to $\{f_i : \mathbb{B}^n \to \mathbb{B}\}_{i=1}^m$) [67]:

$$2^{-n}\Delta_S - \frac{1}{2} = \max_{k \neq k^*} |\kappa(k^*, k) - \frac{1}{2}|. \tag{2.31}$$

Let b be one bit of a sensitive variable $f_k(x)$ for a perfectly secret encryption algorithm. Then b is equiprobable, i.e., $\Pr(b = 1) = \Pr(b = 0) = \frac{1}{2}$ [80]. Assume that the bit b is the one under attack. In such a case, the DPA/CPA and KSA/iKSA distinguishers can be rewritten in following closed-form expressions [67]:

$$\mathcal{D}^k_{\mathrm{CPA}} = \mathcal{D}^k_{\mathrm{DPA}} = \frac{2}{\sqrt{1 + 1/\mathrm{SNR}}} \cdot |\kappa(k^*, k) - \frac{1}{2}|, \tag{2.32}$$

$$\mathcal{D}^k_{\mathrm{KSA}} = 2\mathcal{D}^k_{\mathrm{iKSA}} = (2\Phi(\sqrt{\mathrm{SNR}}) - 1) \cdot |\kappa(k^*, k) - \frac{1}{2}|, \tag{2.33}$$

where $\Phi(x)$ is a cumulative distribution function of the standard noise $\mathcal{N}(0, 1)$. These equations describe the relationship between the distinguisher and noise. Notice that for large noise, the first multiplicand in both equations tends to zero.

Let the Distinguishing margin (distance to the nearest rival) be the distance between the correct key k^* distinguisher value $\mathcal{D}^{k^*}$ and the maximum incorrect key guess $\hat{k}$ distinguisher value [176]:

$$\mathrm{DM} = \mathcal{D}^{k^*} - \max\{\mathcal{D}^{\hat{k}} | \hat{k} \neq k^*\}. \tag{2.34}$$

The distinguishing margin characterizes the ability of the attacker to distinguish the correct key, i.e., her ability to make the attack succeed. For the Kolmogorov–Smirnov distinguisher, it can be explicitly expressed in terms of Confusion coefficient, and therefore differential uniformity [67]:

$$\mathrm{DM}_{\mathrm{KSA}} = \lambda\,(\frac{1}{2} - \max_{k \neq k^*} |\kappa(k^*, k) - \frac{1}{2}|) = \lambda(1 - 2^{-n}\Delta_S). \tag{2.35}$$

This equation demonstrates that the attack becomes easier as the distance between κ and $\frac{1}{2}$ becomes smaller. It also provides a direct link between S-box properties, i.e., its differential uniformity, and its susceptibility to side-channel attacks: the harder the differential cryptanalysis, the easier the side-channel analysis.

2.5 Countermeasures Against Attacks

Countermeasures against side-channel attacks can be categorized in two basic groups [100]:

- **Hiding**, whose main objective is to "hide" the sensitive variable leakage, ideally to entirely remove the data dependency of the $\mathbf{L} \rightarrow \mathbf{O}$ channel. Hiding countermeasures generally focus on the signal-to-noise ratio (recall Equation 2.32 and Equation 2.33). Hiding countermeasures can sometimes be further classified as (1) hiding in amplitude, and (2) hiding in time. **Shuffling**, which randomizes the algorithm flow, is sometimes considered a separate category, as it is implemented on the algorithm level; however, its effect is similar to that of hiding.

- **Masking** randomizes the processed data $\mathbf{W}$ while still providing correct results, therefore making it hard (ideally impossible) for the attacker to predict any intermediate values. The aim is to make the $\mathbf{W} \rightarrow \mathbf{L}$ channel appear random, ideally to remove the data dependency altogether. Unlike hiding countermeasures, masking typically requires a source of fresh randomness. Security of the masking schemes is therefore dependent on the used random generator.

Correctly employing the presented countermeasures does not result in an absolutely secure implementation. The objective is to make the attack infeasible in a real-world scenario, typically by increasing either the number of measurements necessary or the computational cost of the attack over a limit of resources practically available to the attacker. Similarly to the classic cryptanalysis, this limit lowers in time as the available computational capacity increases. Real-world attack examples [95] in 2021 show that protection against extreme numbers of traces (hundreds of millions or more) is necessary. Attacks on protected implementations are presented later in section 2.6.

In implementation terms, the countermeasures can further be categorized in three groups [107]:

- **Secure logic styles**, which incorporate custom logic gates libraries, designed to minimize the data-dependency. These countermeasures inherently introduce a large overhead and their implementation is typically expensive. Countermeasures in this category are generally "hiding in amplitude".

- **Additional modules**, which are incorporated into the cryptographic design. The basic advantage of these modules is their universal applicability and lower cost than that of the secure logic style. Countermeasures in this category are generally hiding countermeasures, either in amplitude or in time.

- **Cryptographic module modifications**, which aim to alter the encryption itself. While offering a lower overhead than the previously mentioned categories, they are often algorithm-specific and may present unforeseen weaknesses. This category contains, most importantly, masking. However, some hiding countermeasures fall within the category as well.

Additionally, some types of cryptography were believed to be resistant to SCA by their nature. For example, chaos cryptography [105, 119] was expected to be hard to attack because of its unpredictable behaviour [99]. Nevertheless, these beliefs were disproven, e.g., by a study showing that the chaos-based S-Boxes are similarly vulnerable to SCA as the AES ones [1]. Similarly, ARX-based cryptography [16, 9] was expected to be more resistant to SCA, as there is no highly nonlinear element (S-Box) whereas later research showed the opposite [78, 124].

2.5.1 Secure Logic Styles

The logic gates are typically designed in static CMOS technology, as illustrated in Figure 2.1. Various logic styles aim to compensate for their data-dependent behavior, often combining concepts of differential and dynamic logic [137]. Differential logic uses complementary signals (e.g., A and $\overline{A}$) at gate inputs and output, implementing differential pull-down networks to equalize the power consumption of different transitions. Two-phase dynamic logic introduces clock-driven pre-charge, where the load capacitance is artificially charged. Combination of the two styles is often referred to as dynamic differential logic or dual-rail pre-charge logic [107].

Various countermeasures based on secure logic styles were proposed in the literature. Sense Amplifier-based Logic (SABL) [167] is an example of dynamic differential logic. It aims to ensure constant consumption of all input and output transitions at the cost of complicated design (custom logic gates, "Domino" logic), under the assumption that all interconnections and capacitances are symmetrical (balanced). Simple Dynamic Differential Logic (SDDL) [168] uses ordinary CMOS gates, implementing the differential logic by using De Morgan's law and the pre-charge using AND gates. Wave Dynamic Differential Logic (WDDL) [168] extends the SDDL by limiting the used logic to AND and OR gates, thanks to which a "precharge wave" is introduced to reduce overhead. Furthermore, WDDL promises to be glitch-resistant, as opposed to SDDL, where data-dependent hazards may compromise the countermeasure. Similarly to Sense amplifier-based logic, SDDL and WDDL require symmetrical interconnections and capacitances. Other extensions of dynamic differential logic are available in the literature [32, 11, 142].

Adiabatic logic [112] was originally designed for low-power applications with the aim of reusing energy efficiently instead of it being discharged. It is powered by a clock-controlled power source, typically trapezoidal. Various logic styles which make use of adiabatic logic were proposed with the aim of hiding leakage [146, 41].

Other logic style examples include randomized multitopology logic [10] or use of asynchronous logic styles [28, 27, 26].

2.5.2 Additional Modules

Unlike countermeasures based on secure logic styles, the countermeasures presented in this subsection put no (or minor) assumptions on the cryptographic module.

Measured SNR can be effectively lowered by employing noise generators inside the cryptographic device. Different primitives can be utilized to build the generator, including shift registers, block RAMs, switch boxes [64] or ring oscillators [184]. The design of the actual cryptographic primitive can be used for correlated noise generation [79, 6].

Current sense-shunt loop-back can be used for active current flattening to hide the leakage [141, 118]. Similarly, the current can be randomized by using a variable current source [73]. Decoupling-based countermeasures are based on powering the cryptographic core from internal capacitors [153, 169, 106] in order to hide the instantaneous consumption.

Hiding in time is typically achieved by employing a specific clock signal or by altering the cryptographic algorithm. Isolated clock network can be used to deny the attacker the possibility of using the global clock network for synchronization [130]. The clock signal can be randomized [64]. Alternatively, dummy operations and data can be inserted randomly during the computation [44, 33, 74]. Partial dynamic reconfiguration can be used to shuffle the algorithm execution to hide the leakage in both time and amplitude [108].

2.5.3 Masking

Unlike previously presented countermeasures, masking [39] requires a detailed knowledge of the cryptographic algorithm. Its implementation is modified so that all intermediate values are masked by using a random value, making it difficult for an attacker to predict the leakage and therefore mount an attack. A relevant function (group law) is chosen for the masking according to the values domain, e.g., an exclusive or (XOR) in case of Galois field—then the masking is called Boolean. When the sensitive value is multiplied by a mask, the masking is called multiplicative. Unlike Boolean masking, multiplicative masking is inherently unable to mask a zero value [56].

Unless stated otherwise, Boolean masking is considered in this subsection. The sensitive value x is split into $d+1$ shares x_i, where

$$x = \bigoplus_{i=0}^{d} x_i. \tag{2.36}$$

The splitting is done by generating d uniform random masks $x_1, \ldots, x_d$ and by putting $x_0 = x \oplus x_1 \oplus \ldots \oplus x_d$. The number of masks d is then called a masking order. Implementation secured with d-order masking should ideally be secure against attacks up to d-th order [39, 134] (as defined further in section 2.6). However, the desired security level is often not reached in practice [101, 113] due to unforeseen imperfections.

A cryptographic algorithm typically consists of several linear and nonlinear operations, some of which must be altered to function properly when the variable is split. Masking of a linear operation f is trivial because all of the shares can be processed independently:

$$f(x) = f(x_0 \oplus \ldots \oplus x_d) = f(x_0) \oplus \ldots \oplus f(x_d). \tag{2.37}$$

There are different approaches to dealing with the nonlinear operations. Considering substitution-permutation network-based ciphers, substitution boxes (S-boxes) are typically the nonlinear operations.

Pre-computed masked S-boxes were originally proposed for first-order masked software implementations [109] and later adapted for hardware [64, 148]. The concept is further described in subsubsection 2.5.3.1.

A more efficient approach suitable for hardware Rijndael/AES implementations splits the S-box into an inversion and an affine operation and masks the inversion by using a multiplicative mask [5]. Even lower overhead can be achieved when the S-box computation is performed in a composite field [170, 126, 38]. However, these hardware masking schemes were later shown to be vulnerable against first-order side-channel attacks due to data-dependent glitches occuring during the S-box computation [101]. An example of glitch-induced leakage in a masked AND gate is shown in [121, section 4.1].

This issue is solved by glitch-resistant masking schemes, such as Domain oriented masking [63] or Threshold implementation [121, 122], which is further described in subsubsection 2.5.3.2. Lower overhead of these schemes can be once again achieved by using a composite field S-box computation [116, 23].

2.5.3.1 Pre-computed Masked Substitution Boxes

Pre-computed masked S-boxes were originally proposed for first-order masked software implementations [109] and later utilized in hardware (FPGA) by using Block RAM [64] or more efficient CFGLUT [148] primitives. In the following paragraphs, the concept will be described as used for PRESENT [25] encryption. PRESENT is a lightweight substitution-permutation network-based cipher with a block size of 64 bits and possible key sizes of 80 or 128 bits. Each round consists of a round key addition (XOR), a non-linear substitution layer (4-bit S-boxes applied 16 times in parallel), and a linear permutation layer. After 31 rounds, the 32^{nd} round key is finally added to produce the ciphertext.

Assuming the PRESENT encryption algorithm accepts plaintext pt masked by XORing a random mask m:

$$state' := pt \oplus m, \tag{2.38}$$

where $state'$ is the masked cipher state, three round operations/layers must be taken into account and altered appropriately so that equation

$$state = state' \oplus m \tag{2.39}$$

holds, allowing the ciphertext to be obtained using $state'$.

The first layer, Round Key Addition, i.e., XOR, is a commutative and associative operation:

$$state' \oplus rk = (state \oplus rk) \oplus m. \tag{2.40}$$

Therefore, addition of the round key rk does not require any further alteration since the output of the layer is already equal to the valid cipher state masked by m.

The last layer, the Permutation layer, is a linear transformation P, used to permute bits of the cipher state. The output of the layer is therefore equal to the valid cipher state masked by a permuted mask:

$$P(state') = P(state) \oplus P(m), \tag{2.41}$$

which means the mask that would need to be subtracted to obtain the valid cipher state changes to $P(m)$.

The middle layer is a non-linear Substitution layer S. The validity of the output is assured by altering the substitution look-up table into a masked substitution layer S'

$$S'(state') := S(state' \oplus m) \oplus P^{-1}(m), \tag{2.42}$$

which realizes the original substitution upon masked input value and outputs the substitution result masked by m processed with inverse permutation P^{-1}. This approach countermands the mask alteration performed by the Permutation layer, since

$$P(state \oplus P^{-1}(m)) = P(state) \oplus P(P^{-1}(m)). \tag{2.43}$$

Therefore, S' is the only alteration which must be performed for Equation 2.39 to hold.

In this example, the mask m is used through entire encryption, allowing usage of a single precomputed substitution layer. However, special care must be taken when the round state is written to a CMOS register holding the previous round state. Assuming the Hamming distance leakage model, the mask m would get subtracted:

$$\mathrm{HD}(x \oplus m, y \oplus m) = \mathrm{HW}(x \oplus y \oplus m \oplus m) = \mathrm{HD}(x, y). \tag{2.44}$$

One possible solution to this problem is a combination with a register precharge hiding countermeasure [148], where the working register is doubled and the encryption context is interleaved with random data.

2.5.3.2 Threshold Implementation

Threshold implementation [121, 122] is a glitch-resistant masking scheme suitable for both hardware and software implementations [147].

According to the selected masking order d, the input is first split into $d + 1$ shares as described in Equation 2.36. Linear operations during computation are performed on each share independently as described in Equation 2.37. Each non-linear operation f is split into $d + 1$ shared functions $f_0, \ldots, f_d$ over which the following properties are defined: correctness, non-completeness and uniformity.

Correctness property assures that the correct result of f can be obtained after the computation:

$$\bigoplus_{i=0}^{d} f_i(x_0, \ldots, x_d) = f(\bigoplus_{i=0}^{d} x_i). \tag{2.45}$$

Non-completeness property requires each function f_i to be independent of at least one share of each input variable, e.g.,

$$\begin{aligned} &f_0(x_1, x_2, \ldots, x_d), \\ &f_1(x_0, x_2, \ldots, x_d), \\ &\qquad \ldots \\ &f_d(x_0, x_1, \ldots, x_{d-1}). \end{aligned} \tag{2.46}$$

For the masking scheme to protect against higher-order attacks, the property must be extended to the d–th order non-completeness [22]: any combination of up to d shared functions f_i must be independent of at least one share of each input variable.

Similarly, as the inputs x_i are uniformly shared, which is assured by generating uniform masks, the uniformity property requires the output of the shared functions f_i to be uniformly shared as well. Unlike previous properties which can be explicitly validated, uniformity is typically checked by using an exhaustive enumeration and conditional probability examination. Since uniformity is often hard to achieve directly, remasking with a fresh randomness may be necessary after the non-linear stage [24].

To eliminate the propagation of glitches, assure non-completeness when consecutive non-linear operations are considered, and possibly split a single non-linear operation, pipeline registers must be used between the stages. At least $d + 1$ shares are required to implement a function of algebraic degree d (e.g., Rijndael/AES S-box has algebraic degree 7). Splitting the non-linear stage (e.g., decomposing the function or computing the S-box in a composite field) may result in functions of a smaller algebraic degree, therefore, a smaller number of shares and lower overall overhead [133, 116].

2.6 Attacks on Protected Implementations

Approaches to attacking protected implementations are presented in this section. Attacks on hiding countermeasures are summarized in subsection 2.6.1 and attacks on masking are explained in subsection 2.6.2.

Most of the presented techniques are typically performed as a pre-processing step before mounting an attack. Moment-based attacks (recall section 2.2) on masking can be computed in an online and parallel fashion [152], sparing computing resources. Machine learning-based attacks may even be mounted on protected implementations in a same fashion as when attacking unprotected implementations [166].

2.6.1 Attacks on Hiding

Different approaches were proposed to deal with hiding in time. Simple time shifts can be overcome by using autocorrelation, i.e., a correlation of a signal with a delayed copy of itself. More generally, a pattern near the sensitive operation can be used with matching techniques known from digital signal processing [100] to identify the time shift and align traces appropriately. These methods are also useful when there is no dependable synchronization signal for measurements (trigger).

When the leakage is spread in time in a more chaotic manner, e.g., by clock jitter, a sliding window attack may be used, where a finite number of (consecutive or not) time samples is summed/integrated to a single value [44]. When attacking implementations with a hiding in time countermeasure in place, this attack results in better signal-to-noise ratio. However, because the noise is integrated as well, it is still less efficient than a direct attack on an unprotected implementation [44].

Another example of an attack on hiding in time is the use of elastic alignment attack [172], which utilizes dynamic time warping techniques [43] to create well-aligned traces.

Machine-learning based attacks were shown to successfully break through hiding in time countermeasures, most importantly convolutional neural networks, thanks to their location-scale invariance properties [34].

Similarly, different techniques were proposed for attacking hiding in amplitude countermeasures. Differential logic (such as WDDL) without proper place&route constraints can be successfully attacked by using electromagnetic analysis [149], as the attacker is able to measure leakage from only a small part the chip. Various approaches to filter out excessive noise (such as that created by noise generators) were also proposed [91, 159, 48, 3], e.g., based on wavelet transform.

2.6.2 Attacks on Masking

Consider a moment-based non-profiled attack (e.g., DPA or CPA). With a masking countermeasure in place, the intermediate sensitive variable $f_k(x_i)$ is split into d shares (recall Equation 2.36). Side-channel attack targeting this intermediate value therefore must consider d mutually independent leakages. Such an attack is then referred to as a higher-order, or d-th order, attack [83]. The d leakages may manifest themselves at different times, resulting in a multivariate higher-order attack. Similarly, when the leakages manifest at the same time, a univariate higher-order attack is mounted.

The combining function C is used to combine multiple key-independent noisy distributions to produce a single key-dependent distribution, which is then exploited by the attack. Two different combining functions are presented in this subsection: absolute difference combining and product combining. The centralized absolute difference combining between two time samples $o_{x_i}(t_1), o_{x_i}(t_2)$ with mean values $\mu_{o(t_1)}, \mu_{o(t_2)}$ is defined as [110]

$$\mathsf{C}(o_{x_i}(t_1), o_{x_i}(t_2)) = |\,(o_{x_i}(t_1) - \mu_{o(t_1)}) - (o_{x_i}(t_2) - \mu_{o(t_2)})\,|, \tag{2.47}$$

and the centralized product combining between arbitrary number of samples is defined as [39]

$$\mathsf{C}(o_{x_i}(t_1), \ldots, o_{x_i}(t_d)) = \prod_{k \in \{1,\ldots,d\}} (o_{x_i}(t_k) - \mu_{o(t_k)}). \tag{2.48}$$

The centralization, i.e., subtracting the mean, normalizes the Gaussian noise and minimizes bias during the combining [135]. Optimal leakage function $\hat{\mathsf{L}}$ to use with combined leakage is then combining function specific.

Assume attacking a first-order masking scheme, i.e., a second-order attack, using Hamming weight model. Denote the targeted intermediate variable $z = f_k(x_i) \in \mathbb{B}^n$, and let it be split in two shares $s_0, s_1 \in \mathbb{B}^n$:

$$s_0 = z \oplus m, \quad s_1 = m, \tag{2.49}$$

where $m \in \mathbb{B}^n$ is a uniform independent mask. Model the observed leakage during processing of each share as:

$$\mathsf{O}_0 = \delta_0 + \mathrm{HW}(s_0) + B_0, \quad \mathsf{O}_1 = \delta_1 + \mathrm{HW}(s_1) + B_1, \tag{2.50}$$

where δ_0, δ_1 are constant parts of the leakage, and $B_0, B_1 \sim \mathcal{N}(0, \sigma)$ are zero-centered Gaussian noise. For Hamming distance model, assume usage of $z' = z \oplus v_1^0 \oplus v_1^1$, $m' = m \oplus v_1^1$ instead of z, m, where v_1^0, v_1^1 are previous states of z, m, respectively. The optimal leakage prediction function $\hat{\mathsf{L}}$ for the centered absolute difference combining is then [135]

$$2^{1-\mathrm{HW}(z)}\,\mathrm{HW}(z)\binom{\mathrm{HW}(z)-1}{\lfloor \frac{\mathrm{HW}(z)}{2} \rfloor}, \tag{2.51}$$

i.e., a non-affine function of the Hamming weight. The optimal leakage prediction function for the centered product combining is [135]

$$-\frac{1}{2}\mathrm{HW}(z) + \frac{n^2+n}{4} + \frac{n}{2}(\delta_1+\delta_2) + \delta_1\delta_2, \tag{2.52}$$

i.e., a linear function of the Hamming weight. The centered product combining is therefore well-suited for a correlation attack (CPA), where simple $\mathrm{HW}(z)$ predictions will show up as a negative correlation. Assuming very noisy observations, the centered product combining function leads to a more efficient attack than the absolute difference combining [135]. Moreover, a higher-order attack using centered product combining can be computed in a one-pass and parallel fashion [152], instead of pre-processing the data.

As mentioned earlier, the higher-order attack can be either univariate or multivariate. Multivariate attack is usually used when attacking masked software implementations [110] and may require a prior points-of-the-interest analysis (recall the Template attack in subsection 2.3.1); otherwise, it may become very expensive in terms of both computational power and memory. Univariate attack is suitable when the implementation leaks information about all of the shares in the same time instant [174], which is usually the case for hardware implementations. In such cases, the time sample will be combined with itself, e.g., $(o_{x_i} - \mu_o)^2$ assuming a second-order attack and product combining. Notice that the combining result is equal to the second central moment.

Combining the samples also results in an amplification of the noise [39]. The amount of side-channel information necessary for a successful attack grows exponentially with the masking order. Assuming variance of a single observation is σ^2, variance of k combined samples is approximately $(\sigma^2)^k$, and to distinguish between two distributions with different means and $(\sigma^2)^k$ variance, approximately $(2\sigma^2)^k$ samples are necessary [39]. Therefore a sufficient noise level is necessary for masking countermeasures to be secure [161].

Mutual Information Analysis is, unlike DPA or CPA, "naturally higher-order" in the sense of examining the entire underlying distribution instead of statistical moments [15]. For a multivariate attack, there are different methods of combining multiple time samples: (1) considering them a d-dimensional vector, (2) computing multivariate mutual information, or (3) computing total correlation. Multi-dimensional probability density function must then be estimated.

Machine learning-based attacks were also shown capable of exploiting higher-order leakage [166, 59] and successfully breaking masked implementations. Compared with CPA, a machine-learning based attack might not require any alterations, and it may be performed on both unprotected and protected implementations with no adjustments [166].

2.7 Leakage Assessment

Leakage assessment methodology examines whether the implementation leaks information. A naïve technique of testing vulnerability against side-channel attacks would be mounting all the known attacks. Leakage assessment methodologies offer a more general and less computationally and time demanding approach. Similarly to the non-profiled attacks, the methods presented in this section are typically based on partitioning the measurements and examining their distinguishability. Contrary to the attack scenario, the evaluator in this scenario has full control over the implementation. The described tests can be categorized as either specific or non-specific tests [62, 151].

The specific tests [62] typically evaluate measurements of random uniform plaintext encryptions with a fixed key. They partition the measurements into two or more groups according to a selected intermediate value and leakage function. For example, assuming single-bit Hamming weight leakage (similarly to DPA in subsection 2.2.1), the measurements are partitioned into two groups according to the value of a bit of S-box output. Considering Rijndael/AES, this intermediate value provides 128 different partitionings (one for each bit of the cipher context). Other options for the intermediate value include round output or XOR of round input and output. Distinguishability of the groups then suggests the possibility of presence of leakage exploitable by targeting the selected intermediate value.

The non-specific tests [62, 151] do not target a specific intermediate value. Instead, e.g., distinguishability between two groups containing measurements of an encryption of either random uniform plaintext, or of pre-selected fixed plaintext, is tested. Such tests are referred to as Random vs. Fixed tests. The other choice is a Fixed vs. Fixed test. In such tests, both groups must be measured in a randomly interleaved fashion during a single evaluation to prevent false results, e.g., due to environmental noise or varying device temperature [151]. Distinguishability of the two groups once again suggests information leakage. Non-specific tests are more sensitive and general than specific tests, and they provide only limited information about the leakage origin.

Various statistical tools can be used to test the distinguishability of the groups. Methodologies based on Welch's t-test and Pearson's χ^2 test are described in subsection 2.7.1 and subsection 2.7.2. A deep learning-based approach is described in subsection 2.7.3.

The measurement setup plays a crucial role in leakage evaluation. Test equipment (e.g., oscilloscope) with sufficient bandwidth, sampling rate and resolution must be used [62], as these and other parameters have a direct impact on potential attack success [127]. A preamplifier may be used to ensure the full range of the ADC is utilized.

Relevant pre-processing should also be performed, especially when evaluating secured implementations (see section 2.6). The role of evaluator is always creative and non-trivial in the sense that the evaluator should consider all possible techniques the attacker may use to increase the chance of attack success. Results of the presented methods must be interpreted carefully, with possible false positives and false negatives in mind [161].

2.7.1 Welch's t-test

A two-tailed Welch's t-test can be used to examine a *null hypothesis* that two groups' means are equal, and can be successfully used in leakage assessment [62, 151]. The univariate statistic t is computed for the two groups, in every sampling point independently:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}, \tag{2.53}$$

where $\bar{X}_1$, $\bar{X}_2$ are sample means, s_1^2, s_2^2 are sample standard deviations and N_1, N_2 are cardinalities of the first and the second group, respectively. The number of degrees of freedom v can be estimated by using

$$v \approx \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{(\frac{s_1^2}{N_1})^2}{N_1 - 1} + \frac{(\frac{s_2^2}{N_2})^2}{N_2 - 1}}. \tag{2.54}$$

Under the null hypothesis, the statistic t follows Student's t-distribution with v degrees of freedom. The null hypothesis is rejected according to the distribution and selected significance level α. For sufficiently large n, the t-distribution can be satisfactorily approximated by normal distribution. In side-channel analysis, the threshold ± 4.5 or ± 5 for the t-value is often considered [62, 161], which roughly corresponds to significance level $\alpha \leq 10^{-5}$. Rejecting the null hypothesis suggests that the two groups have different means, and therefore an information leakage. Not rejecting the null hypothesis suggests nothing; most importantly, it does not suggest there is no leakage.

The Welch's t-test is a univariate moment-based statistic, similar to statistics used in DPA or CPA attacks (subsection 2.2.1 and subsection 2.2.3). The measurements therefore must be aligned. To evaluate leakage exploitable by higher-order attacks, e.g., when evaluating a higher-order masking scheme, relevant (pre-)processing must be performed [151], similarly to the attacks. This includes a use of either univariate or multivariate combining function as described in subsection 2.6.2.

2.7.2 χ^2 test

Pearson's χ^2 test of independence tests a *null hypothesis* that two or more variables are independent, and is well-suited for leakage assessment [117]. Unlike the t-test, χ^2 is a non-parametric test: instead of statistical moments, whole underlying distributions are considered. In this subsection, the univariate test is described first, as in case of the t-test, i.e., the test is performed at every sampling point independently.

A two-row ($r = 2$) contingency table F is created by using histograms of both groups (assuming aligned histograms, i.e., the same range and width of bins), where the number of columns c corresponds to the number of bins. Columns containing only zeros should be eliminated to decrease number of degrees of freedom. Let $F_{i,j}$ be the frequency of each cell, and N be the number of all measurements. The expected frequency of each cell $E_{i,j}$ is then computed as

$$E_{i,j} = \frac{(\sum_{k=0}^{c-1} F_{i,k}) \cdot (\sum_{k=0}^{r-1} F_{k,j})}{N}, \tag{2.55}$$

the χ^2 statistic x as

$$x = \sum_{i=0}^{r-1} \sum_{j=0}^{c-1} \frac{(F_{i,j} - E_{i,j})^2}{E_{i,j}}, \tag{2.56}$$

and the number of degrees of freedom v as

$$v = (c - 1) \cdot (r - 1). \tag{2.57}$$

Under the null hypothesis, the statistic x follows χ^2 distribution with v degrees of freedom. The null hypothesis is rejected according to the distribution and selected significance level α, similarly to Welch's t-test (subsection 2.7.1). Once again, rejection of the null hypothesis suggests information leakage.

Because χ^2 is a nonparametric test, univariate higher-order leakage is considered inherently. To extend the test to a multivariate case, either the combining function can be utilized (as in case of the t-test), or a multivariate histogram can be used. The χ^2 test also enables more than two groups to be used in the test, and it can be also used in an attack scenario similar to the t-test in DPA [117].

2.7.3 Deep Learning Leakage Assessment

The distinguishability of the two groups can also be successfully tested by using a deep learning-based classifier [114]. Assuming there is exploitable leakage, the classifier should be able to learn distinctive features of measurements in each group.

First, the measurements get standardized by subtracting the mean value and then dividing it by the standard deviation, at every sampling point independently. Henceforth, for the classifier, the measurements are considered multivariate vectors. The data are split into training and evaluating sets. The leakage assessment only examines leakage in measurements used in the training stage.

Under a *null hypothesis* that the classifier did not recognize and learn any features, the number of its correct guesses on the evaluating set should follow binomial distribution with probability $p = \frac{1}{2}$. The null hypothesis is rejected once again according to the distribution and selected significance level.

Deep Learning Leakage Assessment provides a powerful tool thanks to its multivariate nature, ability to identify distinctive features, and its detection sensitivity, which outperforms both Welch's t-test and χ^2 test [114]. It displays similar characteristics as machine learning-based attacks (subsection 2.2.6, subsection 2.3.2).

Chapter 3

Symmetric Cryptography

The work presented in section 3.2 was done in cooperation with Jan Brejník (a master's student) and Stanislav Jeřábek (a Ph.D. student) of Czech Technical University, Josep Balasch of KU Leuven, and Nele Mentens of KU Leuven and Leiden University, as a part of the international project CELSA DRASTIC. The results were presented at the DSD conference [A.2] in 2019 and their extended version was published in the Microprocessors and Microsystems journal [A.3] in 2020.

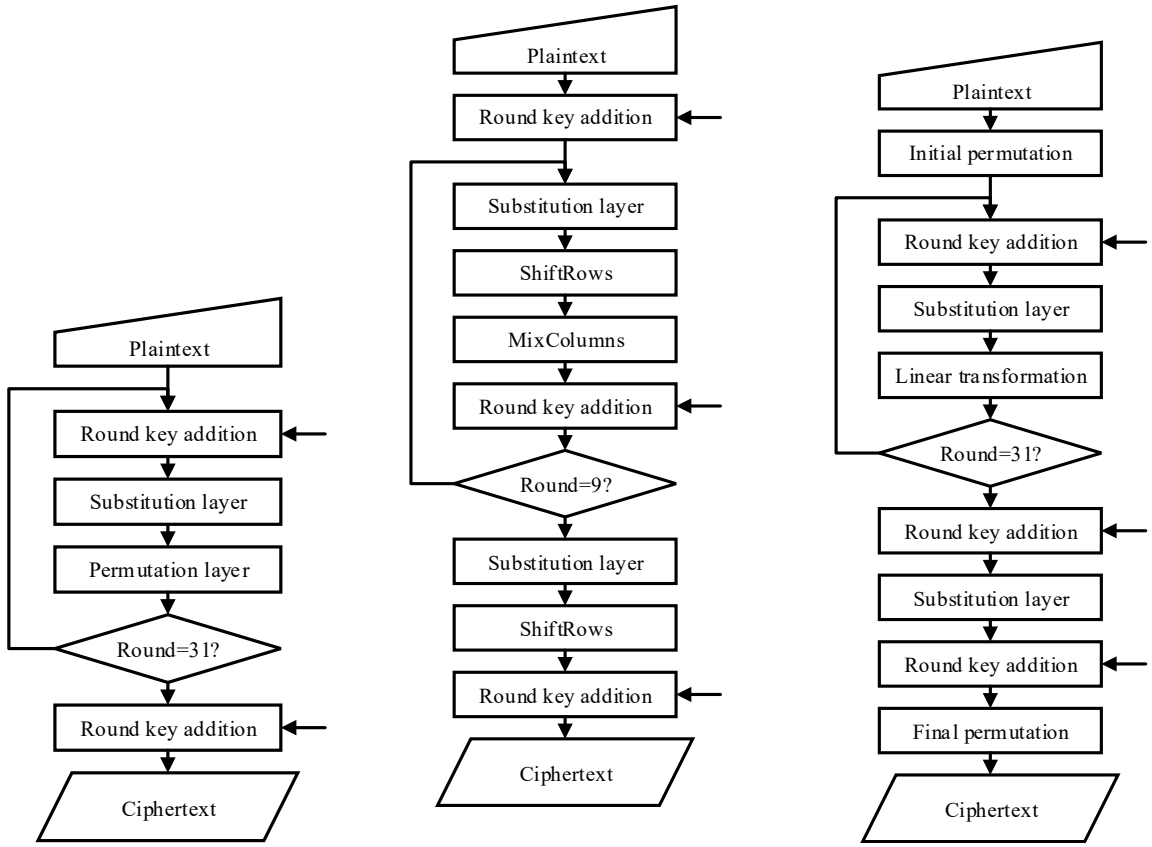
The work in section 3.3 was presented at the DSD conference [A.4] in 2020 and an extended version of the work was published in the Microprocessors and Microsystems journal [A.5] in 2021.

Many different countermeasures have been proposed to prevent side-channel attacks (see section 2.5). This chapter focuses on FPGA implementations of symmetric cryptography, and countermeasures for AES, Serpent and PRESENT are proposed and evaluated.

First, the discussed ciphers are described in section 3.1. Then in section 3.2, a combination of countermeasures previously proposed for PRESENT in [148] is tailored for AES and Serpent ciphers and the secured implementations are evaluated regarding the side-channel leakage and time/area overhead. In section 3.3, a novel approach for securing the symmetric cryptography in FPGAs using high-level synthesis from C is proposed and once again comprehensively evaluated. Finally, a summary of the results is presented in section 3.4.

3.1 Substitution-Permutation Networks

All the ciphers discussed in this chapter are symmetric block ciphers based on iterated substitution-permutation networks (SPN). Plaintext is transformed into ciphertext by iteratively applying specified operations; each iteration is called a *round*. Moreover, every cipher also specifies a method for expanding the secret key into subkeys used in different rounds; the key expansion algorithm is not discussed in detail in this dissertation thesis.



(a) PRESENT encryption. (b) AES/Rijndael encryption. (c) Serpent encryption.

Figure 3.1: Discussed symmetric encryption algorithms (128-bit key variants).

3.1.1 PRESENT

PRESENT by Bogdanov et al. [25] is a lightweight cipher with a block size of 64 bits and possible key sizes of 80 or 128 bits. Figure 3.1a depicts the encryption algorithm. Each round consists of a round key addition (XOR), a non-linear substitution layer (4-bit S-boxes applied 16 times in parallel), and a linear permutation layer. After 31 rounds, the 32nd round key is finally added to produce the ciphertext. The versions with 80-bit and 128-bit keys differ only in the Key Schedule operation.

3.1.2 AES/Rijndael

Rijndael by Daemen et Rijmen [120, 46], also known as Advanced Encryption Standard (AES), is a cipher with a block size of 128 bits, key sizes of 128, 192, or 256 bits and consisting of 10, 12, or 14 rounds (depending on the key length). Figure 3.1b depicts the encryption algorithm. First, the secret key is XORed with the plaintext, followed by round transformations. Each round consists of a substitution layer (8-bit S-boxes applied

16 times in parallel), two linear mixing layers (only one mixing layer in the last round), and an XOR with the round subkey.

3.1.3 Serpent

Serpent by Biham et al. [21] is a cipher with a block size of 128 bits and possible key sizes of 128, 192, or 256 bits, same as Rijndael (Serpent was another AES finalist). Figure 3.1c depicts the encryption algorithm. It consists of 32 rounds, where each round consists of an XOR with the round subkey, a substitution layer (4-bit S-boxes applied 32 times in parallel; 8 different S-boxes to be used in consecutive rounds) and a linear mixing layer (replaced by a second XOR in the last round). Furthermore, the initial permutation is applied prior to round transformations, and the final permutation is applied afterward.

3.2 Combined Countermeasures Utilizing Dynamic Logic Reconfiguration

A combination of countermeasures implemented using dynamic logic reconfiguration is proposed in [148] and evaluated on the lightweight block cipher PRESENT. In this section, the work presented in [148] is extended by using dynamic logic reconfiguration to secure two of the Advanced Encryption Standard (AES) competition finalists, Rijndael and Serpent. The implementations and the non-straightforward way in which the countermeasures in [148] are tailored to AES and Serpent are described. The side-channel leakage and the effectiveness of different countermeasures combinations are evaluated.

In subsection 3.2.1, the concept of dynamic logic reconfiguration and CFGLUTs is described. Then in subsection 3.2.2, the countermeasures which are later implemented are described, and in subsection 3.2.3, the secured cipher design is proposed. The time and area requirements are discussed in subsection 3.2.4 and the side-channel leakage assessment is presented in subsection 3.2.5. The section is summarized in subsection 3.2.7.

3.2.1 Dynamic Logic Reconfiguration using CFGLUTs

Dynamic logic reconfiguration is a concept that allows for efficient on-the-fly modifications of combinational circuit behavior in both ASIC [7, 47] and FPGA devices. In FPGAs, combinational circuits are typically implemented using Look-Up Tables (LUTs), i.e., configurable primitives which store truth tables of k-input Boolean functions $f : \mathbb{B}^k \rightarrow \mathbb{B}$. Dynamic logic reconfiguration allows for the run-time alteration of the circuit behavior by modifying the content of specific look-up tables, while leaving the routing intact. The reconfiguration of LUTs is done from within the chip itself and can be achieved, e.g., by using a shift register (allowing for serial programming) and a cascade of addressing multiplexers. This concept is demonstrated in Figure 3.2.

In Xilinx FPGAs [180], this functionality is provided by k-input Configurable Look-Up Tables (CFGLUTs) with a serial configuration input and output (allowing to connect CFG-

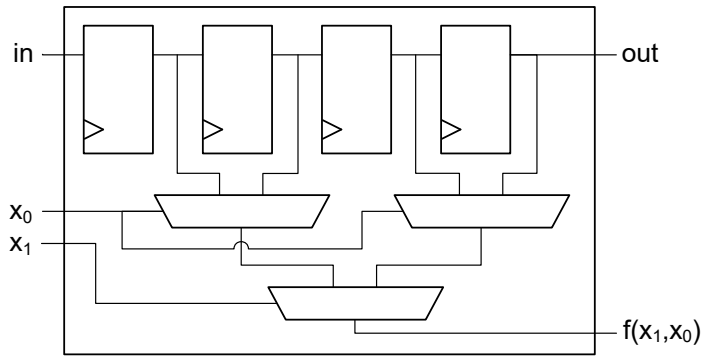


Figure 3.2: Example of a 2-input reconfigurable look-up table with serial programming I/O.

LUTs in separate configuration chains). In Xilinx Spartan-6 FPGAs, 5-input CFGLUTs are available.

In order to implement dynamically reconfigurable Boolean functions $f : \mathbb{B}^n \to \mathbb{B}$, where $n > k$, multiple k-input CFGLUTs are required in combination with addressing multiplexers (using Boole's expansion, also referred to as the Shannon expansion [30]). Specifically, to implement an n-input function using k-input CFGLUTs and 2-to-1 multiplexers, we need 2^{n-k} CFGLUTs and $2^{n-k} - 1$ multiplexers.

Multiple-output Boolean functions $f : \mathbb{B}^n \to \mathbb{B}^m$ can be trivially implemented as m single-output Boolean functions $f_i : \mathbb{B}^n \to \mathbb{B}$.

3.2.2 Countermeasures Combination

To protect AES and Serpent, the countermeasures proposed (and evaluated on PRESENT) by Sasdrich et al. in [148] are implemented. In this subsection, these countermeasures are briefly described.

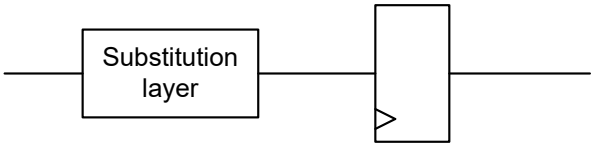
3.2.2.1 S-box Decomposition

Since information leakage often occurs based on changing values in registers, and since the output of the non-linear substitution layer is a frequent target of side-channel attacks, the S-box decomposition countermeasure is based on avoiding the storage of the S-box outputs into such registers. This is done by decomposing the S-box into two bijections R_1, R_2, where

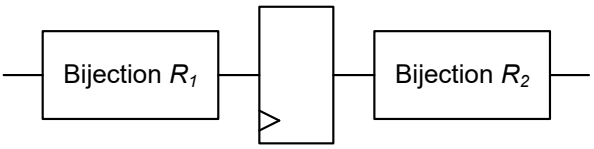
$$\text{S-box}(x) = R_2(R_1(x)), \tag{3.1}$$

and by placing the register in between the two bijections. The decomposition is demonstrated in Figure 3.3. The number of possible n-bit bijections for R_1 is equal to $(2^n)!$. For each option, a bijection R_2 can be found such that Equation 3.1 holds.

Thanks to dynamic logic reconfiguration, different bijections R_1, R_2 can easily be used for every encryption. Starting with R_1 being an identity and R_2 being the actual S-box



(a) Unprotected substitution layer.



(b) Substitution layer decomposed into two bijections.

Figure 3.3: S-box Decomposition.

(or vice versa), the bijections for the next encryption are computed by randomly selecting pair(s) of elements in the R_1 mapping, swapping them, and recomputing R_2 accordingly.

3.2.2.2 Boolean Masking

In order to randomize intermediate values, a random mask is added (XORed) to the data prior to encryption, and subtracted (i.e. once again XORed) after the encryption. For the cipher to produce valid results working with masked data, various alterations must be done.

Boolean masking can be combined with the previously mentioned bijective S-box decomposition and can once again take advantage of dynamic logic reconfiguration. Two different random masks m_1, m_2 are used for every encryption: mask m_1 is used outside the decomposed S-box, and mask m_2 is used inside of it. If the substitution layer were the only layer in the round, the previously mentioned bijections R_1, R_2 would get adjusted as follows:

$$R'_1(x) = R_1(x \oplus m_1) \oplus m_2, \tag{3.2}$$
$$R'_2(x) = R_2(x \oplus m_2) \oplus m_1. \tag{3.3}$$

The function R'_1 first subtracts/removes mask m_1, then performs the R_1 bijection mapping, and finally masks this value using m_2. The output of this function is stored in the register. Analogically, the function R'_2 subtracts the mask m_2, does the R_2 mapping, and masks the result using m_1. This is demonstrated in Figure 3.4. This way, the same CFGLUTs can be used for both the S-box decomposition and the masking, saving both area and reconfiguration time.

However, to deal with the linear transformation layers, further alterations to the R'_1, R'_2 bijections need to be done. A very similar concept was already described in subsubsec-

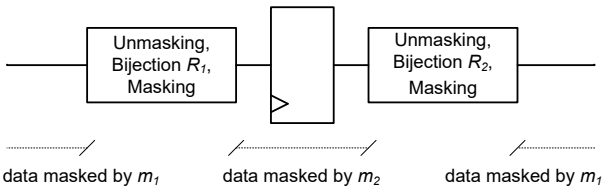


Figure 3.4: S-box Decomposition + Masking: all the three operations (unmasking, bijection and masking) are performed as a single table lookup, therefore unmasked data does not appear on any wires at any time.

tion 2.5.3.1. We can exploit one of these two facts:

$$f(x) = f(x \oplus f^{-1}(m)) \oplus m, \tag{3.4}$$
$$f(x) = f(x \oplus m) \oplus f(m), \tag{3.5}$$

which both hold when f is a linear mapping. These give us two different and fairly straightforward approaches to take linear transformations f into account.

One option is to alter R'_2 function in terms of Equation 3.4 so that m_1 processed by the inverse transformation is used to mask the data, allowing to subtract m_1 in R'_1:

$$R'_1(x) = R_1(x \oplus m_1) \oplus m_2, \tag{3.6}$$
$$R'_2(x) = R_2(x \oplus m_2) \oplus f^{-1}(m_1). \tag{3.7}$$

The second option is to use m_1 for masking in R'_2, and to alter R'_1 according to Equation 3.5, so that m_1 processed by the linear transformation gets subtracted:

$$R'_1(x) = R_1(x \oplus f(m_1)) \oplus m_2, \tag{3.8}$$
$$R'_2(x) = R_2(x \oplus m_2) \oplus m_1. \tag{3.9}$$

Notice that further alterations may be required for the first and the last round, depending on the selected approach.

The last obstacle is the subkey XOR layer, which can be considered an affine transformation. Suppose we have a vector x, which gets XORed with the subkey: $x \oplus k$. Suppose we process masked data the same way: $(x \oplus m) \oplus k$, then by subtracting the mask m with no alterations we have:

$$((x \oplus m) \oplus k) \oplus m = x \oplus k. \tag{3.10}$$

Therefore, no further alterations need to be done to take the XOR layer into account.

3.2.2.3 Register Precharge

Because the same masks are used for the whole encryption (i.e., for every round), the leakage occurs in the register, since

$$\mathrm{HD}(x \oplus m, y \oplus m) = \mathrm{HD}(x, y), \tag{3.11}$$

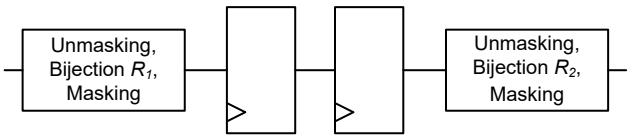


Figure 3.5: S-box Decomposition + Masking + Register Precharge.

where $\mathrm{HD}(x, y)$ denotes the Hamming distance between x and y. To avoid this leakage, the register is duplicated, as shown in Figure 3.5, and the processed data are interleaved with random data. This technique avoids leakage; however, it reduces the throughput of the circuit when it is implemented using an architecture that is not fully unrolled.

3.2.3 Proposed Secure Cipher Design

In this subsection, the specifics of both AES/Rijndael and Serpent ciphers are described and a manner in which these ciphers can be secured against side-channel attacks using the countermeasures described in subsection 3.2.2 is proposed.

In order for our implementations to fit into a Xilinx Spartan-6 FPGA device, we take into account that CFGLUTs with at most 5 input bits are available. When a platform with smaller CFGLUTs is available, the dynamic logic reconfiguration method can be implemented using the approach described in subsection 3.2.1.

3.2.3.1 AES/Rijndael

Rijndael employs an 8×8 S-box, which can be considered as a function $\text{S-box}_{\text{Rijndael}} : \mathbb{B}^8 \rightarrow \mathbb{B}^8$. Therefore, to implement the Rijndael S-box using reconfigurable logic, $8 \cdot 2^{8-5} = 64$ (5-input) CFGLUTs and $8 \cdot (2^{8-5} - 1) = 56$ (2-to-1) multiplexers are necessary. Moreover, the S-box decomposition countermeasure suggests the S-box to be split into two bijections $R_1, R_2 : \mathbb{B}^8 \rightarrow \mathbb{B}^8$, which doubles the amount of CFGLUTs and multiplexers in the secured version. Since the Rijndael algorithm applies 16 S-boxes in parallel, this brings the total count up to 2048 (5-input) CFGLUTs and 1792 (2-to-1) multiplexers.

The decomposition into two bijections is done in a similar fashion as described in subsection 3.2.2, with the round register being placed in between the two bijections. For the AES algorithm, we have decided to swap eight pairs of elements in the R_1 bijection after every encryption (in contrast to the PRESENT 4-bit S-box decomposition in [148], where only a single pair gets swapped).

To implement the Boolean masking countermeasure as described in subsection 3.2.2, bijections R'_1, R'_2 (i.e. the decomposed S-box combined with masking) must be altered. We choose the option where R'_2 adds the mask m_1 and R'_1 subtracts m_1 processed by the linear transformations (see Equation 3.8)):

$$R'_1(x) = R_1(x \oplus \text{MixColumns}(\text{ShiftRows}(m_1))) \oplus m_2, \tag{3.12}$$

$$R'_2(x) = R_2(x \oplus m_2) \oplus m_1. \tag{3.13}$$

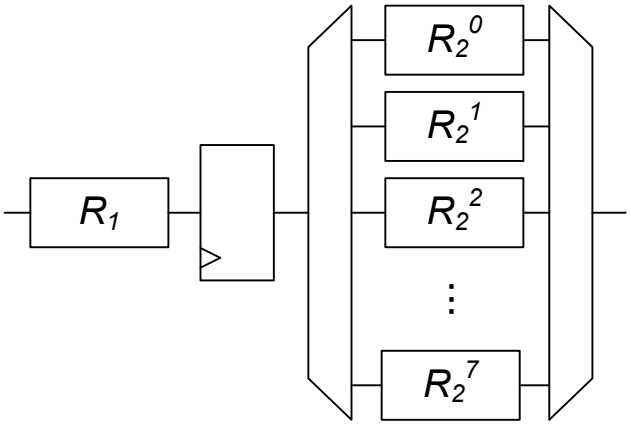


Figure 3.6: Serpent S-boxes decomposition. Notice the demultiplexer, which is necessary to prevent glitches.

Note that the data are masked by m_1 in the second bijection R_2 and that this mask is subtracted in the following round. Therefore prior to the first round, the input data must be masked properly. Also, the last round of Rijndael omits the MixColumns operation, so additional unmasking of the output must be done with this in mind.

The implementation of the register precharge requires the register to be duplicated and the controller to be adjusted appropriately, such that the processed data are interleaved with random data.

3.2.3.2 Serpent

Unlike Rijndael or PRESENT, Serpent defines eight different 4×4 S-boxes. Each S-box is used in a different round. One way to implement the S-box decomposition is to decompose each of these S-boxes into two bijections, resulting in 16 bijections in total. We have decided for an approach where the first bijection R_1 is shared among all S-boxes, while the other eight bijections R_2^i, $i \in \{0, .., 7\}$, implement the eight S-boxes, with the correct output being selected by a multiplexer. The eight decomposed Serpent S-boxes are depicted in Figure 3.6. Notice the **demultiplexer**, which selects the right R_2^i bijection, while the other bijections are fed with zeroes. This demultiplexer is necessary to prevent glitches that lead to information leakage. Since the Serpent S-boxes implement the functions $\text{S-box}^{\text{i}}_{\text{Serpent}} : \mathbb{B}^4 \rightarrow \mathbb{B}^4$, only four CFGLUTs are necessary to implement the bijection. Given the selected architecture, $4 + 8 \cdot 4 = 36$ CFGLUTs are required to decompose all eight S-boxes. Since the S-box is applied 32 times in parallel, this results in 1152 CFGLUTs in total.

Boolean masking is implemented similarly to the Rijndael algorithm, with m_1, processed by the linear transformation, being subtracted in the R_1' bijection (see Equation 3.8)).

Table 3.1: Latency and Area Utilization.

Implementation	Area		Latency (clock cycles)		
	Memory (FFs)	Logic (LUTs)	Encryption	Extra	Total
Unprotected AES/Rijndael, LUT-based S-Boxes	278	1,304	10	0	10
Protected AES/Rijndael, 2 CFGLUT chains	1,073	2,652	20	4,122	4,142
Protected AES/Rijndael, 32 CFGLUT chains	3,229	7,234	20	282	302
Unprotected Serpent, LUT-based S-Boxes	430	1,660	32	0	32
Protected Serpent, 9 CFGLUT chains	2,945	5,696	64	538	602
Protected Serpent, 144 CFGLUT chains	4,441	9,471	64	58	122
Protected Serpent, 288 CFGLUT chains	6,040	13,211	64	42	106

Suppose the Serpent linear transformation is $L_{Serpent}$, then:

$$R'_1(x) = R_1(x \oplus \mathrm{L_{Serpent}}(m_1)) \oplus m_2, \tag{3.14}$$

$$R'_2(x) = R_2(x \oplus m_2) \oplus m_1. \tag{3.15}$$

Regarding the first round, similarly to the Rijndael approach, appropriate initial masking of the input data must be performed first. Also, there is no linear transformation in the last round; therefore, the unprocessed mask m_1 gets subtracted during the final unmasking.

Register precharge is once again implemented simply by duplicating the round register and altering the controller appropriately to interleave the processed data with random data.

3.2.4 Latency and Area Utilization

For every encryption, new bijections are generated (as described in subsection 3.2.2), as well as new masks m_1, m_2. This requires the CFGLUTs configurations to be computed and loaded prior to every encryption. The reconfiguration of all CFGLUTs can be done using different levels of parallelism (the CFGLUTs "programming" I/O can be variously chained, given its shift register nature). The serial reconfiguration of n-input CFGLUT requires 2^n cycles, therefore selected reconfiguration strategy has a direct impact on the overall latency, as well as on the area utilization.

Table 3.1 presents a comparison of the latency and the area of both unprotected and protected AES/Rijndael and Serpent encryption implementations. The Flip-Flop (FF) and the Look-Up Table (LUT) counts are Xilinx ISE post-synthesis statistics for Xilinx Spartan-6 FPGA. The encryption latency of protected implementations is double due to the register precharge. The extra latency is caused mostly by the CFGLUT serial programming, and it can be reduced by using several parallel configuration chains, at the expense of the area.

3.2.5 Side-Channel Leakage Evaluation

In this subsection, we present our experimental setup and a leakage methodology used to evaluate all combinations of previously described countermeasures.

We choose the Sakura-G board [65] with a Xilinx Spartan-6 FPGA as our evaluation platform. AES/Rijndael and Serpent VHDL implementations with a 128-bit key are evaluated. The power traces evaluated in subsubsection 3.2.5.1 and subsubsection 3.2.5.2 are measured using a PicoScope 6404D oscilloscope and the power traces evaluated in subsubsection 3.2.5.3 are measured using a Tektronix DPO 7254 oscilloscope. The current consumption of the FPGA core is measured as a voltage drop across a shunt resistor in the VCCINT path of the FPGA. The voltage drop is furthermore amplified using a built-in preamplifier before sampling by the oscilloscope. The sample rate used for all the measurements is 625 MS/sec.

3.2.5.1 First-Order Test Vector Leakage Assessment

Leakage is evaluated using the non-specific univariate first-order Welch's t-test, as described in section 2.7. This evaluation method consists of two phases. In the active phase, power traces are collected, each trace measured while encrypting either a random or a (preselected) constant plaintext, resulting in two sets of power traces. In the analytical phase of the evaluation, Welch's t-test statistic is computed independently at each time sample. The Welch's t-test statistic examines the null hypothesis of equal population means (where one population consists of random plaintext measurements and the other population consists of constant plaintext measurements). In our case, the null hypothesis can be formulated in the sense that the two populations are not distinguishable by their sample means, which means that the sample means are not data-dependent. The null hypothesis gets rejected for high values of $|t|$, the threshold is usually set around 4.5 or 5.

The necessary random data (random pairs to be swapped in the bijection, random masks, register precharge with random values) are generated externally and sent to the cryptographic device alongside the plaintext. This approach allows us to enable or disable specific countermeasures easily.

We evaluate every possible combination of the proposed countermeasures:

(a) Unprotected

(b) Register Precharge

(c) Masking

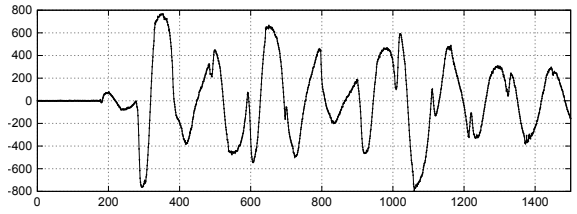
(d) Masking + Register Precharge

(e) S-box Decomposition

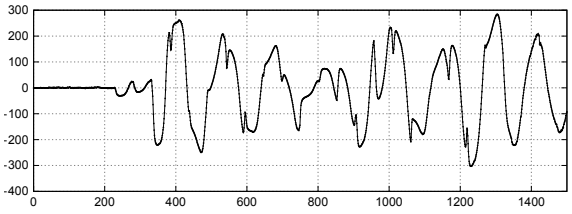
(f) S-box Decomposition + Register Precharge

(g) S-box Decomposition + Masking

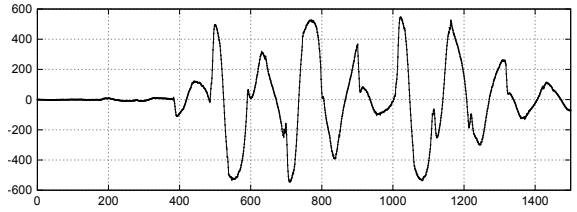
(h) S-box Decomposition + Masking + Register Precharge



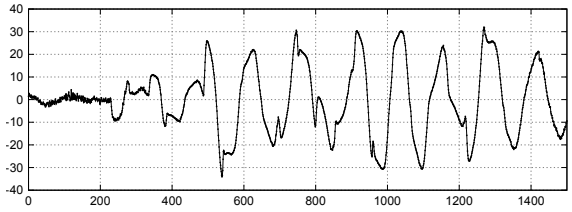
(a) Unprotected.



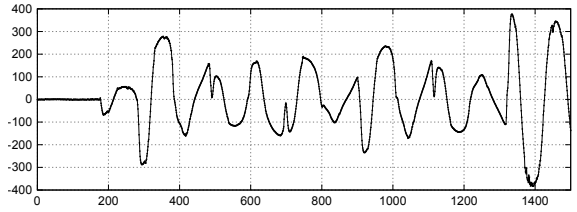
(b) Register Precharge.



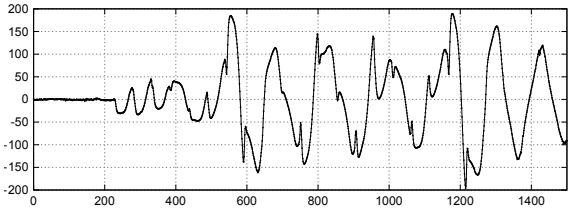
(c) Masking.



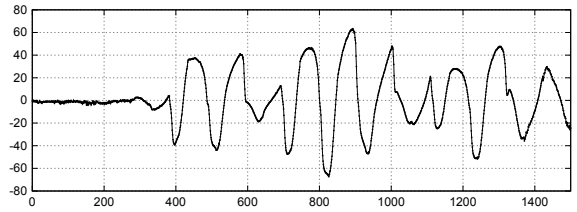
(d) Masking + Register Precharge.



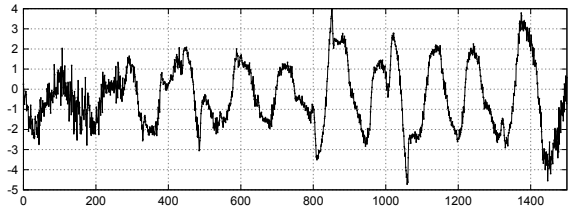
(e) S-box Decomposition.



(f) S-box Decomposition + Register Precharge.

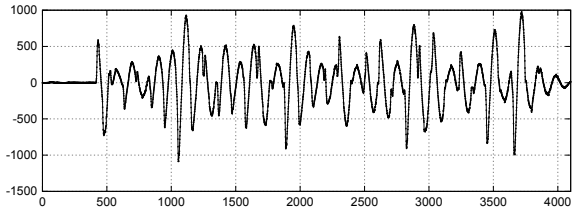


(g) S-box Decomposition + Masking.

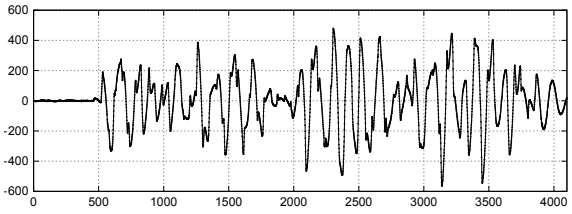


(h) S-box Decomposition + Masking + Register Precharge.

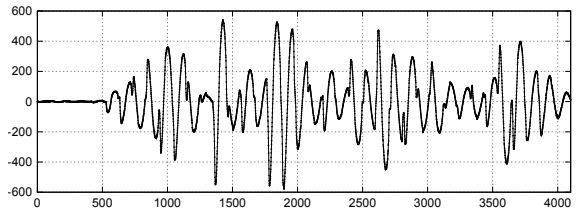
Figure 3.7: Results of the AES/Rijndael t-test, where the t-value is shown on the vertical axis and the time samples during encryption are shown on the horizontal axis.



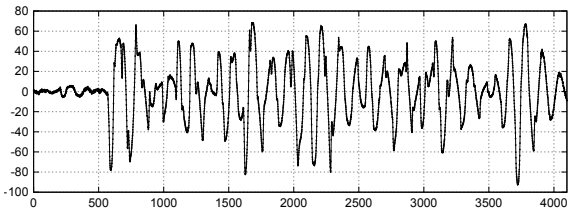
(a) Unprotected.



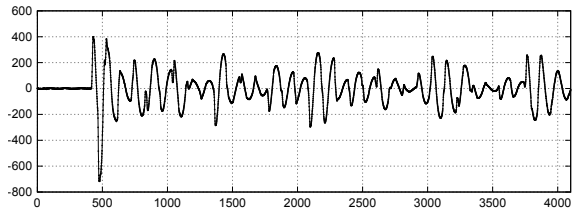
(b) Register Precharge.



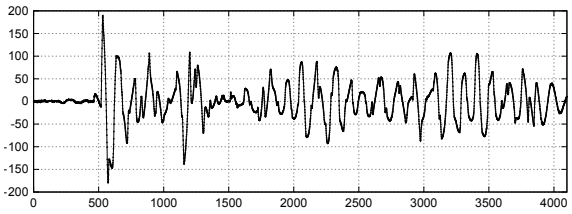
(c) Masking.



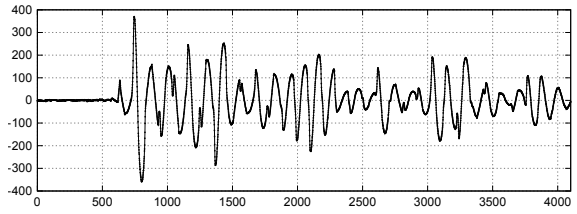
(d) Masking + Register Precharge.



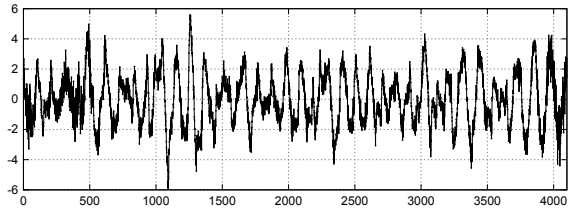
(e) S-box Decomposition.



(f) S-box Decomposition + Register Precharge.



(g) S-box Decomposition + Masking.



(h) S-box Decomposition + Masking + Register Precharge.

Figure 3.8: Results of the Serpent t-test, where the t-value is shown on the vertical axis and the time samples during encryption are shown on the horizontal axis.

For every combination, one million power traces are measured and processed using a non-specific first-order t-test, as described earlier. Figure 3.7 depicts the t-values during the AES encryption and Figure 3.8 depicts the t-values during the Serpent encryption. The sensitive information leakage is the most prominent for the unprotected versions, as expected.

It is also visible that different countermeasures and their combinations have various influence on the significance of the detected leakage. Figures 3.7c and 3.8c show that a countermeasure based on masking protects solely the first round of the cipher, while, starting from the second round, the leakage is comparable to the unprotected version (cf. Figures 3.7a and 3.8a). Figures 3.7d and 3.8d suggest that masking becomes more effective in combination with register precharge (which is expected, as discussed in subsection 3.2.2).

Figures 3.7h and 3.8h show results with all three countermeasures combined. As can be seen, no significant first-order leakage is detected when evaluating these fully protected implementations. However, the used Test Vector Leakage Assessment (TVLA) methodology is merely a first step in the evaluation of a side-channel security of the implementations and the results do not provide any guarantee of a security level [161]. This is not only because of a high risk of both false positives and false negatives, but also because only univariate statistics is considered in this methodology.

3.2.5.2 Second-Order Test Vector Leakage Assessment

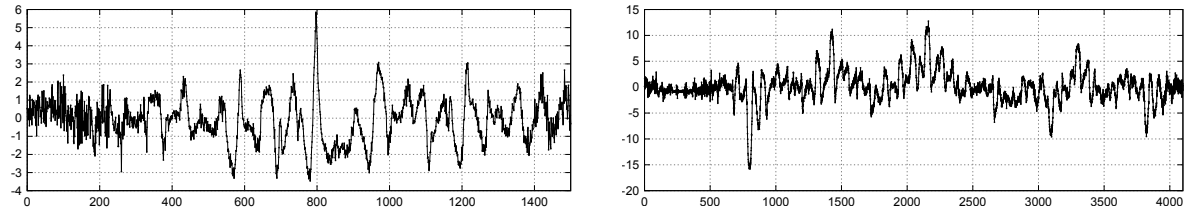
Protected implementations which make use of Boolean masking (i.e., splitting a working variable into d shares) are typically vulnerable to higher-order DPA attacks [110, 39, 134], i.e., attacks which exploit leakage from several variable shares, either by combining multiple time samples together (multivariate), or by analyzing higher statistical moments at a single time sample (univariate). Since a first-order masking scheme is used to protect our implementations, we assume them to be vulnerable against second-order DPA.

We evaluate the second-order leakage of our implementations using Welch's t-test, similar to the first-order leakage evaluation in subsubsection 3.2.5.1. The first phase of the methodology stays the same – therefore, we can use the same sets of power traces obtained for the first-order analysis. To analyze the second statistical moment, the power traces are preprocessed, at each time sample independently, by making every sample mean-free squared:

$$x' = (x - \bar{X})^2, \tag{3.16}$$

where $\bar{X}$ is sample mean at a given time sample. Then the Welch's t-test statistic is computed, same as in case of the first-order leakage evaluation.

We evaluate one million previously captured power traces with all the proposed countermeasures enabled (S-box Decomposition + Masking + Register Precharge). Figure 3.9 depicts (univariate) second-order t-values during AES and Serpent encryption. For AES, there is a single peak reaching as high as 6 halfway the encryption, as can be seen in Figure 3.9a. Second-order leakage is more prominent during Serpent encryption, as can be seen in Figure 3.9b, where the absolute t-value reaches as high as 15.



(a) AES/Rijndael second-order t-test. (b) Serpent second-order t-test.

Figure 3.9: Results of the univariate second-order t-test with all the countermeasures enabled (S-box Decomposition + Masking + Register Precharge), where the t-value is shown on the vertical axis and the time samples during encryption are shown on the horizontal axis.

It is fair to assume that with more than one million power traces available, the second-order leakage would get more prominent and easier to detect. As shown in [39], the amount of power traces required to successfully mount a higher-order side-channel attack increases exponentially with the masking order.

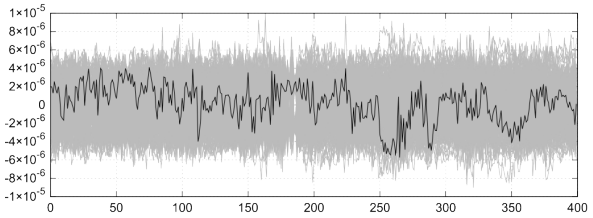
3.2.5.3 Second-Order Attacks

To provide more confidence about the AES/Rijndael implementation resilience, we attempt at breaking the fully protected implementation using second-order attacks [110]. The measured power traces are first preprocessed in the same fashion as in the case of the univariate second-order leakage assessment, as described in subsubsection 3.2.5.2. Afterward, we perform the DPA attack [83] (using t-test distinguisher) and the CPA attack [29], targeting the first and the last cipher round, and considering both Hamming weight and Hamming distance leakage.

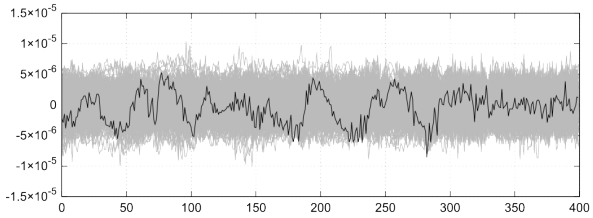
First, we target the AES S-box output in the first round, i.e., $s_i = \text{S-box}(x_i \oplus \hat{k}_i)$ for some index $1 \leq i \leq 16$, key hypothesis $\hat{k}_i \in [0, 255]$ and plaintext byte x_i. Second, we target the AES S-Box input in the last round by predicting values $s_i = \text{S-box}^{-1}(y_i \oplus \hat{k}_i)$ for some index $1 \leq i \leq 16$, key hypothesis $\hat{k}_i \in [0, 255]$ and ciphertext byte ct_i. These predictions are used directly when assuming the Hamming weight leakage. For the Hamming distance leakage model, these predictions are furthermore XORed with the corresponding input/output bytes.

We mount the attacks on a set of 1.25 million power traces. The results of the DPA attack (using t-test distinguisher) are shown in Figure 3.10. Figures 3.10a and 3.10c show the case when targeting S-box output in the first round, while Figures 3.10b and 3.10d show the case when targeting the S-box input in the last round. In all cases, the correct key byte value (in black) is not distinguishable from the other candidates (in grey), so the attack fails in recovering the key.

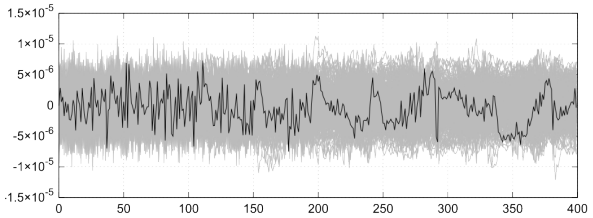
Running the CPA attack yields similar results, as shown in Figures 3.11a and 3.11c when targeting the first round, and in Figures 3.11b and 3.11d when targeting the last round.



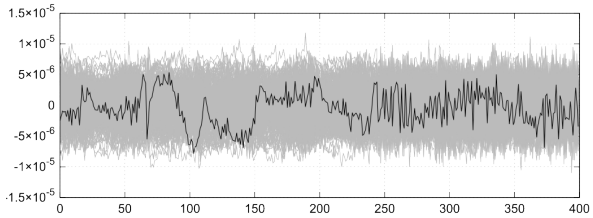
(a) AES/Rijndael, Hamming weight DPA (t-test) first round.



(b) AES/Rijndael, Hamming weight DPA (t-test) last round.



(c) AES/Rijndael, Hamming distance DPA (t-test) first round.



(d) AES/Rijndael, Hamming distance DPA (t-test) last round.

Figure 3.10: Second-order DPA attack (using t-test distinguisher) on first and last round AES/Rijndael subkey (byte #1), where the t-value is shown on the vertical axis and the time samples during the relevant encryption rounds are shown on the horizontal axis.

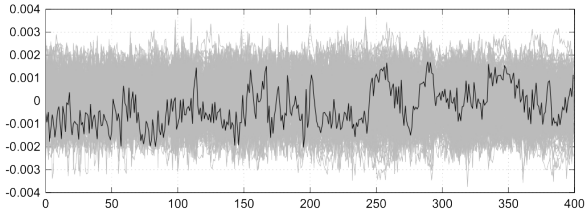
Note that for these experiments, we process only samples in the interval corresponding to the first (resp. last) round, rather than the whole encryption.

3.2.6 Further Experiments

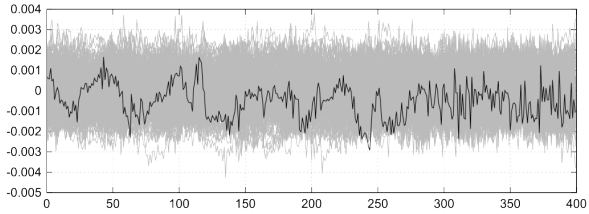
In this subsection, we present additional experiments regarding the proposed countermeasures and the cipher design, and we summarize our results.

3.2.6.1 Importance of Mask m_2

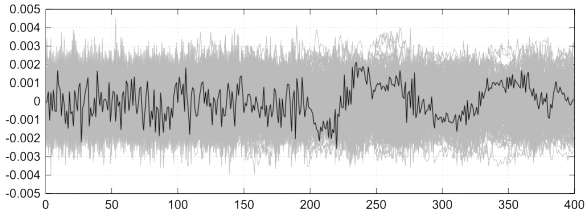
The value stored in the round register is protected by both S-box Decomposition and Boolean Masking (using mask m_2, see Figure 3.4). Considering the CFGLUT based architecture, this masking does not require any extra resources. However, to provide more insight about the necessity of masking inside the decomposed S-box, we have measured one million power traces without it (fully protected encryption, except that mask m_2 is set to zeroes), and performed the test vector leakage assessment as described earlier. In the case of AES/Rijndael, both first-order and second-order t-tests turn out very similar to the previously presented results. However, in the case of Serpent without the m_2 mask, we have encountered worsening, where the first-order t-test values reach as high as 10 (second-order t-test, however, still yields results similar to those presented earlier).



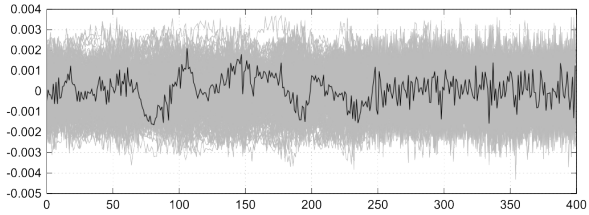
(a) AES/Rijndael, Hamming weight CPA (Pearson) first round.



(b) AES/Rijndael, Hamming weight CPA (Pearson) last round.



(c) AES/Rijndael, Hamming distance CPA (Pearson) first round.



(d) AES/Rijndael, Hamming distance CPA (Pearson) last round.

Figure 3.11: Second-order CPA attack on first and last round AES/Rijndael subkey (byte #1), where the correlation coefficient is shown on the vertical axis and the time samples during the relevant encryption rounds are shown on the horizontal axis.

3.2.6.2 Necessary Randomness

The serial configuration I/O of CFGLUTs allows for various reconfiguration strategies (as mentioned in subsection 3.2.4). Furthermore, a particular number of pairs get swapped when recomputing the S-box decompositions (as mentioned in subsubsection 3.2.2.1). Selected strategy may significantly affect the amount of resources necessary. In this experiment, we compare implementations where:

- either all S-boxes are reconfigured in parallel based on same random data, or every S-box is reconfigured separately based on its own random data,
- the decomposed S-boxes get modified by swapping either one or eight pairs of elements in the bijection mapping.

All four combinations, for both ciphers, perform equally in test vector leakage assessment based on 300,000 measured power traces.

3.2.7 Summary

In this section, we described and evaluated side-channel attack protected AES and Serpent implementations, which are based on an approach demonstrated by Sasdrich et al. [148] for

the PRESENT cipher. These implementations utilize dynamic logic reconfiguration, which can be easily deployed in both FPGA and ASIC designs. We describe a method by means of which a generic substitution-permutation network can be protected against side-channel attacks, and we tailor the approach to a Xilinx Spartan-6 FPGA for the protection of both AES and Serpent.

We demonstrate the effectiveness of the implemented countermeasures by evaluating the side-channel leakage using Welch's t-test, with different combinations of countermeasures in place. We did not detect any significant first-order leakage from the protected versions of both AES and Serpent encryption implementations using one million power traces. Using the same power traces, we detected apparent second-order leakage from Serpent encryption, while AES encryption second-order leakage is barely detectable. Furthermore, to provide more confidence about the implementation resilience, we attempted at breaking the protected AES implementation using second-order DPA and CPA attacks targeting both first and last round. All these attacks fail with 1.25 million power traces available.

3.3 High-Level Synthesis of Masking Countermeasure

Traditional and widely adopted hardware design methodology is based on the Register-Transfer Level (RTL) approach, i.e., modeling the digital system by registers, data signals, and logical operations between them. Such a system is usually described using concurrent languages like VHDL or Verilog. Emerging trends, however, lean to a system-level approach [69, 182], which brings many advancements, including more abstract design flow, optimal hardware/software co-design, simplified verification, validation and co-simulations, and many more. The system-level description may end up compiled into machine code or synthesized into hardware implementation. High-level synthesis is a process of translating the C, C++, or SystemC algorithmic description to a clock-timed RTL or gate-level model, allowing further mapping onto final architecture/technology, e.g., an FPGA.

The high-level synthesis approach draws attention lately, partially thanks to machine learning acceleration frameworks [51, 123], and also thanks to FPGA-based cloud platforms such as Amazon EC2 F1 or Microsoft Project Catapult, which make acceleration using FPGAs easily accessible to a broad community of users. However, usage of multi-tenant FPGA chips, e.g., running multiple cloud computing instances in the same chip, introduces novel security challenges. In such a scenario, the ability to perform side-channel attacks from the inside of the chip [150, 183, 60] allows for remote side-channel attacks, and calls for novel countermeasures such as active fences hiding [88].

In this section, we propose a novel dynamic logic reconfiguration-based masking countermeasure approach to secure a substitution-permutation network-based encryption, described at the system level in C language, against side-channel attacks. We synthesize our implementations of PRESENT, AES/Rijndael, and Serpent using high-level synthesis and we compare the area and time performance. We further evaluate the proposed countermeasures using both specific and non-specific leakage assessment methodologies. We show that in the case of PRESENT, our protected high-level implementation successfully ob-

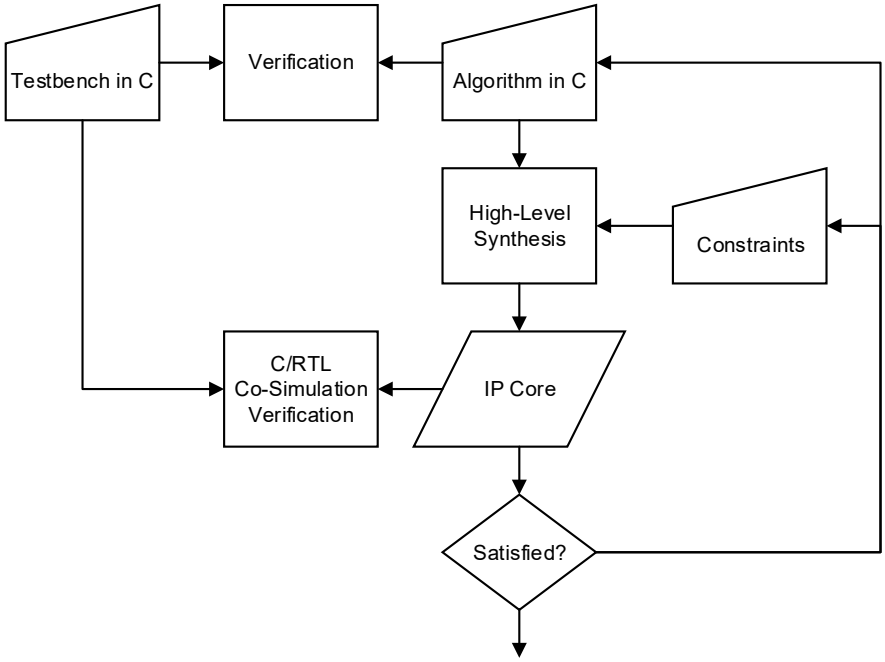


Figure 3.12: Example of a design flow using high-level synthesis.

scures the first-order side-channel leakage while managing to keep reasonable area and time overhead. We further discuss the results of AES/Rijndael and Serpent, and we identify the limitations of the system-level approach and the high-level synthesis.

3.3.1 FPGA Design using High-Level Synthesis

In this work, we choose Xilinx Vivado Design Suite as our high-level synthesis toolchain [181], which provides synthesis from C, C++, or SystemC code to an IP core. For the code to be synthesizable, only a subset of the C language must be used (similar to the VHDL language in case of the RTL design), with most of the basic constructs available, such as variables, arrays, loops, or conditions. Extra features like arbitrary-width integer types allow for better optimization. Furthermore, additional constraints and optimizations may be set using pragma directives, e.g., advising loops to be pipelined or unrolled, partitioning of arrays, selecting memory primitives, and more. Designers may also define the resulting I/O protocol, including various handshakes or bus interfaces like AXI. The final implementation is synthesized using a selected strategy based on the constraints and metrics such as area, throughput, and latency, allowing to explore possible design space quickly. Figure 3.12 depicts the workflow using high-level synthesis.

There are limited research resources regarding usage of high-level synthesis in the cryptographic domain [69, 81], dealing mostly with time and area performance. The effects of various high-level synthesis parameters on the power side-channel are discussed in [182]. Techniques reducing side-channel vulnerabilities such as imbalanced code branches are pre-

sented in [85]. Information flow enforcement [75, 76] introduces prevention against threats such as exploitable I/O and buffer errors, unprivileged access, or timing attacks.

3.3.2 Alternating Masks Scheme

We propose implementation of the precomputed masked S-boxes as described in subsubsection 2.5.3.1 with some further alterations. The leakage on the working register is proportional to the Hamming distance between the old and a new value, which is, by definition, independent of any mask XORed to both values. To solve this issue, we propose a usage of two independent random masks, m_1, m_2, and we alternate these masks in consecutive rounds. This successfully masks the Hamming distance leakage, since

$$HD(x \oplus m_1, y \oplus m_2) = HW(x \oplus y \oplus m_1 \oplus m_2). \tag{3.17}$$

This approach is only sufficient if the implemented encryption algorithm operates at most one round per clock cycle, or, if partially unrolled, an odd number of rounds per clock cycle.

To implement this masking, we propose using two different altered substitution layers, S'_0 and S'_1, to be used alternatingly in odd rounds:

$$S'_1(state') := S(state' \oplus m_1) \oplus P^{-1}(m_2), \tag{3.18}$$

and even rounds:

$$S'_0(state') := S(state' \oplus m_2) \oplus P^{-1}(m_1), \tag{3.19}$$

where P is the PRESENT permutation.

Assuming the PRESENT encryption with 31 rounds, as described in section 3.1, this results in one mask being used for masking the plaintext, with the output ciphertext being masked using the other mask. Two sets of masked 4-bit S-boxes are implemented using dynamic logic reconfiguration.

For AES/Rijndael encryption with 10, 12, or 14 rounds, one mask is being used for both plaintext and ciphertext, with the other one being used internally only. Two sets of masked 8-bit S-boxes are implemented.

Serpent encryption with 32 rounds specifies eight different S-boxes to be used in consecutive rounds. This allows for more efficient implementation of the proposed masking scheme, since only a single set of eight different four-bit S-boxes needs to be implemented, and no replication is necessary (unlike AES or PRESENT). The total number of implemented S-boxes is summarized in Table 3.2.

Two different masks are generated for every encryption. This masking scheme can be easily described purely algorithmically using C language and, as shown further, is well suitable for FPGA high-level synthesis.

3.3.3 Proposed Secure Cipher Design

We implemented the masking scheme proposed in subsection 3.3.2 in C for Xilinx Vivado High-Level Synthesis. For simplicity, we only describe the implementation of the

Table 3.2: Total S-box count.

Algorithm	Unprotected	Alternating Masks Scheme
PRESENT (4-bit S-box)	16	32
AES/Rijndael (8-bit S-box)	16	32
Serpent (4-bit S-box)	256	256

Listing 3.1: Top-level function.

```
present_block_t encrypt(present_block_t plaintext, present_key_t key,
                        present_block_t maskIn, present_block_t maskOut) {
    reconfigure(maskIn, maskOut);
    present_block_t state = plaintext;
    present_key_t roundKey = key;
    state = addRoundKey(state, roundKey);
    for(int round = 1; round < 32; round++) {
        #pragma HLS pipeline
        roundKey = updateKey(roundKey, round);
        state = sLayer(state, round);
        state = pLayer(state);
        state = addRoundKey(state, roundKey);
    }
    return state;
}
```

Listing 3.2: Substitution layer.

```
static present_sbox_t sBoxMasked[2][SBOX_COUNT][SBOX_RANGE];

present_block_t sLayer(present_block_t state, present_round_idx_t round) {
    present_block_t newState;
    for(int i=0; i<16; i++){
        #pragma HLS unroll
        newState(i*4 + 3, i*4) = sBoxMasked[round%2][i][state(i*4 + 3, i*4)];
    }
    return newState;
}
```

PRESENT encryption in this subsection. The AES/Rijndael and Serpent implementations are very similar; all the implementations can be found at [158].

Listing 3.1 shows the top-level function, which determines I/O of the final IP core and defines the PRESENT encryption algorithm. First, the masked substitution layers S'_0 and S'_1 are computed using function $reconfigure()$. After that, 31 encryption rounds are performed, as described in section 3.1. Notice the pragma *pipeline* directive, which in this case assures a single cycle iteration latency. All the functions, but $reconfigure()$, get inlined during the synthesis process.

The substitution layer is described in Listing 3.2. Notice the pragma *unroll* directive, which causes the loop iterations to be scheduled in parallel. Also, notice the bit-slicing indexing of variables.

Listing 3.3 describes an area-optimized version of the dynamic logic reconfiguration of the substitution layers. Pragma $RESOURCE$ directive specifies a memory primitive

Listing 3.3: Reconfiguration of S-boxes.

```
void reconfigure(present_block_t maskIn, present_block_t maskOut) {
    #pragma HLS RESOURCE variable=sBoxMasked core=RAM_1P_LUTRAM
    #pragma HLS ARRAY_PARTITION variable=sBoxMasked complete dim=2
    present_block_t mask1[2]={maskOut, maskIn};
    present_block_t mask2InvP[2]={pInvLayer(maskIn), pInvLayer(maskOut)};
    L1: for(int i = 0; i < 2; i++){
        #pragma HLS unroll
        L2: for(int j = 0; j < SBOX_RANGE; j++){
            #pragma HLS pipeline
            L3: for(int k = 0; k < SBOX_COUNT; k++){
                #pragma HLS unroll
                present_sbox_t idx = mask1[i](4*k+3, 4*k) ^ j;
                present_sbox_t val = sBoxClean[j] ^ mask2InvP[i](4*k+3, 4*k);
                sBoxMasked[i][k][idx] = val;
            }
        }
    }
}
```

used to implement a variable, in this case, LUTRAM, i.e., a single-port distributed RAM. When the memory primitive is not explicitly set, synthesis chooses one automatically to satisfy the elaborated read/write schedule, taking both latency and resource utilization into account (this may, however, result in a nonoptimal solution, such as using 16 18K block RAMs).

Pragma *ARRAY_PARTITION* specifies partitioning of an array into smaller arrays, partially or completely, in the specified dimension. This results in using more instances of the underlying memory primitive, i.e., multiple smaller memories instead of one large memory, and it therefore also increases the number of R/W ports. In this case, the multi-dimensional $sBoxMasked$ array (declared in Listing 3.2) is partitioned completely in the second dimension, resulting in 16 memory instances (S-boxes) addressed using five bits ($round\%2$ and 4-bit input value). The 16 S-boxes, forming the substitution layer, are configured in parallel (loop $L3$), one input value at a time (loop $L2$). Even rounds and odd rounds substitution layers reconfigurations are constrained to be performed in parallel (loop $L1$); however, given the array partitioning and resulting access conflicts, they are scheduled sequentially by the synthesis tool. This version of dynamic logic reconfiguration is further referred to as Version 1.

In order to reduce the reconfiguration time complexity, the $sBoxMasked$ array may be further partitioned in the first dimension, resulting in 32 memory instances addressed using the 4-bit S-box input value. Swapping loops $L1$ and $L2$ now allows for parallel reconfiguration of both even rounds and odd rounds substitution layer S-boxes, resulting in lower time complexity at the expense of the area. This version of dynamic logic reconfiguration is further referred to as Version 2.

Unrolling all the three loops (and letting the synthesis resolve occurring read/write conflicts) results in even lower latency; this version is further referred to as Version 3.

Table 3.3: Post-RTL-synthesis area and timing estimates comparison.

#	Implementation	Area		Timing		
		FF	**LUT**	**CC**	**CP** [ns]	**TP** [Mb/s]
	PRESENT					
1	RTL, Naïve/Unprotected	150	223	32	2.371	843.5
2	HLS, Naïve/Unprotected	229	226	33	2.625	738.8
3	HLS, Protected, Version 1	392	420	33+37	3.056	299.1
4	HLS, Protected, Version 2	442	598	33+19	3.006	409.4
5	HLS, Protected, Version 3	439	771	33+16	3.784	345.2
6	[148] RTL, Protected	601	1,508	62+32	n/a	n/a
	AES/Rijndael					
7	RTL, Naïve/Unprotected	296	1,365	11	5.479	2,124
8	HLS, Naïve/Unprotected	435	1,436	12	3.696	2,886
9	HLS, Protected, Version 1	747	3,159	12+517	3.555	68.06
10	HLS, Protected, Version 2	861	3,248	12+259	4.195	112.5
11	RTL, Protected 2 CFGLUT chains	1,073	2,652	20+4,122	n/a	n/a
12	RTL, Protected 32 CFGLUT chains	3,229	7,234	20+282	n/a	n/a
	Serpent					
13	RTL, Naïve/Unprotected	403	1,403	33	5.598	692.9
14	HLS, Naïve/Unprotected	1,177	1,860	36	3.955	899.0
15	HLS, Protected, Version 1	1,814	2,764	36+144	4.703	151.2
16	HLS, Protected, Version 2	2,645	4,391	36+18	5.094	465.3
17	HLS, Protected, Version 3	2,650	5,103	36+16	5.094	483.2
18	RTL, Protected 2 CFGLUT chains	2,945	5,696	64+538	n/a	n/a
19	RTL, Protected 144 CFGLUT chains	4,441	9,471	64+58	n/a	n/a
20	RTL, Protected 288 CFGLUT chains	6,040	13,211	64+42	n/a	n/a

3.3.4 Latency, Throughput and Area Utilization

Table 3.3 compares area and latency of various FPGA implementations of PRESENT, AES/Rijndael and Serpent encryption algorithms:

- Unprotected VHDL implementation for register-transfer level synthesis,
- Unprotected C implementation for high-level synthesis,
- Alternating Masks Scheme C implementation for high-level synthesis, as proposed in this section,
- Protected register-transfer level implementation implementing S-box decomposition, Boolean masking and register precharge countermeasures combination, and utiliz-

ing a similar concept of dynamic logic reconfiguration using CFGLUT primitives. The concept was proposed for PRESENT by Sasdrich et al. [148] and extended for AES/Rijndael and Serpent in section 3.2.

The proposed implementations are synthesized for Xilinx Kintex-7, and their results are Vivado's post-RTL synthesis utilization statistics, except for implementations 6, 11, 12, and 18-20 (results reported in [148] and section 3.2), which were synthesized for Spartan-6. All implementations use look-up table S-boxes. Note that both Kintex-7 and Spartan-6 FPGAs utilize similar 6-input LUT primitives, making the area results comparable. The table summarizes flip flop count (FF), look-up tables count (LUT), clock cycle count (CC, encryption+reconfiguration), achievable clock period (CP) and final throughput (TP). It is essential to say that the work by Sasdrich et al. [148] and the work in section 3.2 both combine three countermeasures: S-box Decomposition, Boolean Masking, and Register Precharge, while our proposed scheme may be considered equivalent to using Boolean Masking + Register Precharge only.

The results presented for unprotected naïve implementations (implementations 1, 2, 7, 8, 13, 14) show that high-level synthesis is well capable of competing with the traditional register-transfer level synthesis, with reasonable overhead considering the advantages of the system-level approach. The C implementations further pipeline the round loop and its prologue/epilogue, compared to our RTL designs, which results in more clock cycles and larger area, but allows for higher throughput in the end. While similar results could be obtained with VHDL implementations as well, the RTL approach requires significantly more work and skill, and is therefore more time and human resource consuming.

Regarding the protected PRESENT (implementations 3-5), the single encryption clock cycle latency of our implementation is almost half of that reported by Sasdrich et al. [148] (implementation 6), since we do not use register precharge and we utilize two masks instead, making total latency of our implementations lower in all cases. The Version 2 of the proposed masking scheme achieves the best throughput.

Similar results can be observed for AES/Rijndael (implementations 9, 10) and Serpent (implementations 15-17). In the case of protected AES/Rijndael, only Versions 1 and 2 are available because of the synthesis tool limitations (there are too many operations to be scheduled in parallel when all the loops are unrolled). In comparison with results reported in section 3.2 (implementations 11, 12, 18-20), only implementation 11 reaches lower LUT count; the implementations described in this section exhibit better area and timing performance in all other cases. The results also confirm that our approach is more efficient for algorithms with smaller S-boxes: while Serpent, despite its worse clock period, is able to achieve throughput comparable to PRESENT, the performance of AES/Rijndael is significantly reduced.

Presented results demonstrate that a reasonable area overhead can be achieved when using the high-level synthesis. Furthermore, the system-level approach allows for quick exploration of the design space, evaluating the area and latency trade-off.

3.3.5 Side-Channel Leakage Evaluation

We evaluate side-channel leakage using both specific and non-specific t-test leakage assessment methodology (see section 2.7). The protected high-level implementations are synthesized and implemented with Xilinx Vivado 2019.2 tools using the default synthesis strategy. We evaluate all the versions of PRESENT, AES/Rijndael, and Serpent encryption. The implementations run on a dedicated side-channel evaluation board Sakura-X (i.e., a commercial version of SASEBO-GIII [70]), equipped with Xilinx Kintex-7 FPGA (28 nm), clocked by an external 200MHz crystal oscillator, which gets divided by an internal MMCM module down to 15 MHz, used to clock the encryption IP core. The FPGA design receives plaintexts and random masks from an external source (the controlling PC) and sends out the ciphertext using the UART interface. The power consumption is measured using PicoScope 6404D oscilloscope and Langer EMV-Technik PA 303 preamplifier. Voltage over the FPGA core is sampled in the VCCINT path, using 1Ω shunt resistor (Sakura-X originally uses 0.1Ω resistor), with the preamplifier acting as a DC blocker, and with 25MHz bandwidth limiter enabled on the oscilloscope. The sampling frequency used for all the measurements is 1.25 GS/s (i.e., 0.8 ns sampling period). Power traces are captured during every encryption, i.e., while the encryption IP core is active.

For all three encryption algorithms, both protected and unprotected versions (unprotected encryption is measured by setting both masks to all zeroes, so that $S = S'_0 = S'_1$), two sets of traces are captured: (1) during encryptions of uniform random plaintexts, and (2) during encryptions of either uniform or fixed constant plaintexts, in a randomly interleaved fashion. In every data set, one million power traces are measured. The first set of traces is used for the specific t-test, and the second one for the non-specific t-test.

3.3.5.1 Specific t-test

For the specific leakage assessment, the data sets with random plaintexts are used. The data set is split into two disjoint groups according to a bit in a chosen intermediate cipher value. We choose the output of the substitution layer (S-boxes outputs) in the first round (i.e., 64 different models for PRESENT and 128 different models for both AES/Rijndael and Serpent) and XOR of consecutive (second and third) rounds inputs (working register leakage) (i.e., another 64 or 128 different models). For every model independently, Welch's t-test statistic is computed, at every sampling point independently. The *null hypothesis* is that the two groups' means are equal, i.e., the groups are indistinguishable by their sample means. The hypothesis gets rejected for high values of $|t|$ according to Student's t-distribution and selected significance level. In side-channel leakage assessment, the value 4.5 or 5 is often considered a reasonable threshold for the $|t|$ value, to reject the hypothesis. This has to be done with the possibility of both false positives and false negatives in mind [161].

3.3.5.2 Non-specific t-test

For the non-specific leakage assessment, we use the data sets measured with either random or fixed plaintexts, in a randomly interleaved fashion. The data set is split into groups containing power traces measured using either random or fixed plaintexts, and the Welch's t-test statistic is computed once again at every sampling point independently, for the same *null hypothesis*, i.e., the encryptions of random or fixed plaintexts are indistinguishable by their sample means.

Unlike the specific t-test, the non-specific leakage assessment methodology is much more sensitive and less informative. Even though rejecting the *null hypothesis* indicates first-order leakage, its exploitability remains an open question and requires further investigation.

3.3.5.3 Results for Version 1

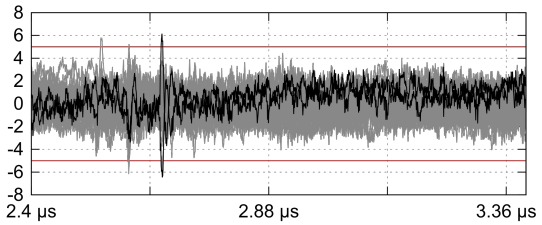
This subsubsection presents leakage evaluation results for the Version 1, i.e., the area-optimized implementations with less (and larger) memory primitives and longer reconfiguration times in comparison with Versions 2 and 3. Since no leakage occurs during the reconfiguration phase, it is cropped off the presented plots.

Figures 3.13, 3.14, and 3.15 show results of PRESENT evaluation, figures 3.16, 3.17, and 3.18 show results of AES/Rijndael evaluation, and figures 3.19, 3.20, and 3.21 show results of Serpent evaluation. Figures 3.13 and 3.14, 3.16 and 3.17, 3.19 and 3.20 show results of the specific t-tests based on the substitution layer (S-boxes) output and round register transitions (xor of consecutive rounds inputs), respectively. The specific test plots depict 64 or 128 overlaid curves: one for each model derived from a bit value in the cipher state. Figures 3.15, 3.18, and 3.21 show results of the non-specific t-tests. Subfigures (a) show results of the unprotected version, subfigures (b) show results of the protected version, and subfigures (c) show a comparison of the t-value progress of both unprotected and protected implementations.

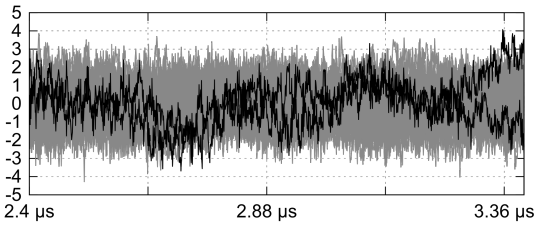
All tests detect leakage of all the unprotected implementations as can be seen in (a) subfigures 3.13a-3.21a. While specific tests aim at a single round, non-specific tests detect leakage during the entire encryption.

Protected PRESENT encryption successfully passes both specific tests, as can be seen in Figures 3.13b and 3.14b. In the non-specific test depicted in Figure 3.15b, there is only a single peak reaching the rejection threshold using one million power traces. However, observing the results by the naked eye, multiple potential points where non-specific leakage may occur can be suspected. A significant improvement over the unprotected version can be seen in Figure 3.15c.

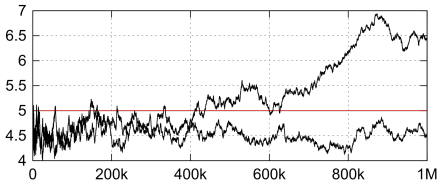
Protected AES/Rijndael encryption successfully passes the specific test aimed at the round register, as can be seen in Figure 3.17b. The specific test aimed at the substitution layer fails, as can be seen in Figure 3.16b, with multiple points reaching over the rejection threshold; its improvement over the unprotected version can be seen in Figure 3.16c. The non-specific test for AES/Rijndael fails during a large part of the encryption, as can be



(a) t-values during unprotected encryption.



(b) t-values during protected encryption.



(c) t-value progress.

Figure 3.13: PRESENT specific t-test results of 64 models based on substitution layer output (S-boxes outputs). (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.

seen in Figure 3.18b, with the most prominent peak observable at the beginning of the encryption.

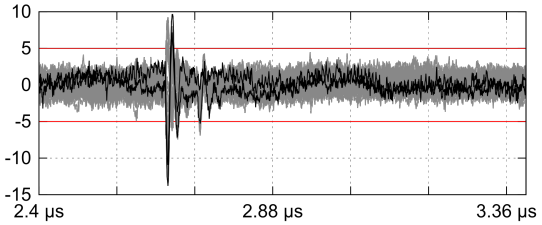
Protected Serpent encryption exhibits similar behavior as the AES/Rijndael: the specific test aimed at the round register passes, as can be seen in Figure 3.20b, while the test aimed at the substitution layer detects leakage, as can be seen in Figure 3.19b. Similar to AES/Rijndael, the non-specific test detects leakage during the entire encryption, with a significant peak at the beginning. However, a significant improvement compared to the unprotected version can be seen in both Figures 3.21b and 3.21c.

3.3.5.4 Results for Versions 2 and 3

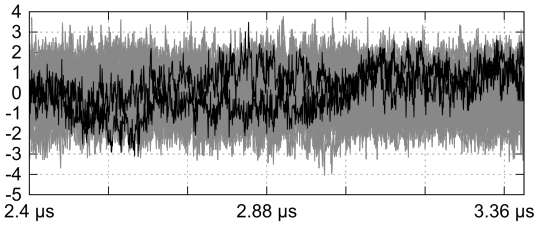
This subsubsection presents leakage evaluation results for the Versions 2 and 3, i.e., the implementations with more (and smaller) memory primitives and shorter reconfiguration times in comparison with Version 1.

The specific t-test aimed at the round register leakage (XOR of consecutive round inputs) passed the assessment for all the implementations, same as in the case of the Version 1 implementation.

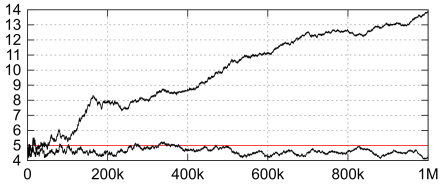
The results of the specific t-test aimed at the substitution layer (S-boxes outputs), for Version 1 and Version 2, are compared in Figure 3.22. Although the protected PRESENT Version 1 implementation passed this test, the Version 2 implementation failed with the t-



(a) t-values during unprotected encryption.

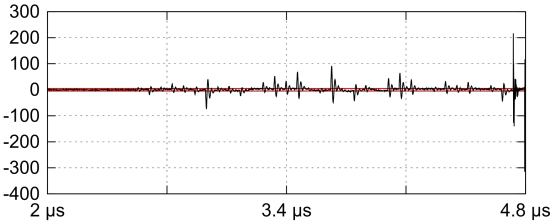


(b) t-values during protected encryption.

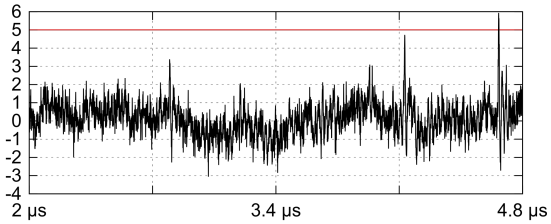


(c) t-value progress.

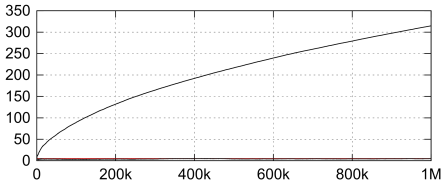
Figure 3.14: PRESENT specific t-test results of 64 models based on round register leakage (xor of consecutive rounds inputs). (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.



(a) t-values during unprotected encryption.

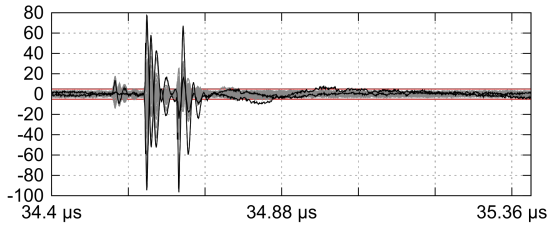


(b) t-values during protected encryption.

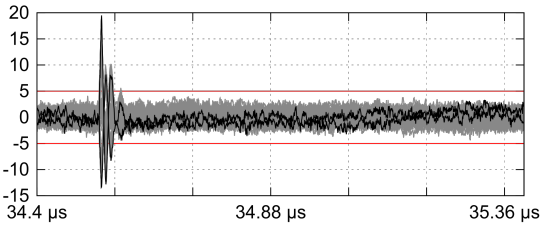


(c) t-value progress.

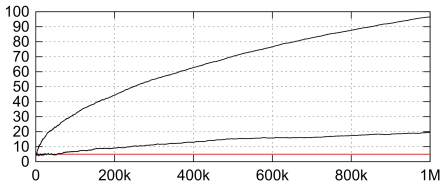
Figure 3.15: PRESENT non-specific t-test results. (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.



(a) t-values during unprotected encryption.

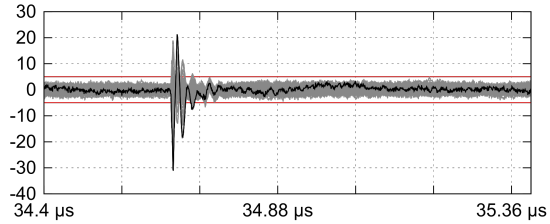


(b) t-values during protected encryption.

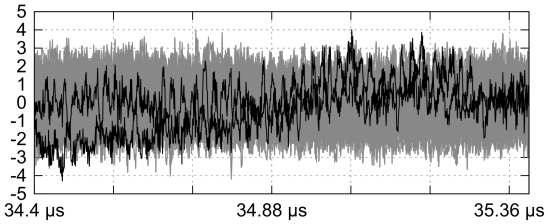


(c) t-value progress.

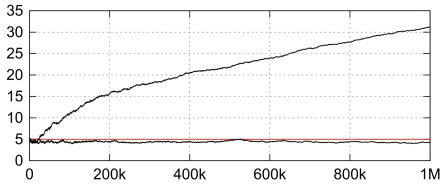
Figure 3.16: AES/Rijndael specific t-test results of 128 models based on substitution layer output (S-boxes outputs). (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.



(a) t-values during unprotected encryption.

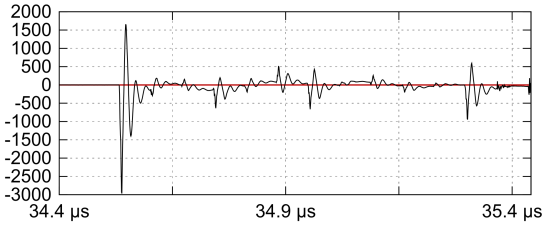


(b) t-values during protected encryption.

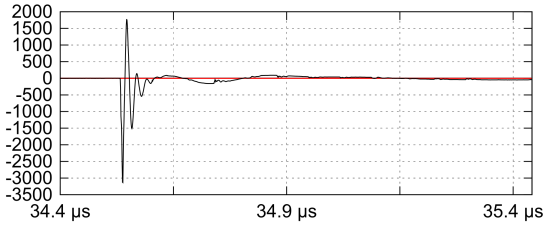


(c) t-value progress.

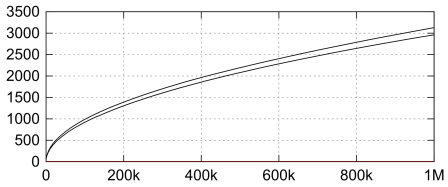
Figure 3.17: AES/Rijndael specific t-test results of 128 models based on round register leakage (xor of consecutive rounds inputs). (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.



(a) t-values during unprotected encryption.

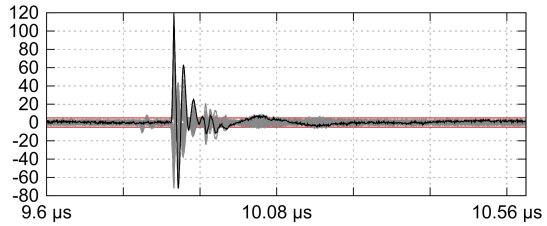


(b) t-values during protected encryption.

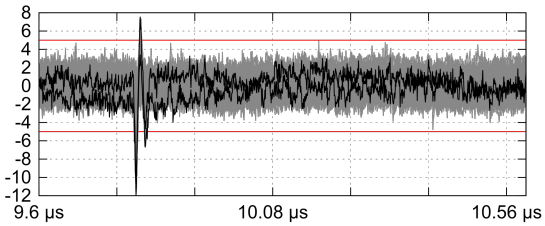


(c) t-value progress.

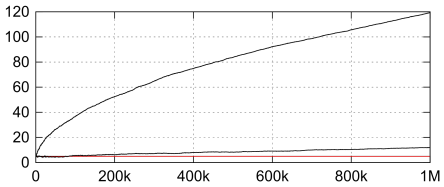
Figure 3.18: AES/Rijndael non-specific t-test results. (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the lower curve corresponds to the unprotected encryption, and the upper curve corresponds to the protected encryption.



(a) t-values during unprotected encryption.

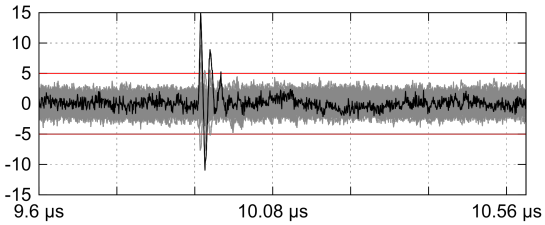


(b) t-values during protected encryption.

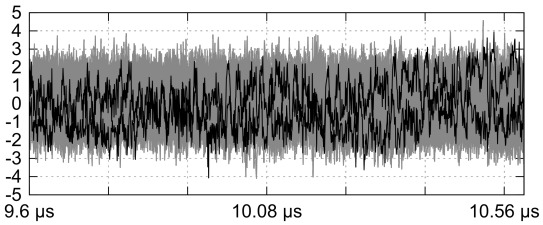


(c) t-value progress.

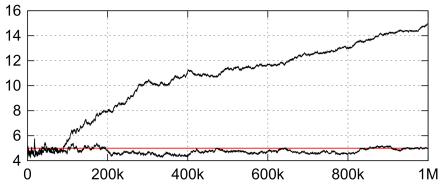
Figure 3.19: Serpent specific t-test results of 128 models based on substitution layer output (S-boxes outputs). (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.



(a) t-values during unprotected encryption.

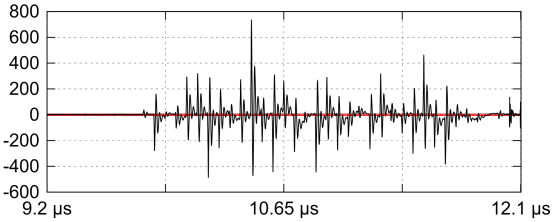


(b) t-values during protected encryption.

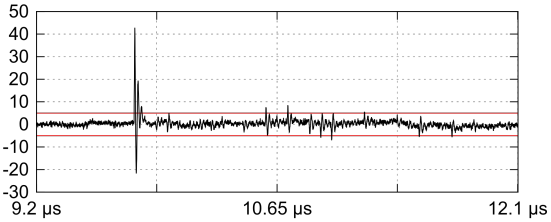


(c) t-value progress.

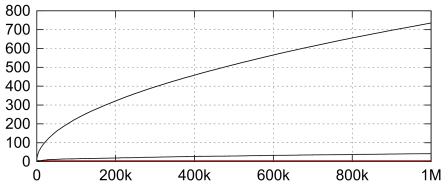
Figure 3.20: Serpent specific t-test results of 128 models based on round register leakage (xor of consecutive rounds inputs). (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.



(a) t-values during unprotected encryption.

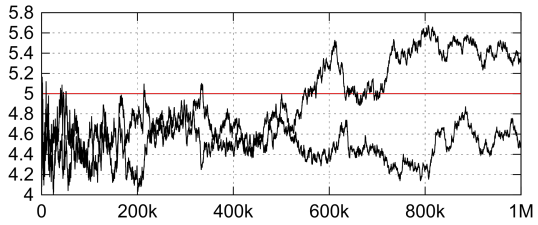


(b) t-values during protected encryption.

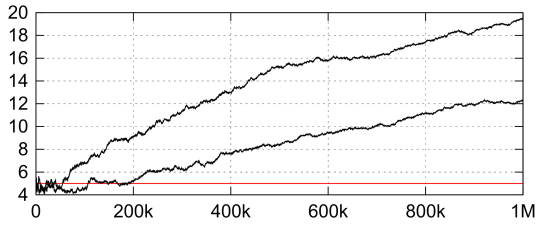


(c) t-value progress.

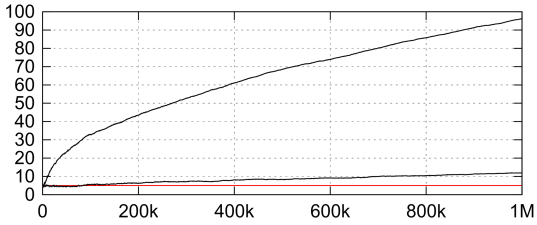
Figure 3.21: Serpent non-specific t-test results. (a) and (b) show the t-value on the vertical axis and the time samples during the encryption on the horizontal axis. (c) shows maximum reached absolute t-value on the vertical axis and the number of power traces on the horizontal axis, where the upper curve corresponds to the unprotected encryption, and the lower curve corresponds to the protected encryption.



(a) PRESENT.



(b) AES/Rijndael.



(c) Serpent.

Figure 3.22: Results of the specific t-tests based on substitution layer output (S-boxes outputs), where the maximum reached absolute t-value is shown on the vertical axis and the number of power traces is shown on the horizontal axis. In (a) and (c), the lower curve corresponds to the Version 1 and the upper curve corresponds to the Version 2 protected encryption implementations. In (b), the upper curve corresponds to the Version 1 and the lower curve corresponds to the Version 2.

value reaching as high as 5.5. While the AES/Rijndael's t-value reaches lower for Version 2 than for Version 1, it still fails the test. A significant worsening can be seen for the Serpent implementation in Figure 3.22c. We suspect this difference to be caused by the eight Serpent S-boxes being more vulnerable to glitches, as opposed to Rijndael's single S-box.

The results of the non-specific t-tests exhibit very similar results as the ones aimed at the substitution layer. The Version 2 PRESENT non-specific t-test fails with the t-value reaching as high as 15, compared to approximately 5 for the Version 1. The Version 2 of the Serpent encryption once again exhibits a significant worsening with the t-value reaching as high as 500, compared to approximately 40 for the Version 1.

The Version 3 implementations of PRESENT and Serpent exhibit very similar behavior as the Version 2 implementations.

3.3.6 Discussion and Future Work

The presented results show that Version 1 of our protected PRESENT encryption implementation passes the specific leakage assessment, suggesting that it is not vulnerable to the first-order non-profiled side-channel attacks (such as DPA or CPA) using one million measurements and the straightforward leakage model. It subtly fails the non-specific leakage assessment in a single time instant only, giving a reasonable confidence about its resistance

against first-order side-channel attacks using at most one million measurements.

While AES/Rijndael and Serpent pass the specific tests aimed at the round register, the presented results suggest an exploitable leakage when aiming the tests at the substitution layer (S-boxes) output. Also, their non-specific leakage assessment results suggest an exploitable leakage during entire encryptions.

Note that while the proposed countermeasures work quite well for PRESENT, their performance is worse on the "full-sized" ciphers AES/Rijndael and Serpent. We believe this is caused by more complex linear layers that are more vulnerable to glitches, which are known to cause side-channel leakage in masked circuits. Similarly, we believe that the Version 2 and 3 implementations are more vulnerable to glitches because of the additional logic due to smaller memory primitives. While the leakage caused by glitches may not be as trivially exploitable, a successful attack may still be feasible [101]. The AES/Rijndael, Serpent and Version 2 and 3 PRESENT failing the non-specific leakage assessment also suggest that attacks assuming a more powerful adversary (such as Template attacks, see subsection 2.3.1) might be feasible.

Leakage sources such as glitches are mitigated by more complex higher-order glitch-resistant masking schemes, such as Threshold implementation (see subsubsection 2.5.3.2) or Domain oriented masking [63]. However, these schemes typically require a more precise control over the FPGA design than provided by the high-level synthesis (e.g., placement of pipeline registers). These implementation necessities make these schemes less suitable for high-level synthesis, where the data path design and control is offloaded to the synthesis tools.

Another possible approach might be altering the high-level synthesis process so that a glitch-free circuit is synthesized. An algorithm for formal verification of higher-order masking schemes is presented in [12], and a side-channel analysis of a source code at compilation time was recently proposed in [31]. System-level FPGA design introduces many novel challenges, including automated side-channel leakage assessment and verification.

3.3.7 Summary

In this section, we showed that the high-level synthesis is well capable of competing with the traditional RTL design when it comes to the unprotected cryptographic implementations. We implemented PRESENT, AES/Rijndael and Serpent encryption for FPGA in both C and VHDL, and we compared the results, demonstrating the benefits of the high-level synthesis such as rapid prototyping and easy design space exploration.

We further proposed a Boolean masking scheme, utilizing dynamic logic reconfiguration, and suitable for high-level synthesis from the C language algorithmic description. The Alternating Masks Scheme was proposed as an alternative to the masking and register precharge combination to deal with Hamming distance leakage. In addition to a purely algorithmic description, the proposed scheme also allows for higher throughput than a combination of masking and register precharge.

We implemented PRESENT, AES/Rijndael, and Serpent encryption with the proposed side-channel countermeasures in C language, and we synthesized the protected implementa-

tions for Xilinx FPGA. We have evaluated the area/latency trade-off, and we compared our results with unprotected implementations and with an existing state-of-the-art dynamic reconfiguration-based protected implementation. We showed that the overhead brought in by high-level synthesis is reasonable considering both area and latency while allowing for fast design space exploration.

To evaluate the effectiveness of the proposed side-channel countermeasure, we performed a specific t-test leakage assessment using one million power traces, focusing on the substitution layer output and the XOR of consecutive rounds inputs, and a non-specific t-test leakage assessment using fixed vs. random plaintext methodology. Our results show that the proposed masking scheme successfully conceals the first-order side-channel leakage of PRESENT encryption; however, we were able to detect leakage of AES/Rijndael and Serpent implementations. We discussed the leakage assessment results and their consequences, and we identified the related limitations of the system-level approach and high-level synthesis.

3.4 Summary

In this chapter, the countermeasures for symmetric cryptography in FPGA were proposed and evaluated. All the proposed countermeasures were implemented for Xilinx FPGAs and their time/area performance was comprehensively evaluated. The countermeasures effectiveness was evaluated using a state-of-the-art leakage assessment methodology. Countermeasures based on dynamic logic reconfiguration for AES and Serpent were discussed in section 3.2 and thoroughly summarized in subsection 3.2.7. A novel approach for high-level synthesis of side-channel countermeasures for PRESENT, AES and Serpent was presented in section 3.3 and thoroughly summarized in subsection 3.3.7.

Chapter 4

Asymmetric Cryptography

The work presented in this chapter was done in cooperation with David Pokorný (a master's student supervised by the thesis author until 2021, now a Ph.D. student) of Czech Technical University. The work in section 4.2 was presented at the DATE conference [A.6] in 2021. The work presented in section 4.3 was published in the Electronics journal [A.7] in 2022.

One of the significant threats to asymmetric cryptographic algorithms is quantum computation, which is expected to effectively break RSA and ECC cryptosystems. This is due to Shor's algorithm [155], which allows prime factorization and discrete logarithm solving in polynomial time, compared to exponential time on a classical computer. For this reason, the National Institute of Standards and Technology (NIST) of the United States Department of Commerce has initiated a process to standardize quantum-resistant public key cryptographic algorithms. The digital signature candidates in the third round were CRYSTALS-Dilithium [52], FALCON [55], both lattice-based schemes, and Rainbow [50], a multivariate quadratic scheme. The Rainbow algorithm, as proposed, was unfortunately found insecure [18] in 2022. Both lattice-based signatures have been currently recommended for standardization, with some multivariate quadratic candidate expected to be further evaluated. In this chapter, we focus on the multivariate quadratic signature scheme Rainbow, as most of the work was done prior to it being broken. However, the presented work is also applicable to other multivariate quadratic schemes, which, besides Rainbow, also include the Unbalanced Oil and Vinegar (UOV) [82] and LUOV [19] schemes.

First, section 4.1 describes the Rainbow algorithm, a generalization of the oil and vinegar scheme (which is a single-layered Rainbow). Then in section 4.2, a side-channel attack on the 32-bit reference implementation of the Rainbow, which was submitted to the NIST standardization process, is described and evaluated. In section 4.3, a novel countermeasure for multivariate quadratic signature schemes is proposed, analyzed and evaluated regarding side-channel leakage and time/memory overhead. Finally in section 4.4, a summary of the results is given.

4.1 Rainbow Multivariate Quadratic Signature

Rainbow is a post-quantum digital signature, a generalization of the oil and vinegar signature scheme [50]. Its security depends on the fact that solving m quadratic equations for n variables becomes very difficult. These equations are defined by a central map F, which consists of multivariate quadratic polynomials. The central map is structured so that it is possible to solve $F(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ using a linear equation solver. In a public key, the special structure of the central map is hidden by two linear maps, S and T, applied on an input and an output of the central map. The central map is layered, where each layer defines two types of variables: vinegar variables for known variables (randomly generated or computed from the prior layer) and oil variables for unknown variables (to be computed using polynomials in the given layer). In this thesis, we used the version of the Rainbow algorithm as defined in the submission to the NIST competition [139].

Let $\mathbb{F}$ be a finite field and $v_1, o_1, o_2 \in \mathbb{N}$ the parameters of the two-layered version of Rainbow. The parameters define the number of variables and the sizes of the layers. The first layer consists of o_1 quadratic polynomials with v_1 vinegar variables with indices $V_1 = \{1, 2, \ldots, v_1\}$ and o_1 oil variables with indices $O_1 = \{v_1 + 1, \ldots, v_1 + o_1\}$. The second layer contains o_2 quadratic polynomials with $v_2 = v_1 + o_1$ vinegar variables with indices $V_2 = \{1, 2, \ldots, v_2\}$ and o_2 oil variables with indices $O_2 = \{v_2 + 1, \ldots, v_2 + o_2\}$. Overall, we have $m = o_1 + o_2$ polynomials with $n = v_1 + o_1 + o_2$ variables.

The central map $F = (f^{(v_1+1)}, f^{(v_1+2)}, \ldots, f^{(n)})$ is an m-tuple of n-variate quadratic polynomials. These polynomials, indexed by $k \in O_1 \cup O_2$, are defined as:

$$f^{(k)}(x_1, \ldots, x_n) := \sum_{\substack{i,j \in V_l \\ i \leq j}} \alpha_{i,j}^{(k)} x_i x_j + \sum_{\substack{i \in V_l \\ j \in O_l}} \beta_{i,j}^{(k)} x_i x_j + \sum_{i \in V_l \cup O_l} \gamma_i^{(k)} x_i + \delta^{(k)}, \tag{4.1}$$

where $\alpha_{i,j}^{(k)}, \beta_{i,j}^{(k)}, \gamma_i^{(k)}, \delta^{(k)} \in \mathbb{F}$ are term coefficients and $l \in \{1, 2\}$ is such that $k \in O_l$.

For simplicity, we changed the notation in Equation 4.1 in the following sense:

- We restricted the polynomials, omitting the linear and absolute parts of the polynomials in the central map, resulting in quadratic forms. These coefficients are not necessary for the security of Rainbow, but add more complexity. The reference implementation submitted to the standardization process considers these coefficients in the documentation, but they were not implemented in the code. Nevertheless, the countermeasure discussed later in section 4.3 can be applied to the original polynomials defined in Equation 4.1 as well.

- Sometimes, we unite the α and β coefficients into the λ coefficients for simplicity of notation and also include zero coefficients as described in Equation 4.2. We do not allow setting any (by definition) zero coefficient to a non-zero value, so there is no

change from the original definition.

$$\lambda_{i,j}^{(k)} := \begin{cases} \alpha_{i,j}^{(k)} & (k \in O_1), (i,j \in V_1), (i \leq j), \\ \beta_{i,j}^{(k)} & (k \in O_1), (i \in V_1), (j \in O_1), \\ \alpha_{i,j}^{(k)} & (k \in O_2), (i,j \in V_2), (i \leq j), \\ \beta_{i,j}^{(k)} & (k \in O_2), (i \in V_2), (j \in O_2), \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Applying our changes, we obtain a simple notation of the central map polynomials, equivalent to Equation 4.1:

$$f^{(k)}(\mathbf{x}) = \sum_{i,j \in \hat{n}} \lambda_{i,j}^{(k)} \cdot x_i \cdot x_j, \tag{4.3}$$

where $k \in O_1 \cup O_2, \hat{n} = \{1, 2, \ldots, n\}$ and $\mathbf{x} = (x_1, \ldots, x_n)$.

In Rainbow, the structure of the central map F (described in Equation 4.1) is hidden using two random invertible affine maps $S : \mathbb{F}^m \to \mathbb{F}^m$ and $T : \mathbb{F}^n \to \mathbb{F}^n$. Similar to the linear and absolute parts of the central map polynomials, we also omitted translations of the affine maps. Therefore, we used two linear maps represented by regular matrices $\mathbf{S} \in \mathbb{F}^{m \times m}, \mathbf{T} \in \mathbb{F}^{n \times n}$. This is also consistent with the reference implementation.

4.1.1 Matrix Multiplication in the Reference Implementation

Three variants of the Rainbow signature scheme are proposed in the NIST competition, and their reference implementations are available. The Cyclic variant is motivated by Petzoldt's cyclic Rainbow scheme [131], the Compressed variant stores the private key in the form of a 512-bit seed, and the Classic variant stores plain matrices. For the demonstration in this subsection, we will discuss the Classic variant Ia with parameters selected to fit NIST security categories I and II.

This two-layered ($u = 2$) variant uses $m = 64$ quadratic polynomials with $n = 96$ variables over $\mathbb{F} = GF(2^{2^2}) = GF(16)$. Layers are structured as $(v_1, v_2, v_3) = (32, 64, 96)$ and $(o_1, o_2) = (32, 32)$. This variant uses matrices S and T of the form

$$S = S^{-1} = \begin{pmatrix} \mathbb{I} & S' \\ \mathbb{O} & \mathbb{I} \end{pmatrix}, \tag{4.4}$$

$$T = \begin{pmatrix} \mathbb{I} & T^{(1)} & T^{(2)} \\ \mathbb{O} & \mathbb{I} & T^{(3)} \\ \mathbb{O} & \mathbb{O} & \mathbb{I} \end{pmatrix}, \; T^{-1} = \begin{pmatrix} \mathbb{I} & T^{(1)} & T^{(4)} \\ \mathbb{O} & \mathbb{I} & T^{(3)} \\ \mathbb{O} & \mathbb{O} & \mathbb{I} \end{pmatrix}, \tag{4.5}$$

where $S \in \mathbb{F}^{64 \times 64}$, $T \in \mathbb{F}^{96 \times 96}$, their submatrices are elements of $\mathbb{F}^{32 \times 32}$, $\mathbb{O}$ is zero matrix, $\mathbb{I}$ is identity matrix, and $T^{(4)} := T^{(1)} \cdot T^{(3)} - T^{(2)}$. In the following text, we denote $y, h \in \mathbb{F}^{64}$ and $z, x \in \mathbb{F}^{96}$, where $y = S^{-1} \cdot h$ and $z = T^{-1} \cdot x$

Each element of $GF(2^{2^2})$ is identified by 4 bits, while the considered reference implementation uses a 32-bit word. A suitable Galois field was selected so that multiplication of

a vector by one element can be performed using simple bit operations, allowing for word-level data parallelism. Consequently, a vector of eight elements can be multiplied by one element using only a few instructions.

E.g., consider a computation of $y = S^{-1} \cdot h$. First, the product $S' \cdot h_{33:64}$ is computed in column-wise order. Using word-level parallelism, each matrix column is processed as four vectors of eight elements (i.e., the external loop iterates across all columns, and the internal loop iterates across four vectors). Finally, the matrix-vector product $y = S^{-1} \cdot h$ is obtained by addition of the h vector, to take the identities $\mathbb{I}$ in S^{-1} into account: $y_{1:32} = S' \cdot h_{33:64} + h_{1:32}$, $y_{33:64} = h_{33:64}$.

4.1.2 Central Map in Matrix Representation

For compact writing, we rewrite the quadratic forms in the central map as a product of matrix–vector multiplication for $k \in O_1 \cup O_2$:

$$
\begin{aligned}
f^{(k)}(\mathbf{x}) &= \mathbf{x}^\intercal \mathbf{F}^{(k)} \mathbf{x} = \mathbf{x}^\intercal \begin{pmatrix} \mathbf{F}_{1,1}^{(k)} & \mathbf{F}_{1,2}^{(k)} & \mathbf{F}_{1,3}^{(k)} \\ \mathbf{0} & \mathbf{F}_{2,2}^{(k)} & \mathbf{F}_{2,3}^{(k)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \mathbf{x}, \\
\mathbf{F}_{1,1}^{(k)} &= \{\lambda_{i,j}^{(k)}\}_{i,j \in V_1}, \\
\mathbf{F}_{1,2}^{(k)} &= \{\lambda_{i,j}^{(k)}\}_{i \in V_1, j \in O_1}, \\
\mathbf{F}_{1,3}^{(k)} &= \{\lambda_{i,j}^{(k)}\}_{i \in V_1, j \in O_2}, \\
\mathbf{F}_{2,2}^{(k)} &= \{\lambda_{i,j}^{(k)}\}_{i,j \in O_1}, \\
\mathbf{F}_{2,3}^{(k)} &= \{\lambda_{i,j}^{(k)}\}_{i \in O_1, j \in O_2}.
\end{aligned}
\tag{4.6}
$$

The quadratic form $f^{(k)}(\mathbf{x})$ can be expressed using a matrix $\mathbf{F}^{(k)}$. This matrix is partitioned according to the structure of the central map. Submatrices $\mathbf{F}_{1,1}^{(k)}, \mathbf{F}_{2,2}^{(k)}$ are upper triangulars and correspond to alpha coefficients. Submatrices $\mathbf{F}_{1,2}^{(k)}, \mathbf{F}_{1,3}^{(k)}, \mathbf{F}_{2,3}^{(k)}$ correspond to beta coefficients. Specifically for the first layer, we obtain:

$$
(k \in O_1) \implies \mathbf{F}^{(k)} = \begin{pmatrix} \mathbf{F}_{1,1}^{(k)} & \mathbf{F}_{1,2}^{(k)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{4.7}
$$

The central map can be expressed as a vector of m matrices (representing quadratic forms):

$$
\mathbf{F} = \begin{pmatrix} \mathbf{F}^{(v_1+1)} \\ \mathbf{F}^{(v_1+2)} \\ \vdots \\ \mathbf{F}^{(n)} \end{pmatrix}. \tag{4.8}
$$

4.1.3 Secret and Public Keys

Secret key SK can be expressed and stored as:

$$\mathrm{SK} := (S^{-1}, F, T^{-1}) \leftrightarrow (\mathbf{S}^{-1}, \mathbf{F}, \mathbf{T}^{-1}), \tag{4.9}$$

where

$$\begin{aligned} &F : \mathbb{F}^n \to \mathbb{F}^m \\ &\mathbf{x} \mapsto \mathbf{y} = F(\mathbf{x}) = \begin{pmatrix} f^{(v_1+1)}(\mathbf{x}) \\ f^{(v_1+2)}(\mathbf{x}) \\ \vdots \\ f^{(n)}(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \mathbf{x}^\intercal \mathbf{F}^{(v_1+1)} \mathbf{x} \\ \mathbf{x}^\intercal \mathbf{F}^{(v_1+2)} \mathbf{x} \\ \vdots \\ \mathbf{x}^\intercal \mathbf{F}^{(n)} \mathbf{x} \end{pmatrix}. \end{aligned} \tag{4.10}$$

The public key PK := (P) contains only a quadratic map P defined as

$$P := S \circ F \circ T. \tag{4.11}$$

4.1.4 Signing and Verification Process

For document $\mathbf{d}$, random salt $\mathbf{r}$, and a secret key $\mathrm{SK} = (S^{-1}, F, T^{-1})$, we define

$$\begin{aligned} \mathbf{h} &\coloneqq \mathrm{hash}\left(\mathrm{hash}\left(\mathbf{d}\right) || \ \mathbf{r}\right), \\ \mathbf{y} &\coloneqq S^{-1}(\mathbf{h}), \\ \mathbf{x} &\coloneqq F^{-1}(\mathbf{y}), \\ \mathbf{z} &\coloneqq T^{-1}(\mathbf{x}), \end{aligned} \tag{4.12}$$

where $\mathbf{h}, \mathbf{y} \in \mathbb{F}^m$ and $\mathbf{x}, \mathbf{z} \in \mathbb{F}^n$. The pair $(\mathbf{z}, \mathbf{r})$ is called a signature.

The signature $(\mathbf{z}, \mathbf{r})$ of the document $\mathbf{d}$ is valid iff $\mathbf{h} = \mathbf{h}'$, where

$$\begin{aligned} \mathbf{h} &\coloneqq \mathrm{hash}\left(\mathrm{hash}\left(\mathbf{d}\right) || \ \mathbf{r}\right), \\ \mathbf{h}' &\coloneqq P(\mathbf{z}). \end{aligned} \tag{4.13}$$

4.2 Side-Channel Attack on the 32-bit Reference Implementation of the Rainbow

In this section, we propose a CPA attack (see subsection 2.2.3) on the 32-bit reference implementation of Rainbow from the NIST competition's second round, which has been submitted as a candidate for NIST post-quantum digital signature standardization. The proposed attack is an extension of the work presented in [129]. We propose a way to extract the private key from the device and evaluate our attack on an STM32F3 ARM microcontroller.

The signing process is described in subsection 4.1.4. The S^{-1} map is applied first, followed by the inverse of central map F, and finally applying the T^{-1} map. Vinegar variables for the first layer are generated randomly at the beginning of the algorithm.

With matrices S^{-1} and T^{-1} known, we can reveal the central map F easily with knowledge of public key PK = (P). Therefore, we aim our attack at the linear parts S and T only. Note that $S = S^{-1}$ and T^{-1} can be computed from T and vice versa. We cannot choose the input of the S matrix multiplication directly due to salting, but we can compute its value as we supply d and know s from the resulting signature.

4.2.1 Attack on S Map

In the first signing step, a matrix-vector product $y = S^{-1} \cdot h$ is computed (detailed in subsection 4.1.1). Our attack is aimed at the computation of $y_{1:32} = S' \cdot h_{33:64} + h_{1:32}$, where h is a known vector and S' is a part of the private key.

4.2.1.1 Attack I

Our CPA attack therefore is row-oriented. Each element i of the final product can be expressed as

$$y_i = \sum_{j=1}^{32} (S'_{i,j} \cdot h_{j+32}) + h_i. \tag{4.14}$$

In the reference implementation, vector y_i is initialized with zeroes, then h is multiplied with S' iteratively, and finally h_i is added due to the identity submatrices in Equation 4.4. This is one of the differences compared to [129], where h_i is added first, making their attack substantially easier to mount. Our predictions are based on a Hamming weight of the intermediate sum value for $j \in \{2, \ldots, 32\}$. Table 4.1 summarizes intermediate values for *Attack I*.

Table 4.1: Attack I: Revealing a matrix row.

Target	Intermediate value
$S'_{i,1}$ and $S'_{i,2}$	$S'_{i,1} \cdot h_{33} + S'_{i,2} \cdot h_{34}$
$S'_{i,3}$	$\sum_{j=1}^{2}(S'_{i,j} \cdot h_{j+32}) + S'_{i,3} \cdot h_{35}$
$S'_{i,4}$	$\sum_{j=1}^{3}(S'_{i,j} \cdot h_{j+32}) + S'_{i,4} \cdot h_{36}$
$\vdots$	$\vdots$
$S'_{i,32}$	$\sum_{j=1}^{31}(S'_{i,j} \cdot h_{j+32}) + S'_{i,32} \cdot h_{64}$

In the first step, we target both $S'_{i,1}$ and $S'_{i,2}$ subkeys. Since the attacked 4-bit subkeys can have 16 different values, and there are 32 subkeys in the first matrix column, targeting only $S'_{i,1}$ would not lead to a useful solution. Using these predictions, multiple subkeys are found in the first step since the targeted intermediate value does not correspond to a unique input value. To resolve this, we further process these subkeys independently.

Another problem arises if $S'_{i,2} = 0$. The power predictions would then be the same as targeting only $S'_{i,1}$:

$$S'_{i,1} \cdot h_{33} + S'_{i,2} \cdot h_{34} = S'_{i,1} \cdot h_{33} + 0 \cdot h_{34} = S'_{i,1} \cdot h_{33}. \tag{4.15}$$

The same problem occurs for each zero element in the row. This problem can be overcome for columns with index $k > 2$. Knowing $S'_{i,j}, j \in \{1, \ldots, k-1\}$, our attack considers only non-zero values, i.e., $S_{i,k} \in \{1, \ldots, 15\}$. If no significant correlation with any of these 15 hypotheses is found, the element is assumed to be a zero.

Furthermore, each row is multiplied by the same vector $h_{33:64}$, and therefore, even if we find the whole row, we are not able to directly distinguish the row's index. We obtain $S'_{i,1:32}$ for some $i \in \{1, \ldots, 32\}$.

4.2.1.2 Attack II

After revealing elements of the entire row, the row index must be further identified. This is accomplished by *Attack II*, targeting the final addition of vector $h_{1:32}$. The used power predictions are described in Table 4.2. Using *Attack I* and *Attack II*, we are able to reveal 28 rows[1] out of 32 on average.

Table 4.2: Attack II: Row identification.

Target	Intermediate value
$k \in \{1, \ldots, 32\}$ in h_k	$\sum_{j=1}^{32}(S'_{i,j} \cdot h_{j+32}) + h_k$

4.2.1.3 Attack III

The last step is revealing the remaining rows. To do so, we exploit the word-level parallelism described in subsection 4.1.1. All the matrix rows are partitioned into sets based on this parallelism, i.e., the rows that are processed together are in their respective sets. For each row we attack, the subkey hypotheses are considered together with the other (already known) rows in the set. The power predictions are then based on the Hamming weight of the whole word. We define the aforementioned partitions and a hypothesis H_j^l for every column j and a certain set $l \in \{1, 2, 3, 4\}$ as

$$Set_l \coloneqq \{8 \cdot (l-1) + 1, \ldots, 8 \cdot l\}, \tag{4.16}$$

$$H_j^l \coloneqq \sum_{i \in Set_l} \mathrm{HW}(\sum_{k=1}^{j}(S'_{i,k} \cdot h_{k+32})). \tag{4.17}$$

Considering leakage hypotheses of the entire set allows for a more precise prediction based on the processed word. For simplicity, we consider zeroes instead of the unknown elements in S'.

[1]Probability of non-zero values in the first two columns is $(15/16)^2$, the average number of these rows in a submatrix such as S' is Mean(BinomialDistribution$(32, (15/16)^2)) = 28.125$.

Attack III for i-th row, where l is such that $i \in Set_l$, uses power predictions described in Table 4.3. The predictions are based on a sum of the most probable hypotheses for the other rows determined by previous attacks, and on a hypothesis for the i-th row. In case there are multiple missing rows in the set, the row index must be identified. We accomplish this using the last power prediction in Table 4.3. The attack is then repeated until the entire matrix S' is revealed. *Attack III* is only necessary for revealing the first two row elements $S'_{i,1}$ and $S'_{i,2}$. The following row elements can be revealed using either *Attack III*, or using *Attack I* and *Attack II*.

Table 4.3: Attack III: Revealing remaining rows.

Target	Intermediate value I	Power prediction
$S'_{i,1}$ and $S'_{i,2}$	$S'_{i,1} \cdot h_{33} + S'_{i,2} \cdot h_{34}$	$H_2^l + \mathrm{HW}(I)$
$S'_{i,3}$	$\sum_{j=1}^{2}(S'_{i,j} \cdot h_{j+32}) + S'_{i,3} \cdot h_{35}$	$H_3^l + \mathrm{HW}(I)$
$S'_{i,4}$	$\sum_{j=1}^{3}(S'_{i,j} \cdot h_{j+32}) + S'_{i,4} \cdot h_{35}$	$H_4^l + \mathrm{HW}(I)$
$\vdots$	$\vdots$	$\vdots$
$S'_{i,32}$	$\sum_{j=1}^{31}(S'_{i,j} \cdot h_{j+32}) + S'_{i,32} \cdot h_{64}$	$H_{32}^l + \mathrm{HW}(I)$
$k \in Set_l$	$\sum_{j=1}^{32}(S'_{i,j} \cdot h_{j+32}) + h_k$	$\mathrm{HW}(I)$

4.2.2 Attack on T Map

Matrix T^{-1} has three non-trivial sub-matrices $T^{(1)}$, $T^{(4)}$ and $T^{(3)}$. We attack them separately, but similarly. First, we express the matrix-vector multiplication described in subsection 4.1.1 using equations

$$\begin{aligned} x_{1:32} + T^{(1)} \cdot x_{33:64} + T^{(4)} \cdot x_{65:96} &= z_{1:32}, \\ x_{33:64} + T^{(3)} \cdot x_{65:96} &= z_{33:64}, \\ x_{65:96} &= z_{65:96}. \end{aligned} \tag{4.18}$$

As this multiplication is performed at the end of the signing, we know the output z, but not the input x. This is the main difference compared to attacking the S matrix.

4.2.2.1 Attack on $T^{(3)}$

To reveal $T^{(3)}$, we use the *Attacks I, II* as described for S, since we know the vector used in multiplication:

$$\underbrace{T^{(3)}}_{\text{secret key}} \cdot \underbrace{z_{65:96}}_{\text{known}} + \underbrace{x_{33:64}}_{\text{unknown}} = \underbrace{z_{33:64}}_{\text{known}}. \tag{4.19}$$

Unfortunately, we cannot use the final addition to determine the row indices in this case. Instead, we use the right side of the Equation 4.19 and the fact that

$$\sum_{j=1}^{32}(T_{i,j}^{(3)} \cdot z_{j+65}) + z_{i+32} = x_{i+32}. \tag{4.20}$$

Similarly to the S matrix attack, we use *Attack III* to reveal elements from rows beginning with zeroes. When $T^{(3)}$ is found, we can compute

$$x_{33:64} = T^{(3)} \cdot z_{65:96} + z_{33:64}. \tag{4.21}$$

4.2.2.2 Attack on $T^{(4)}$

Attacking $T^{(4)}$ is performed in the same manner as attacking $T^{(3)}$, using equation

$$\underbrace{x_{1:32} + T^{(1)} \cdot x_{33:64}}_{\text{unknown}} + \underbrace{T^{(4)}}_{\text{secret key}} \cdot \underbrace{z_{65:96}}_{\text{known}} = \underbrace{z_{1:32}}_{\text{known}}. \tag{4.22}$$

The submatrix $T^{(4)}$ is multiplied by a known vector $z_{65:96}$. We use the right side $z_{1:32}$ of the Equation 4.22 for the row index determination.

4.2.2.3 Attack on $T^{(1)}$

Attacking $T^{(1)}$ is performed in the same fashion as attacking $T^{(3)}$ and $T^{(4)}$:

$$\begin{aligned} x_{1:32} + T^{(1)} \cdot x_{33:64} + T^{(4)} \cdot z_{65:96} &= z_{1:32}, \\ \underbrace{x_{1:32}}_{\text{unknown}} + \underbrace{T^{(1)}}_{\text{secret key}} \cdot \underbrace{x_{33:64}}_{\text{known}} &= \underbrace{z_{1:32} + T^{(4)} \cdot z_{65:96}}_{\text{known}}. \end{aligned} \tag{4.23}$$

The secret submatrix is multiplied by a known vector $x_{33:64}$ computed using Equation 4.21. The right side of the Equation 4.23 is used for row index determination.

4.2.3 Extraction of the Central Map F

There are two possible approaches to extraction of the central map F with knowledge of S and T. The first one is extracting the central map F by eliminating T and S maps from the public key P. The second approach finds the central map F via known Rainbow inputs and outputs, using a system of linear equations. In this case, knowledge of the public key is not needed. Enough input and output data should be obtained while attacking the S and T maps.

4.2.4 Experimental Evaluation

We evaluate the proposed attack on a ChipWhisperer-Lite side-channel evaluation platform with a 32-bit STM32F303 microcontroller based on ARM Cortex-M4 core as a target. ChipWhisperer-Lite features an integrated 10-bit ADC with 105MS/s sampling rate and uses a synchronous sampling technique [127] for measurements of the target power consumption. We are attacking the implementation proposed in the NIST competition second round, with random data generated by a controlling PC.

For side-channel attack evaluation, we use the success rate (see subsection 2.4.1), i.e., the expected probability of attack successfully distinguishing the correct subkey. The presented results are based on 33 independent experiments and further averaged over 32 random subkey elements/rows.

Attack I targets a single 4-bit subkey and is directly applicable to non-zero subkeys only as described in subsection 4.2.1. Its success rate is therefore based on attacking non-zero subkeys only. *Attack II* is then used to distinguish between up to 32 matrix rows revealed by *Attack I*. Subkeys which *Attacks I and II* fail to reveal are then discovered using *Attack III*, which makes more precise predictions of a processed 32-bit word using previously discovered subkeys.

Attacks I and II (both targeting 4-bit value) have a success rate of 0.75/0.95 using approx. 280/475 power traces. *Attack III* has a success rate of 0.75/0.95 with one, three, or seven other known subkeys (i.e., targeting eight, 16, or 32 bits) using approx. 150/240, 90/140, or 40/70 power traces, respectively.

Attack III exhibits a better success rate than *Attacks I and II*, which is expected thanks to better signal-to-noise ratio given more precise power predictions.

4.2.5 Summary

In this section, we presented a side-channel attack on the Rainbow digital signature, NIST's third round candidate for post-quantum standard. We analyzed the 32-bit reference implementation and proposed a combined Correlation Power Analysis attack allowing extraction of a full secret key. We evaluated the proposed attack on a 32-bit microcontroller with ARM Cortex-M4 core and successfully extracted the secret key. Finally, we proposed an extension of a known masking scheme, that allows randomization of intermediate values used in the signing process, including the non-linear part, with no significant time or memory overhead.

4.3 Equivalent Keys as a Side-Channel Countermeasure for Multivariate Quadratic Signatures

In this section, we propose a novel side-channel countermeasure for multivariate cryptography based on private key randomization. We continue the work presented in [179] and introduce the concept of equivalent private keys, i.e., a class of private keys with the same public key. We propose a new private key to be generated prior to every signing. The adversary would then be forced to target the whole class instead of a fixed private key, making it difficult to mount a side-channel attack. We demonstrate our approach on the Rainbow algorithm. Since the UOV is a special (single-layered) case of Rainbow, our work is applicable to different multivariate cryptography algorithms as well.

4.3.1 Equivalent Key

The main idea of the equivalent key is to change the secret key SK into some other $\overline{\text{SK}}$, which is equivalent to the original one, i.e., SK and $\overline{\text{SK}}$ both have the same public key. This equivalency was previously used for storage, computation, and randomness reduction, where only the normal form of the key was generated [179]. Our countermeasure is based on the generation of the equivalent key before each signing process. We therefore substituted a fixed key with an equivalence class containing a huge number (as shown in Equation 4.40) of different, but equivalent, keys.

In our work, the equivalent key can be derived from the original key using two linear maps A and B, represented as matrices $\mathbf{A}$ and $\mathbf{B}$, which change all three maps contained in a secret key without changing their composition:

$$\begin{aligned} P &= S \circ F \circ T, \\ P &= S \circ (A^{-1} \circ A) \circ F \circ (B \circ B^{-1}) \circ T, \\ P &= (S \circ A^{-1}) \circ (A \circ F \circ B) \circ (B^{-1} \circ T). \end{aligned} \tag{4.24}$$

With this in mind, we define the secret key $\overline{\text{SK}}$ derived from SK as

$$\begin{aligned} \text{SK} &= (S^{-1}, F, T^{-1}), \\ \overline{\text{SK}} &= (A \circ S^{-1}, A \circ F \circ B, T^{-1} \circ B). \end{aligned} \tag{4.25}$$

The pair (A, B) is called Gauss sustaining transformation, where the representing matrices $\mathbf{A}$ and $\mathbf{B}$ cannot be chosen arbitrarily. They are composed with the central map F, which must maintain its special structure.

The necessary constraint is the invertibility of these matrices due to the signing process, where the inverse central map must be computed. Therefore, $\mathbf{A}$ and $\mathbf{B}$ must be regular; thus, $\det(\mathbf{A}) \neq 0$ and $\det(\mathbf{B}) \neq 0$. The compositions $A \circ S^{-1}$ and $T^{-1} \circ B$ can be computed simply as $\mathbf{A}\mathbf{S}^{-1}$ and $\mathbf{T}^{-1}\mathbf{B}$. The composition $A \circ F \circ B$ is discussed in the following subsubsections.

4.3.1.1 Composition $A \circ F$

Let $A : \mathbb{F}^m \to \mathbb{F}^m, \mathbf{h} \mapsto A(\mathbf{h}) = \mathbf{A}\mathbf{h}$, where $\mathbf{A} \in \mathbb{F}^{m \times m}$ is a regular matrix. The composition $A \circ F$ is then

$$(A \circ F)(\mathbf{x}) = \mathbf{A} \begin{pmatrix} f^{(v_1+1)}(\mathbf{x}) \\ f^{(v_1+2)}(\mathbf{x}) \\ \vdots \\ f^{(n)}(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} A_{1,i} f^{(v_1+i)}(\mathbf{x}) \\ \sum_{i=1}^{m} A_{2,i} f^{(v_1+i)}(\mathbf{x}) \\ \vdots \\ \sum_{i=1}^{m} A_{m,i} f^{(v_1+i)}(\mathbf{x}) \end{pmatrix}, \tag{4.26}$$

and $\forall k \in \{1, \ldots, m\}$:

$$[A \circ F]_k(\mathbf{x}) = \sum_{i=1}^{m} A_{k,i} \cdot f^{(v_1+i)}(\mathbf{x}) = \sum_{i=1}^{m} A_{k,i} \cdot (\mathbf{x}^\intercal \mathbf{F}^{(v_1+i)} \mathbf{x}) = \mathbf{x}^\intercal \sum_{i=1}^{m} (A_{k,i} \cdot \mathbf{F}^{(v_1+i)}) \mathbf{x}. \tag{4.27}$$

The composition is equivalent to a linear combination of $\mathbf{F}$ matrices. In the following equation, we show how arbitrary $\mathbf{A}$ affects the structure of the central map:

$$\sum_{i=1}^{m} A_{k,i} \cdot \mathbf{F}^{(v_1+i)} = \sum_{i=1}^{m} A_{k,i} \cdot \begin{pmatrix} \mathbf{F}_{1,1}^{(v_1+i)} & \mathbf{F}_{1,2}^{(v_1+i)} & \mathbf{F}_{1,3}^{(v_1+i)} \\ \mathbf{0} & \mathbf{F}_{2,2}^{(v_1+i)} & \mathbf{F}_{2,3}^{(v_1+i)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{4.28}$$

To maintain the special structure for the first layer ($k \in \{1, \ldots, o_1\}$), as described in Equation 4.7, we must fulfill the following restrictions:

$$\begin{aligned} \sum_{i=1}^{m} A_{k,i} \cdot \mathbf{F}_{1,3}^{(v_1+i)} &= \mathbf{0}, \\ \sum_{i=1}^{m} A_{k,i} \cdot \mathbf{F}_{2,2}^{(v_1+i)} &= \mathbf{0}, \\ \sum_{i=1}^{m} A_{k,i} \cdot \mathbf{F}_{2,3}^{(v_1+i)} &= \mathbf{0}. \end{aligned} \tag{4.29}$$

Since, $\forall i \in O_1 : \mathbf{F}_{1,3}^{(i)}, \mathbf{F}_{2,2}^{(i)}, \mathbf{F}_{2,3}^{(i)}$ are zero matrices, Equation 4.29 can be rewritten as

$$\begin{aligned} \sum_{i=o_1+1}^{m} A_{k,i} \cdot \mathbf{F}_{1,3}^{(v_1+i)} &= \mathbf{0}, \\ \sum_{i=o_1+1}^{m} A_{k,i} \cdot \mathbf{F}_{2,2}^{(v_1+i)} &= \mathbf{0}, \\ \sum_{i=o_1+1}^{m} A_{k,i} \cdot \mathbf{F}_{2,3}^{(v_1+i)} &= \mathbf{0}. \end{aligned} \tag{4.30}$$

This restriction can be trivially satisfied by the following condition:

$$(\forall k \in \{1, \ldots, o_1\})(\forall i \in \{o_1 + 1, \ldots, m\}) : A_{k,i} = 0. \tag{4.31}$$

Regarding the second layer, the arbitrary A maintains its polynomial structure by itself, and therefore, no further restrictions are necessary.

An applicable matrix $\mathbf{A} \in \mathbb{F}^{m \times m}$ is therefore

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{1,1} & \mathbf{0} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{pmatrix}, \tag{4.32}$$

where $\mathbf{A}_{1,1} \in \mathbb{F}^{o_1 \times o_1}, \mathbf{A}_{2,1} \in \mathbb{F}^{o_2 \times o_1}, \mathbf{A}_{2,2} \in \mathbb{F}^{o_2 \times o_2}$. Matrix $\mathbf{A}$ must be regular; therefore, $\mathbf{A}_{1,1}$ and $\mathbf{A}_{2,2}$ are arbitrary regular matrices and $\mathbf{A}_{1,2}$ is an (possibly singular) arbitrary matrix.

From the matrix $\mathbf{A}$, we deduce that the linear combination is performed separately in layers ($\mathbf{A}_{1,1}$ and $\mathbf{A}_{2,2}$), and quadratic polynomials from the first layer can be combined into the second layer ($\mathbf{A}_{2,1}$).

4.3.1.2 Composition $F \circ B$

Let $B : \mathbb{F}^n \to \mathbb{F}^n, \mathbf{x} \mapsto B(\mathbf{x}) = \mathbf{B}\mathbf{x}$, where $\mathbf{B} \in \mathbb{F}^{n \times n}$ is a regular matrix. The composition $F \circ B$ is then

$$(F \circ B)(\mathbf{x}) = F \circ (\mathbf{B}\mathbf{x}) = \begin{pmatrix} f^{(v_1+1)}(\mathbf{B}\mathbf{x}) \\ f^{(v_1+2)}(\mathbf{B}\mathbf{x}) \\ \vdots \\ f^{(n)}(\mathbf{B}\mathbf{x}) \end{pmatrix}, \tag{4.33}$$

and, $\forall k \in \{1, \ldots, m\}$:

$$[(F \circ B)]_k(\mathbf{x}) = f^{(v_1+k)}(\mathbf{B}\mathbf{x}) = (\mathbf{B}\mathbf{x})^\intercal \mathbf{F}^{(v_1+k)} \mathbf{B}\mathbf{x} = \mathbf{x}^\intercal (\mathbf{B}^\intercal \mathbf{F}^{(v_1+k)} \mathbf{B})\mathbf{x}. \tag{4.34}$$

This composition changes every quadratic polynomial separately. Before we consider the restrictions on the matrix $\mathbf{B}$, let us first introduce the following lemma.

Lemma 4.3.1. Every quadratic form can be represented as an upper triangular matrix.

Proof. Let $q \in \mathbb{F}[\mathbf{x}]$ be a quadratic form represented by matrix $\mathcal{Q}$, then $\mathcal{U}$ is an upper triangular representation of q, where $\mathcal{U}$ is

$$\forall i, j \in \{1, \ldots, n\} : \mathcal{U}_{i,j} = \begin{cases} \mathcal{Q}_{i,i} & i = j, \\ \mathcal{Q}_{i,j} + \mathcal{Q}_{j,i} & i < j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.35}$$

Then,

$$q(\mathbf{x}) = \mathbf{x}^\intercal \mathcal{Q}\mathbf{x} = \sum_{i,j \in \{1,\ldots,n\}} \mathcal{Q}_{i,j} x_i x_j = \sum_{i \in \{1,\ldots,n\}} \mathcal{Q}_{i,i} x_i^2 + \sum_{\substack{i,j \in \{1,\ldots,n\} \\ i<j}} (\mathcal{Q}_{i,j} + \mathcal{Q}_{j,i}) x_i x_j = \mathbf{x}^\intercal \mathcal{U}\mathbf{x}. \tag{4.36}$$

□

Let $\triangledown : \mathbb{F}^{n \times n} \to \mathbb{F}^{n \times n}; \mathcal{Q} \mapsto \mathcal{U}$ be a function, where $\mathcal{U}$ is an upper triangular matrix, such that $\forall \mathbf{x} \in \mathbb{F}^n : \mathbf{x}^\intercal \mathcal{Q}\mathbf{x} = \mathbf{x}^\intercal \mathcal{U}\mathbf{x}$.

In Equation 4.37, we show how matrix $\mathbf{B}$ changes the structure of a polynomial in the central map. We used the same partitioning of the matrix $\mathbf{B}$ as we used for the matrix $\mathbf{F}^{(k)}$. Size compatibility for the matrix multiplication is guaranteed thanks to the symmetry of block sizes. We further applied the function $\triangledown$, as it does not change the concerned polynomial and it helps us with the wanted structure. It ensures the upper

triangular submatrices on the diagonal, which are required by the definition of Rainbow.

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} & \mathbf{B}_{1,3} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} & \mathbf{B}_{2,3} \\ \mathbf{B}_{3,1} & \mathbf{B}_{3,2} & \mathbf{B}_{3,3} \end{pmatrix},$$

$$k \in O_1 : \triangledown(\mathbf{B}^\intercal \mathbf{F}^{(k)} \mathbf{B}) = \begin{pmatrix} \mathbf{F}'^{(k)}_{1,1} & \mathbf{F}'^{(k)}_{1,2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{4.37}$$

$$k \in O_2 : \triangledown(\mathbf{B}^\intercal \mathbf{F}^{(k)} \mathbf{B}) = \begin{pmatrix} \mathbf{F}'^{(k)}_{1,1} & \mathbf{F}'^{(k)}_{1,2} & \mathbf{F}'^{(k)}_{1,3} \\ \mathbf{0} & \mathbf{F}'^{(k)}_{2,2} & \mathbf{F}'^{(k)}_{2,3} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

Solving this system of equations with matrix $\mathbf{B}$ as a variable for arbitrary central map $\mathbf{F}$, where the right sides of the equations are zeroes, we obtain:

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_{1,1} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} & \mathbf{0} \\ \mathbf{B}_{3,1} & \mathbf{B}_{3,2} & \mathbf{B}_{3,3} \end{pmatrix}, \tag{4.38}$$

where non-zero submatrices are arbitrary submatrices, except that matrix $\mathbf{B}$ must be regular, and therefore, submatrices $\mathbf{B}_{1,1}, \mathbf{B}_{2,2}, \mathbf{B}_{3,3}$ must be regular.

For an insight into what is happening to the vector $\mathbf{x}$ (which elements are the variables in the central map), we can partition $\mathbf{x}$ into three different types of variables: vinegar variables of the first layer ($\mathbf{x}_1^v \in \mathbb{F}^{v_1}$), oil variables of the first layer ($\mathbf{x}_1^o \in \mathbb{F}^{o_1}$), and oil variables of the second layer ($\mathbf{x}_2^o \in \mathbb{F}^{o_2}$). If we apply matrix $\mathbf{B}$ to the vector $\mathbf{x}$, we obtain:

$$\mathbf{B}\mathbf{x} = \begin{pmatrix} \mathbf{B}_{1,1} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} & \mathbf{0} \\ \mathbf{B}_{3,1} & \mathbf{B}_{3,2} & \mathbf{B}_{3,3} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1^v \\ \mathbf{x}_1^o \\ \mathbf{x}_2^o \end{pmatrix} = \begin{pmatrix} \mathbf{B}_{1,1}\mathbf{x}_1^v \\ \mathbf{B}_{2,1}\mathbf{x}_1^v + \mathbf{B}_{2,2}\mathbf{x}_1^o \\ \mathbf{B}_{3,1}\mathbf{x}_1^v + \mathbf{B}_{3,2}\mathbf{x}_1^o + \mathbf{B}_{3,3}\mathbf{x}_2^o \end{pmatrix}. \tag{4.39}$$

In this situation, the variables to solve are not defined only by vector $\mathbf{x}$, but the vector $\mathbf{B}\mathbf{x}$. To be able to sign, it is necessary not to mix the vinegar variable with the oil variables of each layer. At the beginning of the signing process, we used $\mathbf{B}_{1,1}\mathbf{x}_1^v$ instead of $\mathbf{x}_1^v$. Vector $\mathbf{x}_1^v$ is randomly generated, thus known. We can easily compute $\mathbf{B}_{1,1}\mathbf{x}_1^v$. Next, the first layer is solved, where we computed $\mathbf{B}_{2,1}\mathbf{x}_1^v + \mathbf{B}_{2,2}\mathbf{x}_1^o$ instead of $\mathbf{x}_1^o$. We can immediately substitute $\mathbf{B}_{2,1}\mathbf{x}_1^v$, which is already known. We still obtain a system of the linear equations that has only variables of the vector $\mathbf{x}_1^o$. In the end, the second layer is solved. We can first substitute $\mathbf{B}_{3,1}\mathbf{x}_1^v + \mathbf{B}_{3,2}\mathbf{x}_1^o$, and again, we obtain only a system of linear equations. No oil and vinegar variables (of the same layer) are mixed. This was only an insight into why the inversion of the central map can be still computed. In the implementation, matrix $\mathbf{B}$ is incorporated in the central map (as a generator of the equivalent key).

4.3.1.3 Analysis of an Equivalent Key

In this subsubsection, we discuss the basic properties of the equivalent key. We concentrate only on equivalent keys that can be generated according to Equation 4.25, with the maps A

and B defined in Equation 4.32 and Equation 4.38. We start with the number of equivalent keys, and we calculate entropy where applicable. Then, we state the benefits of our masking scheme. For the analysis in this section, we considered two sets of Rainbow parameters:

- **Ia** (128-bit security): $\mathbb{F}_q = GF(16), v_1 = 36, o_1 = o_2 = 32$;
- **Vc** (256-bit security): $\mathbb{F}_q = GF(256), v_1 = 96, o_1 = 36$ and $o_2 = 64$.

We note that these parameters, while suggested in the NIST standardization process, are probably already inadequate due to the recently presented attacks [18].

Lemma 4.3.2. Let $M_{l,s}(\mathbb{F}_q) = \{l \times s \text{ matrices over } \mathbb{F}_q\}$ be a set. The cardinality of the set is $|M_{l,s}(\mathbb{F}_q)| = q^{l \cdot s}$.

Lemma 4.3.3. Let $M_n(\mathbb{F}_q) = \{n \times n \text{ matrices over } \mathbb{F}_q\}$, with the matrix multiplication, be the full linear monoid. The order of the monoid is $|M_n(\mathbb{F}_q)| = q^{n^2}$.

Lemma 4.3.4. Let $GL_n(\mathbb{F}_q) = \{n \times n \text{ invertible matrices over } \mathbb{F}_q\}$, with the matrix multiplication, be the general linear group. The order is $|GL_n(\mathbb{F}_q)| = \prod_{k=0}^{n-1}(q^n - q^k) = q^{n^2}(q^{-n}; q)_n$. [2]

The proofs of these three lemmas are well known [165].

Theorem 4.3.5. The number of different equivalent keys, over a fixed secret key SK, is

$$q^{v_1(v_1+o_1+o_2)+2(o_1^2+o_1o_2+o_2^2)}(q^{-v_1}; q)_{v_1}((q^{-o_1}; q)_{o_1})^2((q^{-o_2}; q)_{o_2})^2. \tag{4.40}$$

This is an exact number, where the left part $q^{(\dots)}$ describes the number of different combinations of matrices $\mathbf{A}$ and $\mathbf{B}$. The remaining part counts for the cases where $\mathbf{A}$ and $\mathbf{B}$ are regular. For GF(16), its value is ≈ 0.7092, and for GF(256) it is ≈ 0.9805. This value is dependent on the values v_1, o_1, o_2, but it converges with these parameters extremely fast. For GF(16), the difference between $v_1 = o_1 = o_2 = 10$ and $v_1 = o_1 = o_2 = 1000$ is less then 10^{-12}.

Proof. (Number of different equivalent keys) We show that the number of equivalent keys is the number of possible non-equal matrices $\mathbf{A}$ multiplied by the number of possible non-equal matrices $\mathbf{B}$. This holds because both matrices are applied to different parts of the secret key (maps S and T), so they cannot interfere with each other, and they are both generated independently. We discuss the number of keys for matrix $\mathbf{A}$ only, and the same procedure can be applied to matrix $\mathbf{B}$. Let $\mathbb{A}$ be the set of matrices with the structure defined in Equation 4.32 and a regular matrix $\mathbf{S} \in \mathrm{GL}_m(\mathbb{F}_q)$ be the representation of the linear map S.

[2] q-Pochhammer symbol: $(a; q)_n := \prod_{k=0}^{n-1}(1 - aq^k), n > 0$, defined inter alia in [8]. For example: the number of 10×10 regular matrices over GF(16) is $|GL_{10}(\mathbb{F}_{16})| \approx 0.9336 \cdot 16^{100}$, where $(16^{-10}; 16)_{10} \approx 0.9336$. Symbol $(q^{-n}, q)_n$ denotes the ratio between regular matrices $n \times n$ and all matrices $n \times n$ over $\mathbb{F}_q$.

First, we show that the lower bound of the number of equivalent keys generated by $\mathbb{A}$ is equal to the cardinality of the set $\mathbb{A}$. This is because a regular matrix $\mathbf{S}$ is multiplied by regular matrices from the set $\mathbb{A}$. Thus, two distinct matrices $\mathbf{A}, \mathbf{A}' \in \mathbb{A} \subset \mathrm{GL}$ cannot generate the same product matrix:

$$\forall \mathbf{A}, \mathbf{A}' \in \mathrm{GL}_m(\mathbb{F}_q) : \mathbf{A} \neq \mathbf{A}' \implies \mathbf{AS} \neq \mathbf{A}'\mathbf{S}. \tag{4.41}$$

Therefore, each distinct matrix in $\mathbb{A}$ generates a different equivalent key; hence, the cardinality of the set $\mathbb{A}$ is the lower bound of the number of different equivalent keys generated by $\mathbb{A}$.

Second, we show that the cardinality of the set $\mathbb{A}$ is also the upper bound of the number of different equivalent keys generated by $\mathbb{A}$, since every equivalent key can be reached by a multiplication with a single $\mathbf{A} \in \mathbb{A}$. Let us examine a product of two matrices $\mathbf{A}, \mathbf{A}' \in \mathbb{A}$:

$$\mathbf{AA}' = \begin{pmatrix} \mathbf{A}_{1,1}\mathbf{A}'_{1,1} & \mathbf{0} \\ \mathbf{A}_{2,1}\mathbf{A}'_{1,1} + \mathbf{A}_{2,2}\mathbf{A}'_{2,1} & \mathbf{A}_{2,2}\mathbf{A}'_{2,2} \end{pmatrix}. \tag{4.42}$$

The matrix $\mathbf{AA}'$ has the same structure as the matrices $\mathbf{A}$ and $\mathbf{A}'$. On the diagonal, we have the products of the elements of GL, which also result in an element of the GL, and the submatrix $[\mathbf{AA}']_{2,1}$ can be an arbitrary matrix, thus $\mathbf{AA}' \in \mathbb{A}$. In other words, every key reachable by subsequent multiplications with two matrices from $\mathbb{A}$ can be reached by a single multiplication with some matrix from $\mathbb{A}$:

$$(\forall \mathbf{A}, \mathbf{A}' \in \mathbb{A})(\exists \mathbf{A}'' \in \mathbb{A}) : \mathbf{A}'(\mathbf{AS}) = \mathbf{A}''\mathbf{S}. \tag{4.43}$$

Therefore, the upper bound of the number of equivalent keys generated by $\mathbb{A}$ is the size of the set $\mathbb{A}$.

The previous statements imply that the number of equivalent keys generated by $\mathbb{A}$ is equal to the size of the set $\mathbb{A}$. Every matrix $\mathbf{A} \in \mathbb{A}$ has four parts: two submatrices from GL, one from $M_{o_1,o_2}(\mathbb{F}_q)$, and a zero submatrix. All matrices are independent, so the total number of matrices is the product of the numbers of different submatrices:

$$\begin{aligned} &|\mathrm{GL}_{o_1}(\mathbb{F}_q)| \cdot |\mathrm{GL}_{o_2}(\mathbb{F}_q)| \cdot |M_{o_1,o_2}(\mathbb{F}_q)| \cdot 1 = \\ &= q^{(o_1^2+o_2^2+o_1o_2)}(q^{-o_1};q)_{o_1}(q^{-o_2};q)_{o_2}. \end{aligned} \tag{4.44}$$

The number of different matrices $\mathbf{B}$ can be computed in a similar fashion:

$$\begin{aligned} &|\mathrm{GL}_{v_1}(\mathbb{F}_q)| \cdot |\mathrm{GL}_{o_1}(\mathbb{F}_q)| \cdot |\mathrm{GL}_{o_2}(\mathbb{F}_q)| \cdot |M_{v_1,o_1}(\mathbb{F}_q)| \cdot |M_{v_1,o_2}(\mathbb{F}_q)| \cdot |M_{o_1,o_2}(\mathbb{F}_q)| = \\ &= q^{(v_1^2+o_1^2+o_2^2+v_1o_1+v_1o_2+o_1o_2)}(q^{-v_1};q)_{v_1}(q^{-o_1};q)_{o_1}(q^{-o_2};q)_{o_2}. \end{aligned} \tag{4.45}$$

By multiplying the results in Equation 4.44 and Equation 4.45, we obtain the total number of equivalent keys, as stated in Equation 4.40. □

The number of equivalent keys is approximately $16^{9744} \cdot 0.7097 \approx 2^{38976}$ and $256^{34208} \cdot 0.9805 \approx 2^{273664}$ for the Ia and Vc parameters, respectively. If the matrices $\mathbf{A}$ and $\mathbf{B}$ are

generated from a uniform distribution, the entropy of the generated equivalent key for the Ia and Vc parameters is ≈ 38976 Sh and ≈ 273664 Sh, respectively.

The beneficial property of an equivalent key scheme is an option to forget the original key and keep only an equivalent key. We were able to generate an equivalent key from an already-computed equivalent key. The equivalent key (or more of them) can be generated and prepared at any time prior to the signing process. The signing process itself then has no overhead.

4.3.2 Efficient Implementation

In subsection 4.3.1, we discuss the equivalent key defined by matrices $\mathbf{A}$ and $\mathbf{B}$. However, we propose choosing only a subset of these matrices with respect to:

- Calculation performance;
- Amount of fresh randomness necessary;
- Entropy of the generated equivalent key;
- Implementation size and simplicity;
- Total number of keys equivalent to each public key.

With respect to the aforementioned desires, we propose using the following submatrices to generate the equivalent keys:

$$\mathcal{M}_n := \left\{ \mathbf{M} = \begin{pmatrix} 1 & m_1 & 0 & \dots & 0 \\ 0 & 1 & m_2 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_n & 0 & 0 & \dots & 1 \end{pmatrix} \mid \det(\mathbf{M}) \neq 0 \right\}, \tag{4.46}$$

where $m_1, \dots, m_n \in \mathbb{F}_q$, and the condition for regularity is

$$\det(\mathbf{M}) \neq 0 \iff \prod_{i=1}^{n} m_i \neq (-1)^{n+1}. \tag{4.47}$$

The equivalent key generator is a tuple $(\mathbf{A}, \mathbf{B})$ defined as

$$\mathbf{A} := \begin{pmatrix} \mathbf{A}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^{(2)} \end{pmatrix}, \ \mathbf{B} := \begin{pmatrix} \mathbf{B}^{(1)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^{(2)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}^{(3)} \end{pmatrix}, \tag{4.48}$$

where $\mathbf{A}^{(1)}, \mathbf{B}^{(2)} \in \mathcal{M}_{o_1}, \mathbf{A}^{(2)}, \mathbf{B}^{(3)} \in \mathcal{M}_{o_2}, \mathbf{B}^{(1)} \in \mathcal{M}_{v_1}$. The proposed generator form is further justified in subsubsection 4.3.2.1. The generator matrices are defined as

$$\mathbf{A}_{k,l} = \begin{cases} 1 & k = l, \\ a_k & (k \in \{1, \ldots, o_1\}) \wedge (l = |k+1|_{o_1}), \\ a_k & (k \in \{o_1+1, \ldots, m\}) \wedge (l = |k-o_1+1|_{o_2} + o_1), \\ 0 & \text{otherwise}, \end{cases}$$
$$\mathbf{B}_{k,l} = \begin{cases} 1 & k = l, \\ b_k^{(1)} & (k \in \{1, \ldots, v_1\}) \wedge (l = |k+1|_{v_1}), \\ b_{k-v_1}^{(2)} & (k \in \{v_1+1, \ldots, v_2\}) \wedge (l = |k-v_1+1|_{o_1} + v_1), \\ b_{k-v_2}^{(3)} & (k \in \{v_2+1, \ldots, n\}) \wedge (l = |k-v_2+1|_{o_2} + v_2), \\ 0 & \text{otherwise}, \end{cases} \tag{4.49}$$

where $|a|_b := (a - 1 \bmod b) + 1$ is the modulo operation with offset[3]. Matrices $\mathbf{A}$ and $\mathbf{B}$ can be stored as vectors of their non-trivial values:

$$(a_1, a_2, \ldots, a_m), (b_1^{(1)}, \ldots, b_{v_1}^{(1)}, b_1^{(2)}, \ldots, b_{o_1}^{(2)}, b_1^{(3)}, \ldots, b_{o_2}^{(3)}). \tag{4.50}$$

For further analysis, in Equation 4.51, we show the probability that the randomly generated matrix from the definition in Equation 4.46 (i.e., the matrix with generated values $m_1, \ldots, m_n$) over $\mathbb{F}_q$ is regular. In Equation 4.51, we omit the set $\{1, \ldots, n\}$ in all quantifiers $\forall i \in \{1, \ldots, n\}$ and $\exists i \in \{1, \ldots, n\}$ since it is always identical, and we write only $\forall i$ and $\exists i$ for simplicity.

$$\begin{aligned} &\mathrm{P}(\det(\mathbf{M}) \neq 0 \mid \mathbf{M} \in \mathcal{M}_n) = \\ &= \mathrm{P}\left(\exists i : m_i = 0 \vee \left(\forall i : m_i \neq 0 \wedge \prod_{i=1}^{n} m_i \neq (-1)^{n+1}\right)\right) \\ &= \mathrm{P}\left(\exists i : m_i = 0\right) + \mathrm{P}\left(\forall i : m_i \neq 0 \wedge \prod_{i=1}^{n} m_i \neq (-1)^{n+1}\right) \\ &= 1 - \mathrm{P}(\forall i : m_i \neq 0) + \mathrm{P}\left(\prod_{i=1}^{n} m_i \neq (-1)^{n+1} \;\middle|\; \forall i : m_i \neq 0\right) \cdot \mathrm{P}(\forall i : m_i \neq 0) \\ &= 1 - \left(\frac{q-1}{q}\right)^n + \left(\frac{q-2}{q-1}\right)\left(\frac{q-1}{q}\right)^n \\ &= 1 - \left(\frac{1}{q-1}\right)\left(\frac{q-1}{q}\right)^n. \end{aligned} \tag{4.51}$$

The non-zero values of the submatrices in $\mathbf{A}$ and $\mathbf{B}$ are to be generated independently from a uniform random distribution, so we can easily express the probability, e.g., a ran-

[3]E.g., it holds that $|m|_m = m$ and $|m+1|_m = 1$. Indexing from zero would result in using a regular modulo operation instead.

domly generated matrix $\mathcal{M}_{32}$ over $GF(16)$ is regular with probability 99.15% and matrix $\mathcal{M}_{64}$ over $GF(256)$ with probability 99.69%.

Using the regularity probability, we can express the cardinality of $\mathcal{M}_n$. This can be performed by multiplying the number of all possible matrices by the probability of each matrix over $\mathbb{F}_q$ being regular:

$$|\mathcal{M}_n| = q^n - (q-1)^{n-1}. \tag{4.52}$$

4.3.2.1 Justification of the Selected Generators

In this subsubsection, we justify the proposed form of our efficient equivalent key generators with respect to the desires mentioned in subsection 4.3.2.

We propose the form of generators in Equation 4.46, i.e., the ones on the diagonal and the only non-trivial elements adjacent to it, so that vector–matrix (and analogically, matrix–matrix) multiplication can be efficiently performed as described by formula:

$$\mathbf{M} \in \mathcal{M}_n : \mathbf{M} \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} = (\mathbf{I} + \widetilde{\mathbf{M}}) \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} + \begin{pmatrix} m_1 g_2 \\ m_2 g_3 \\ \vdots \\ m_n g_1 \end{pmatrix}. \tag{4.53}$$

Based on Equation 4.53, the vector–matrix multiplication $\mathbf{M} \cdot \mathbf{g}$ can be performed simply by copying the vector $\mathbf{g}$ to the result variable, then cyclically shifting the vector $\mathbf{g}$ and adding an elementwise product of the vector $\mathbf{g}$ and the vector of non-trivial values of the matrix $\mathbf{M}$ (as listed in Equation 4.50).

Next, we discuss a number of different equivalent keys for our efficient generators. This is equal to the number of possible distinct linear maps of generators, in other words, the number of different matrices that can be obtained by multiplying matrices in $\mathcal{M}_n$ by each other. We believe that these matrices generate the whole group GL_n for $n > 2$, but we were not able to prove this claim in general[4]. Assuming that the following formula holds over a fixed $\mathbb{F}_q$ and for every integer $n > 2$:

$$\begin{aligned} &\forall \mathbf{M} \in \mathrm{GL}_n, \exists k \in \mathbb{N}, \exists \mathbf{M}_1, \ldots, \mathbf{M}_k \in \mathcal{M}_n : \prod_{i=1}^{k} \mathbf{M}_i = \mathbf{M} \\ &\implies \mathrm{card}(\{\text{matrices generated from } \mathcal{M}_n\}) = \mathrm{ord}(\mathrm{GL}_n), \end{aligned} \tag{4.54}$$

the number of equivalent keys using the proposed efficient generators, with the possibility of generating an equivalent key from another equivalent key, would be

$$N = q^{v_1^2 + 2o_1^2 + 2o_2^2} (q^{-v_1}; q)_{v_1} (q^{-o_1}; q)_{o_1}^2 (q^{-o_2}; q)_{o_2}^2. \tag{4.55}$$

[4]We managed to prove the claim for matrices 3×3, 4×4, 5×5 over $\mathbb{Z}_2$, then for 3×3, 4×4 over $\mathbb{Z}_3$, and also, for matrices 3×3 over $\mathbb{Z}_5$ by brute force.

For parameters Ia and Vc, respectively, the number of transitively reachable equivalent keys (i.e., generating the equivalent key from another equivalent key) would be approximately $16^{5392} \cdot 0.7092 \approx 2^{21567}$ and $256^{20000} \cdot 0.9805 \approx 2^{160000}$, respectively.

The number of possible distinct equivalent keys after one generating process is $|\mathcal{M}_{v_1}| \cdot |\mathcal{M}_{o_1}|^2 \cdot |\mathcal{M}_{o_2}|^2$; for parameters Ia and Vc, respectively, the number of possible equivalent keys after one generation is approximately 2^{656} and 2^{2368}, respectively. Assuming the equivalent key generators are sampled from a uniform distribution, the entropy of the next-generated equivalent key is approximately 656 Sh and 2368 Sh, respectively. Given this entropy, we believe that the probability of the signer using the same equivalent key multiple times is negligible. The number of distinct equivalent keys is summarized in Table 4.4.

Table 4.4: Summary of equivalent key variants.

Generator	$\mathbf{Log_2}$ of Number of Equivalent Keys			
	Single Generated Key		Transitively Reachable	
	Ia	Vc	Ia	Vc
General	38,976	273,664	38,976	273,664
Efficient	656	2368	21,567	160,000

We further discuss the number of distinct linear maps generated in equivalent keys, as these are typical targets in a side-channel attack scenario (section 4.2, [129]). The number of distinct maps S generated by the proposed efficient generator A over a single class of equivalent keys is

$$q^{o_1^2+o_2^2}(q^{-o_1};q)_{o_1}(q^{-o_2};q)_{o_2}. \tag{4.56}$$

For parameters Ia and Vc, respectively, the number of transitively reachable linear maps S is approximately $16^{2048} \cdot 0.8716 \approx 2^{8192}$ and $256^{5392} \cdot 0.9922 \approx 2^{43136}$, respectively. After one generating process, the number of different linear maps S is $|\mathcal{M}_{o_1}| \cdot |\mathcal{M}_{o_2}|$, that is almost 2^{256} for the Ia parameters and almost 2^{800} for the Vc parameters.

Lastly, we discuss the required fresh randomness. The generators $\mathbf{A}$ and $\mathbf{B}$ defined in Equation 4.32 and 4.38 use a quadratic number of random elements in regard to parameters m and n. For Rainbow parameters Ia and Vc, respectively, this corresponds to $\approx$ 39 kb and $\approx$ 274 kb of fresh randomness, respectively. The number of required random elements is linear for our efficient generators defined in Equation 4.48, leaving us with only 656 bits and 2368 bits of fresh randomness, respectively. This amount of randomness refers to the case when the matrices $\mathbf{A}$ and $\mathbf{B}$ are successfully generated according to the requirements, i.e., the generated matrices are regular[5].

[5]The probability of the randomly generated matrices $\mathbf{A}$ and $\mathbf{B}$ being regular is 96 % for the Ia parameters and 98.4 % for the Vc parameters.

4.3.2.2 Efficient Computation of Equivalent Keys

Compositions $A \cdot S^{-1}$ and $T^{-1} \cdot B$ can be computed in the same way as the multiplication in Equation 4.53.

The composition of $A \circ F$, where A is represented as $\mathbf{A}$ in Equation 4.48, is:

$$\forall k \in \{1, \ldots, o_1\} : [A \circ F]_k \leftrightarrow \mathbf{F}^{(v_1+k)} + a_k \mathbf{F}^{(v_1+|k+1|_{o_1})}, \tag{4.57}$$

$$\forall k \in \{o_1+1, \ldots, m\} : [A \circ F]_k \leftrightarrow \mathbf{F}^{(v_1+k)} + a_k \mathbf{F}^{(v_1+|k-o_1+1|_{o_2}+o_1)}. \tag{4.58}$$

The composition of $F \circ B$, where B is represented as $\mathbf{B}$ in Equation 4.48, is

$$\begin{aligned} k \in \{v_1, \ldots, n\} : [F \circ B]_k \leftrightarrow \triangledown(\mathbf{B}^\intercal \mathbf{F}^{(k)} \mathbf{B}) = \\ = \begin{pmatrix} \triangledown\left((\mathbf{B}^{(1)})^\intercal \mathbf{F}_{1,1}^{(k)} \mathbf{B}^{(1)}\right) & (\mathbf{B}^{(1)})^\intercal \mathbf{F}_{1,2}^{(k)} \mathbf{B}^{(2)} & (\mathbf{B}^{(1)})^\intercal \mathbf{F}_{1,3}^{(k)} \mathbf{B}^{(3)} \\ \mathbf{0} & \triangledown\left((\mathbf{B}^{(2)})^\intercal \mathbf{F}_{2,2}^{(k)} \mathbf{B}^{(2)}\right) & (\mathbf{B}^{(2)})^\intercal \mathbf{F}_{2,3}^{(k)} \mathbf{B}^{(3)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}. \end{aligned} \tag{4.59}$$

Upper triangular matrices $\triangledown\left((\mathbf{B}^{(1)})^\intercal \mathbf{F}_{1,1}^{(k)} \mathbf{B}^{(1)}\right)$ and $\triangledown\left((\mathbf{B}^{(2)})^\intercal \mathbf{F}_{2,2}^{(k)} \mathbf{B}^{(2)}\right)$ in Equation 4.59 can be efficiently evaluated as described in subsubsection 4.3.2.3. The remaining parts of Equation 4.59 consist of matrix multiplications $(\mathbf{B}^{(u)})^\intercal \mathbf{F}_{u,v}^{(k)} \mathbf{B}^{(v)}$, where $(u, v) \in \{(1,2), (1,3), (2,3)\}$. This expression can be computed as:

$$\begin{aligned} &[(\mathbf{B}^{(u)})^\intercal \mathbf{F}_{u,v}^{(k)} \mathbf{B}^{(v)}]_{r,c} = \\ =&[\mathbf{F}_{u,v}^{(k)}]_{r,c} + b_{|r-1|_s}^{(u)} [\mathbf{F}_{u,v}^{(k)}]_{|r-1|_s,c} + b_{|c-1|_w}^{(v)} [\mathbf{F}_{u,v}^{(k)}]_{r,|c-1|_w} + b_{|r-1|_s}^{(u)} b_{|c-1|_w}^{(v)} [\mathbf{F}_{u,v}^{(k)}]_{|r-1|_s,|c-1|_w}, \end{aligned} \tag{4.60}$$

where $\mathbf{B}^{(u)} \in \mathcal{M}_s$, $\mathbf{B}^{(v)} \in \mathcal{M}_w$, and $\mathbf{F}_{u,v}^{(k)} \in \mathbb{F}^{s \times w}$ is an arbitrary matrix. If the elements are stored sequentially by index k, the computation can be accelerated using word-level parallelism and vector processing.

4.3.2.3 Algorithm for Upper Triangular Matrices' Evaluation

Algorithm 4.1 Computation of $[\triangledown\left((\mathbf{B}^{(u)})^\intercal \mathbf{F}^{(k)}_{u,u}\mathbf{B}^{(u)}\right)]_{r,c}$.

input: matrix $\mathbf{B}^{(u)} \in \mathcal{M}_s$ as vector $(b_1, \ldots, b_s), u \in \{1, 2\}$
upper triangular matrix $\mathbf{F}^{(k)}_{u,u} \in \mathbb{F}^{s\times s}, k \in O_1 \cup O_2$
integers $r, c \in \{1, \ldots, s\}$, where $r \leq c$
output: element $[\triangledown\left((\mathbf{B}^{(u)})^\intercal \mathbf{F}^{(k)}_{u,u}\mathbf{B}^{(u)}\right)]_{r,c}$

1: $\mathbf{F}_{i,j} := [\mathbf{F}^{(k)}_{u,u}]_{i,j}, \forall i, j \in \{1, \ldots, s\}$ (for brevity)
2: $m := c - r$
3: $r_1 := |r - 1|_s$
4: $c_1 := |c - 1|_s$
5: $t := \mathbf{F}_{r,c} + b_{r_1} b_{c_1} (r \neq 1 \ ? \ \mathbf{F}_{r_1,c_1} : \mathbf{F}_{c_1,r_1})$
6: **if** $m \neq s - 1$ **then**
7: $\quad t := t + b_{r_1} (r \neq 1 \ ? \ \mathbf{F}_{r_1,c} : \mathbf{F}_{c,s})$
8: **else if** $\mathrm{char}(\mathbb{F}) \neq 2$ **then**
9: $\quad t := t + 2b_s \mathbf{F}_{s,s}$
10: **end if**
11: **if** $m > 1$ **then**
12: $\quad t := t + b_{c_1} \mathbf{F}_{r,c_1}$
13: **else if** $m = 1 \wedge \mathrm{char}(\mathbb{F}) \neq 2$ **then**
14: $\quad t := t + 2b_r \mathbf{F}_{r,r}$
15: **end if**
16: **return** t

Algorithm 4.1 was deduced as described in the following text. First, we rewrote output $\triangledown\left((\mathbf{B}^{(u)})^\intercal \mathbf{F}^{(k)}_{u,u}\mathbf{B}^{(u)}\right)$, without writing free variables u and k, as $\triangledown(\mathbf{B}^\intercal\mathbf{F}\mathbf{B})$, where $\mathbf{F}$ is an upper triangular matrix in $\mathbb{F}^{s\times s}$ and $\mathbf{B}$ is a matrix from the set $\mathcal{M}_s \subset \mathbb{F}^{s\times s}$, which is defined using the vector of its non-trivial values $(b_1, \ldots, b_s)$. Then,

$$[\triangledown(\mathbf{B}^\intercal\mathbf{F}\mathbf{B})]_{r,c} = \begin{cases} [\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{r,r} & r = c, \\ [\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{r,c} + [\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{c,r} & r < c, \\ 0 & r > c. \end{cases} \tag{4.61}$$

By expanding matrix multiplication, we obtain

$$[\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{r,c} = \mathbf{F}_{r,c} + b_{|r-1|_s}\mathbf{F}_{|r-1|_s,c} + b_{|c-1|_s}\mathbf{F}_{r,|c-1|_s} + b_{|r-1|_s}b_{|c-1|_s}\mathbf{F}_{|r-1|_s,|c-1|_s}. \tag{4.62}$$

Specifically, for $r = c$, we obtain

$$[\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{r,r} = \mathbf{F}_{r,r} + b_{|r-1|_s}(\mathbf{F}_{|r-1|_s,r} + \mathbf{F}_{r,|r-1|_s}) + b^2_{|r-1|_s}\mathbf{F}_{|r-1|_s,|r-1|_s}. \tag{4.63}$$

One term of $(\mathbf{F}_{|r-1|_s,r} + \mathbf{F}_{r,|r-1|_s})$ is zero, as matrix $\mathbf{F}$ is upper triangular and $|r-1|_s \neq r$.

The second part of Equation 4.61 with the condition $r < c$ is

$$\begin{aligned}[\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{r,c} + [\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{c,r} = \mathbf{F}_{r,c} + b_{|r-1|_s}(\mathbf{F}_{|r-1|_s,c} + \mathbf{F}_{c,|r-1|_s}) + \\ + b_{|c-1|_s}(\mathbf{F}_{r,|c-1|_s} + \mathbf{F}_{|c-1|_s,r}) + b_{|r-1|_s}b_{|c-1|_s}(\mathbf{F}_{|r-1|_s,|c-1|_s} + \mathbf{F}_{|c-1|_s,|r-1|_s}).\end{aligned} \tag{4.64}$$

In all sets of parentheses, there is at least one term equal to zero (note that $\mathbf{F}$ is an upper triangular matrix), except for two cases. The exception is iff the indices are the same. The first case is when $|r-1|_s = c \implies c = r + s - 1$. This condition can occur only when $r = 1 \wedge c = s$:

$$\begin{aligned}[\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{1,s} + [\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{s,1} &= \mathbf{F}_{1,s} + 2b_s\mathbf{F}_{s,s} + b_{s-1}(\mathbf{F}_{1,s-1} + \mathbf{F}_{s-1,1}) + b_sb_{s-1}(\mathbf{F}_{s,s-1} + \mathbf{F}_{s-1,s}) = \\ &= \mathbf{F}_{1,s} + 2b_s\mathbf{F}_{s,s} + b_{s-1}\mathbf{F}_{1,s-1} + b_sb_{s-1}\mathbf{F}_{s-1,s}.\end{aligned} \tag{4.65}$$

The second case is when $r = |c-1|_s \implies c = r + 1$:

$$\begin{aligned}&[\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{r,r+1} + [\mathbf{B}^\intercal\mathbf{F}\mathbf{B}]_{r+1,r} = \\ =&\mathbf{F}_{r,r+1} + b_{|r-1|_s}(\mathbf{F}_{|r-1|_s,r+1} + \mathbf{F}_{r+1,|r-1|_s}) + 2b_r\mathbf{F}_{r,r} + b_{|r-1|_s}b_r(\mathbf{F}_{|r-1|_s,r} + \mathbf{F}_{r,|r-1|_s}).\end{aligned} \tag{4.66}$$

In the last parenthesis in Equation 4.64, there is no possibility of the same indices, i.e. $|r-1|_s = |c-1|_s$, because $r < c$. In the case of $\mathrm{char}(\mathbb{F}) = 2$, we were able to skip the computation of $2b_s\mathbf{F}_{s,s}$ and $2b_r\mathbf{F}_{r,r}$.

The result can be summarized as:

$$[\triangledown(\mathbf{B}^\intercal\mathbf{F}\mathbf{B})]_{r,c} = \begin{cases} \textit{Equation } 4.63 & r = c, \\ \textit{Equation } 4.66 & r = c - 1, \\ \textit{Equation } 4.65 & r = 1 \wedge c = s, \\ 0 & r > c, \\ \textit{Equation } 4.64 & \textit{otherwise}. \end{cases} \tag{4.67}$$

Equation 4.67 for the condition $r \leq c$ can be rewritten as Algorithm 4.1.

4.3.3 Side-Channel Leakage Evaluation

We describe our key-vs.-key t-test methodology in subsubsection 4.3.3.1, and then, we present our results in subsubsection 4.3.3.2. We used an attack-independent test vector leakage assessment methodology (see section 2.7). We were not able to detect any statistically significant leakage from the protected implementation.

4.3.3.1 Methodology

We implemented Rainbow with the proposed equivalent key scheme (using the unprotected reference implementation) on a 32-bit STM32F303 microcontroller based on the

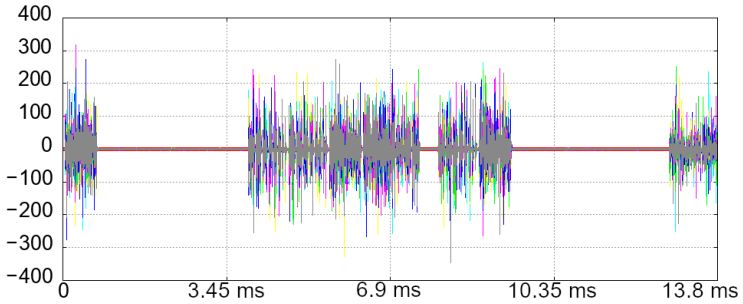
ARM Cortex-M4 core. To allow for a thorough and feasible side-channel evaluation of the proposed countermeasure, we chose Rainbow parameters $\mathbb{F}_q = GF(16), v_1 = o_1 = o_2 = 8$ to shorten the signing algorithm runtime compared to the Ia or Vc variants. We used the proposed efficient implementation of the equivalent keys scheme. The microcontroller was mounted in a ChipWhisperer C308 stand-alone evaluation board powered by an external 5 V laboratory power supply and clocked by a 7.37 MHz crystal.

The embedded Rainbow implementation receives a random seed from a controlling PC, which is then expanded using a linear congruential generator. The microcontroller then performs multiple signings without any external communication. Based on the generated pseudorandom numbers, the implementation generates the digests to be signed and also chooses one of four predefined private keys. This way, the microcontroller signs multiple randomly generated digests, each one using one of four randomly chosen private keys. The randomly interleaved private keys are a necessary prerequisite for our leakage evaluation methodology, as described later. When the signings are done, the microcontroller sends a checksum of the signatures back to the controlling PC, and the whole process is repeated as many times as necessary. This approach allows for a significant speedup of the measurement process.

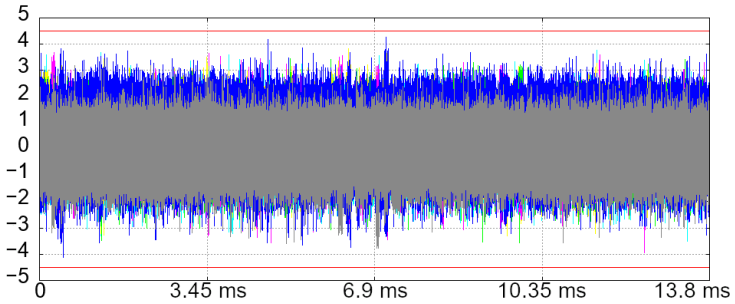
Voltage drops over the core were amplified using the Langer EMV-Technik PA303 preamplifier and sampled using the Picoscope 6404D oscilloscope. A 10Ω shunt resistor was used. The oscilloscope had a 25 MHz bandwidth limiter enabled, and the measurement channel was set in DC 50 Ω mode (since the preamplifier acts like a DC blocker). The sampling rate was 312 MS/s, in our case resulting in over 4 million samples per trace. The measured traces were then decimated 1:10 to 31.2 MS/s to allow for a feasible evaluation while maintaining a good signal-to-noise ratio [127].

The voltage was sampled during each signing, which lasted approximately 13.8 ms. Each decimated power trace consisted of 430,560 samples. The unprotected and protected implementations were evaluated independently. For the unprotected implementation, one of the four original private keys was randomly chosen in every signing. For the protected implementation, an equivalent key of one of the four original private keys was randomly chosen, and the next equivalent key was generated from the previous one.

The side-channel leakage was then evaluated using a fixed-fixed t-test methodology (see section 2.7). The measured power traces were partitioned into four groups based on the original private key used during the signing. Welch's t-statistic was then computed between every two groups (six evaluations in total), in every sampling point independently. The *null hypothesis* was that the two groups' means are equal, i.e., the original keys are indistinguishable by the mean power consumption. The hypothesis was rejected for high values of the $|t|$-statistic according to the Student's distribution and selected significance level. In side-channel leakage evaluation, the threshold of 4.5 or 5 is typically considered for the $|t|$-statistic, which must be further evaluated carefully with the possibility of both positive and negative false results in mind.



(a) Unprotected.



(b) Protected.

Figure 4.1: Results of the t-tests. Each graph contains six overlaid curves. The t-value is depicted on the vertical axis, and time is on the horizontal axis.

4.3.3.2 Results

Figure 4.1 depicts the results of the leakage evaluations. Figure 4.1a depicts the results of the evaluation on the unprotected implementation, where every t-test was performed using approximately 40,000 power traces. Figure 4.1b depicts the results for the protected implementation, where every t-test was performed using approximately one million power traces. Each graph contains six overlaid curves, one for each key-vs.-key t-test. As can be seen in Figure 4.1a, many peaks reach t-value of 200 and some even exceed 300, in contrast to Figure 4.1b, where the threshold of 4.5 (marked by red horizontal lines) is not surpassed by the protected implementation. Therefore, we did not detect any statistically significant side-channel leakage from our protected implementation during the signing.

4.3.4 Time Evaluation

We evaluated the time performance on three different platforms:

- The STM32F303 ARM microcontroller (a single-core Cortex-M4);
- A Raspberry Pi 3 B+ single-board computer equipped with the Broadcom BCM2837 ARM microprocessor (a four-core Cortex-A53);

Table 4.5: Time overhead comparison.

Implementation	Unprotected	Protected	Slowdown
STM32F303	13.8 ms	$13.8 + 33.28 =$ 47.08 ms[1]	3.41×
RPi	7.72 ms	$7.72 + 19.87 =$ 27.59 ms[1]	3.57×
PC	116 μs	$116 + 420 = 536$ μs[1]	4.62×
Countermeasure from [154]	162, 821 cycles	539, 338 cycles	3.31×

[1] Our equivalent key can be precomputed any time prior to the signing, and then, the signing itself has no overhead.

- A desktop computer equipped with an Intel Core i5-2400 processor (four cores).

On the STM32F303 microcontroller, we used the same parameters as for the leakage assessment, i.e., $v_1 = o_1 = o_2 = 8$, where the secret key size is 1776 bytes. On the Raspberry Pi and desktop computer, we used the parameters $v_1 = o_1 = o_2 = 32$, where the secret key size is 95,520 bytes. The Rainbow algorithm runtime was measured including random number generating using a linear congruent generator, excluding hashing. All the implementations were compiled with GCC without any optimizations enabled.

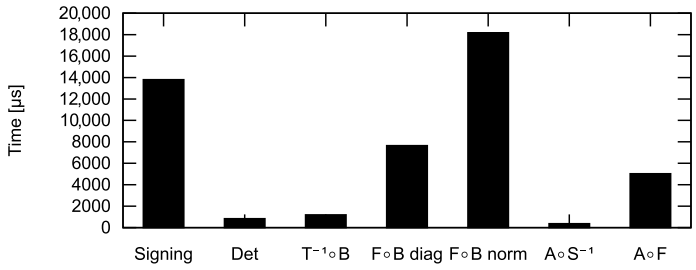
On the STM32F303 microcontroller, the time to sign a document was 13.8 ms, and the average time to generate the equivalent key was 33.28 ms. The whole signing including the equivalent key generation was 3.41-times slower compared to the unsecured signing.

On the Raspberry Pi 3 single-board computer, the average time to sign a document was 7.72 ms, and the average time to generate the equivalent key was 19.87 ms. The whole signing including the equivalent key generation was 3.57-times slower compared to the unsecured signing.

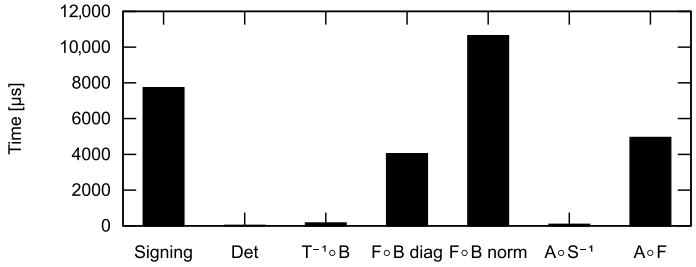
On the desktop computer, the average time to sign a document was 116 μs, and the average time to generate the equivalent key was 420 μs. The whole signing including the equivalent key generation was 4.62-times slower compared to the unsecured signing.

For comparison, the randomization countermeasure of Rainbow proposed in [154] resulted in 3.31-times slower signing. Table 4.5 summarizes the time overhead comparison. However, our equivalent key can be precomputed any time prior to the signing, and then, the signing itself has no overhead.

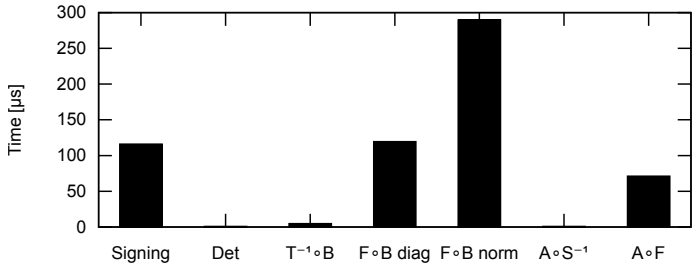
Figure 4.2 shows the calculation time of the individual generation components. `Det` is the time of the $(\mathbf{A}, \mathbf{B})$ tuple generation including the regularity verification. $\mathtt{T}^{-1} \circ \mathtt{B}$, $\mathtt{A} \circ \mathtt{S}^{-1}$ and $\mathtt{A} \circ \mathtt{F}$ correspond to the calculation of $\mathbf{T}^{-1}\mathbf{B}$, $\mathbf{A}\mathbf{S}^{-1}$, and $A \circ F$, respectively. The time of `F∘B diag` refers to Algorithm 4.1, and `F∘B norm` refers to Equation 4.60. The time of the signing itself is also included for comparison.



(a) STM32F303 ARM microcontroller, $v_1 = o_1 = o_2 = 8$.



(b) Raspberry Pi 3 single-board computer, $v_1 = o_1 = o_2 = 32$.



(c) Desktop computer, $v_1 = o_1 = o_2 = 32$.

Figure 4.2: Execution times of the components in equivalent key generation.

4.3.5 Memory Evaluation

From the memory perspective, the generating of the equivalent key is not an in-place algorithm due to the matrix multiplication in Equation 4.59. As the key structure is the same as for the unprotected version, we further describe only the extra variables needed for the generation of the equivalent key. The generating of the equivalent key does not need a dynamically allocated memory (as we assumed predefined parameters), and the biggest heap variable is the $(\mathbf{A}, \mathbf{B})$ two-tuple itself, which only takes $(n+m) \cdot \log_2(q)$ bits, where $\log_2(q)$ is the size of an element of $\mathbb{F}_q$ in bits. Other local variables are negligible. Additionally, one can split each part of the mask to be applied separately. This can further reduce extra memory needs to $\max(n, m) \cdot \log_2(q)$ bits.

The implementation submitted to NIST uses a special case of the secret key, where matrices $\mathbf{S}$ and $\mathbf{T}$ are specifically selected for memory effectiveness. After the generating of the equivalent key, this special format is no longer possible. As a consequence, we obtained a slightly larger secret key, where the size difference was $(v_1^2 + 2o_1^2 + 2o_2^2) \cdot \log_2(q)$ bits. For the Ia parameters, the difference was 2560 bytes, resulting in a key that was approximately 2.6% larger. Our countermeasure also requires a minor modification of the submitted implementation.

Fresh random bits are needed during the generation and even during the signing process. The amount of randomness is deterministic, except the case where singular matrices are generated and need to be generated again. For the equivalent key generation, the mode (most common value) of the required bits of randomness is $(n+m) \cdot \log_2(q)$, and for the signing process, it is $v_1 \cdot \log_2(q)$ (excluding the hashing salt). The mode is also a minimum number of required bits.

4.3.6 Summary

In this section, we proposed a side-channel countermeasure for multivariate quadratic signature schemes. We used the Rainbow signature scheme as our use case. However, the proposed countermeasure is applicable to other schemes such as unbalanced oil and vinegar as well (UOV is equivalent to a single-layer Rainbow).

We proposed an equivalent private key scheme, in which a precomputed randomly generated equivalent key is used for signing instead of the original fixed private key. We examined the scheme in general and described its restrictions and security properties from the theoretical point of view. These include the number of different equivalent keys or reached entropy. We showed that the number of equivalent keys is enormous, and given that, we believe the probability of using the same equivalent key twice is negligible in practice. This efficiently prevents the attacker from mounting attacks such as differential or correlation power analysis.

We further proposed an efficient equivalent key scheme. Our scheme requires significantly less fresh randomness (bounded by the $O(n)$) than the general equivalent key (bounded by the $O(n^2)$) and allows for faster and more efficient computation. We de-

scribed its properties, similar to the general case. We described an efficient algorithm for the generation of the equivalent key.

We evaluated our proposed countermeasure using attack-independent side-channel leakage assessment with one million power traces, and we were not able to detect any statistically significant information leakage. Lastly, we described the time and memory requirements of the countermeasure. The overhead of our countermeasure is comparable to other relevant countermeasures. Moreover, our equivalent key can be precomputed any time prior to the signing, and then, there is no overhead at the time of signing.

4.4 Summary

In this chapter, side-channel security of the post-quantum candidate for digital signature Rainbow was discussed. Even though the Rainbow algorithm was proven unsecure in 2022 [18], the presented work is, with some changes, applicable to other multivariate-quadratic algorithms, such as UOV, as well. An attack on the reference 32-bit software implementation of Rainbow was presented and evaluated in section 4.2 and is thoroughly summarized in subsection 4.2.5. A novel countermeasure for multivariate-quadratic signature schemes, based on a concept of equivalent keys and applicable in both software and hardware implementations, was presented, discussed, and evaluated in section 4.3 and is thoroughly summarized in subsection 4.3.6.

Chapter 5

Conclusion

Side-channel analysis poses a serious threat to the security of embedded devices. Since approximately 25 years ago, when its basics were laid, it has become a widely recognized and studied research field. Unlike classical cryptanalysis, side-channel attacks exploit implementation weaknesses of cryptography, both symmetric and asymmetric. Both software and hardware implementations are endangered and various countermeasures against such attacks must be implemented, with overall cost in mind.

In Chapter 2, the current state of the art in the field of passive non-invasive side-channel analysis was presented. Attacks based on various principles were described, including non-profiled and profiled attacks using classical statistics, as well as those based on machine learning. Various categories of countermeasures were also presented, followed by methods of overcoming these at the additional computational cost. Metrics related to side-channel security were also described. Finally, leakage assessment methodologies were presented.

Countermeasures tailored for FPGAs for protection against side-channel attacks were proposed in Chapter 3. A combination of masking and hiding countermeasures, based on dynamic logic reconfiguration, previously proposed for PRESENT, was modified for AES and Serpent. The result was thoroughly evaluated, including effects of specific countermeasures and their combinations. Furthermore, a novel approach for improving side-channel security of AES, PRESENT, and Serpent hardware implementations using high-level synthesis was proposed and evaluated.

Asymmetric cryptography is discussed in Chapter 4, namely the multivariate-quadratic signature scheme Rainbow. The signature scheme was a candidate for the post-quantum standard in the NIST competition until its third round. A side-channel attack on the reference 32-bit implementation of Rainbow was proposed and evaluated. Finally, a novel countermeasure for multivariate-quadratic signature schemes, based on the concept of equivalent private keys, was proposed and thoroughly discussed and evaluated. Unfortunately, the Rainbow scheme was found to be insecure in 2022 [18], and the focus of the cryptographic community has moved back to algorithms such as Unbalanced Oil and Vinegar (UOV). Luckily, since the Rainbow algorithm is a generalization of UOV, the work presented in Chapter 4 is applicable to UOV and its different variants as well.

5.1 Summary of Contributions

In section 3.2, we described and evaluated side-channel attack-protected AES and Serpent implementations, based on an approach previously proposed for PRESENT by Sasdrich et al. [148]. The countermeasures comprise both hiding and masking approaches and can be easily deployed in both FPGA and ASIC designs. We tailored the approach to a Xilinx Spartan-6 FPGA, and we described an approach to secure a generic permutation-substitution network cipher. We evaluated the latency and area utilization for different reconfiguration strategies. We demonstrated the effectiveness of the implemented countermeasures by evaluating side-channel leakage using one million power traces, with different combinations of countermeasures in place, demonstrating their practical contribution. We have not detected any exploitable side-channel leakage in our fully protected implementations.

In section 3.3, we proposed a first side-channel attack-resistant design of PRESENT, AES, and Serpent using high-level synthesis. We showed that the high-level synthesis is well capable of competing with the traditional RTL design when it comes to unprotected implementations, and we evaluated the effectiveness and efficiency of the protected implementations as well. We showed that the area/latency overhead brought in by high-level synthesis is reasonable considering its other advantages and compared to other similarly protected implementations. We performed a side-channel leakage assessment using one million power traces, showing that our proposed masking scheme successfully conceals the first-order leakage of PRESENT implementation. However, we were able to detect leakage of AES and Serpent implementations, which is consistent with the results presented in the previous section, where different combinations of countermeasures were examined. We discussed the consequences of the system-level approach and high-level synthesis and the related limitations.

In section 4.2, we extended a previously published attack to the 32-bit Rainbow signature scheme implementation, submitted to the NIST post-quantum standardization competition. We overcame several practical obstacles, and we described our attack. We evaluated the proposed attack on a 32-bit microcontroller with an ARM Cortex-M4 core and successfully extracted the secret key. The work aimed to contribute to the standardization process.

In section 4.3, we proposed a countermeasure against side-channel attacks applicable to multivariate-quadratic signature schemes such as Rainbow. We described a concept of equivalent private keys and proposed an approach to randomize these in order to avoid exploitable side-channel leakage. We provided a formal analysis of the proposed equivalent keys, showing a number of different keys and their entropy. Then we proposed a more efficient implementation of the countermeasure with respect to calculation performance, the amount of fresh randomness necessary, the entropy of the generated equivalent key, implementation size and simplicity, and the total number of keys equivalent to each public key. We evaluated the proposed countermeasures regarding both time/memory requirements and side-channel leakage. We were not able to detect any statistically significant side-channel leakage from our protected implementation using one million measurements.

All the results were presented and discussed within the scientific community. The works contained in this thesis were published in three conference proceedings and four journal articles. Also, the presented works have already received a considerable amount of citations.

5.2 Future Work

The author of the dissertation thesis suggests following future work:

- We suggest implementation of more complex glitch-resistant side-channel countermeasures for FPGA using high-level synthesis. Domain oriented masking for high-level synthesis was already proposed in [145], where our original research is cited.
- More research on system-level countermeasures for hardware implementations, using high-level synthesis, would be beneficial. Specifically, countermeasures for AI accelerators, such as those generated by HLS4ML [53], are necessary. These are typically employed in cloud FPGA platforms, where IP theft using side-channel attacks poses a serious threat.
- The side-channel attack on the Rainbow signature scheme could be extended to other Rainbow variants, such as cyclic Rainbow. However, since the Rainbow signature scheme was found to be insecure, this is not of great interest anymore. Therefore, the attack could instead be evaluated on the original UOV scheme.
- Our countermeasure for multivariate quadratic signatures, implemented and evaluated on the Rainbow, could be tailored for other signature schemes, such as UOV and its variants.

Bibliography

[1] Mehmet Şahin Açikkapi, Fatih Özkaynak, and Ahmet Bedri Özer. Side-channel analysis of chaos-based substitution box structures. *IEEE Access*, 7:79030–79043, 2019.

[2] Ali Afzali-Kusha, Makoto Nagata, Nishath K Verghese, and David J Allstot. Substrate noise coupling in soc design: Modeling, avoidance, and validation. *Proceedings of the IEEE*, 94(12):2109–2138, 2006.

[3] Juan Ai, Zhu Wang, Xinping Zhou, and Changhai Ou. Improved wavelet transform for noise reduction in power analysis attacks. In *2016 IEEE International Conference on Signal and Image Processing (ICSIP)*, pages 602–606. IEEE, 2016.

[4] Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, and Didier Moyart. Power analysis, what is now possible... In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 489–502. Springer, 2000.

[5] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of des and aes, secure against some attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 309–318. Springer, 2001.

[6] Amir Alipour, Athanasios Papadimitriou, Vincent Beroulle, Ehsan Aerabi, and David Hély. On the performance of non-profiled differential deep learning attacks against an aes encryption algorithm protected using a correlated noise generation based hiding countermeasure. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 614–617. IEEE, 2020.

[7] Mythri Alle, Keshavan Varadarajan, Alexander Fell, Nimmy Joseph, Saptarsi Das, Prasenjit Biswas, Jugantor Chetia, Adarsh Rao, SK Nandy, and Ranjani Narayan. Redefine: Runtime reconfigurable polymorphic asic. *ACM Transactions on Embedded Computing Systems (TECS)*, 9(2):1–48, 2009.

[8] George E. Andrews and Bruce Berndt. *Ramanujan's Lost Notebook*. Springer New York, 2005. doi:10.1007/0-387-28124-x.

[9] Jean-Philippe Aumasson and Daniel J Bernstein. Siphash: a fast short-input prf. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*, pages 489–508. Springer, 2012.

[10] Moshe Avital, Hadar Dagan, Osnat Keren, and Alexander Fish. Randomized multi-topology logic against differential power analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(4):702–711, 2014.

[11] Karthik Baddam and Mark Zwolinski. Divided backend duplication methodology for balanced dual rail routing. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 396–410. Springer, 2008.

[12] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–485. Springer, 2015.

[13] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In *International Conference on Smart Card Research and Advanced Applications*, pages 263–276. Springer, 2012.

[14] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Comparative evaluation of rank correlation based dpa on an aes prototype chip. In *International Conference on Information Security*, pages 341–354. Springer, 2008.

[15] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, 2011.

[16] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck lightweight block ciphers. In *Proceedings of the 52nd annual design automation conference*, pages 1–6, 2015.

[17] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.

[18] Ward Beullens. Breaking rainbow takes a weekend on a laptop. Cryptology ePrint Archive, Report 2022/214, 2022. https://ia.cr/2022/214.

[19] Ward Beullens and Bart Preneel. Field lifting for smaller uov public keys. In *International Conference on Cryptology in India*, pages 227–246. Springer, 2017.

[20] Régis Bevan and Erik Knudsen. Ways to enhance differential power analysis. In *International Conference on Information Security and Cryptology*, pages 327–342. Springer, 2002.

[21] Eli Biham, Ross Anderson, and Lars Knudsen. Serpent: A new block cipher proposal. In *International Workshop on Fast Software Encryption*, pages 222–238. Springer, 1998.

[22] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 326–343. Springer, 2014.

[23] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient aes threshold implementation. In *International Conference on Cryptology in Africa*, pages 267–284. Springer, 2014.

[24] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-offs for threshold implementations illustrated on aes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1188–1200, 2015.

[25] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *International workshop on cryptographic hardware and embedded systems*, pages 450–466. Springer, 2007.

[26] Fraidy Bouesse, Marc Renaudin, and Gilles Sicard. Improving dpa resistance of quasi delay insensitive circuits using randomly time-shifted acknowledgment signals. In *Vlsi-Soc: From Systems To Silicon*, pages 11–24. Springer, 2007.

[27] Fraidy Bouesse, Gilles Sicard, and Marc Renaudin. Path swapping method to improve dpa resistance of quasi delay insensitive asynchronous circuits. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 384–398. Springer, 2006.

[28] G Fraidy Bouesse, Marc Renaudin, Sophie Dumont, and Fabien Germain. Dpa on quasi delay insensitive asynchronous circuits: Formalization and improvement. In *Design, Automation and Test in Europe*, pages 424–429. IEEE, 2005.

[29] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[30] Frank Markham Brown. *Boolean reasoning: the logic of Boolean equations*. Springer Science & Business Media, 2012.

[31] Nicolas Bruneau, Charles Christen, Jean-Luc Danger, Adrien Facon, and Sylvain Guilley. Security evaluation against side-channel analysis at compilation time. In *International Conference on Algebra, Codes and Cryptology*, pages 129–148. Springer, 2019.

[32] Marco Bucci, Luca Giancane, Raimondo Luzzi, and Alessandro Trifiletti. Three-phase dual-rail pre-charge logic. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 232–241. Springer, 2006.

[33] Marco Bucci, Raimondo Luzzi, Michele Guglielmo, and Alessandro Trifiletti. A countermeasure against differential power analysis based on random delay insertion. In *2005 IEEE International Symposium on Circuits and Systems*, pages 3547–3550. IEEE, 2005.

[34] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 45–68. Springer, 2017.

[35] Giovanni Camurati, Aurélien Francillon, and François-Xavier Standaert. Understanding screaming channels: From a detailed analysis to improved attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 358–401, 2020.

[36] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 163–177, 2018.

[37] Cécile Canovas and Jessy Clédière. What do s-boxes say in differential side channel attacks? *IACR Cryptol. ePrint Arch.*, 2005:311, 2005.

[38] David Canright and Lejla Batina. A very compact "perfectly masked" s-box for aes. In *International Conference on Applied Cryptography and Network Security*, pages 446–459. Springer, 2008.

[39] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference*, pages 398–412. Springer, 1999.

[40] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[41] Byong-Deok Choi, Kyung Eun Kim, Ki-Seok Chung, and Dong Kyue Kim. Symmetric adiabatic logic circuits against differential power analysis. *ETRI journal*, 32(1):166–168, 2010.

[42] Omar Choudary and Markus G Kuhn. Efficient template attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 253–270. Springer, 2013.

[43] Selina Chu, Eamonn Keogh, David Hart, and Michael Pazzani. Iterative deepening dynamic time warping for time series. In *Proceedings of the 2002 SIAM International Conference on Data Mining*, pages 195–212. SIAM, 2002.

[44] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 252–263. Springer, 2000.

[45] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In *International Conference on Smart Card Research and Advanced Applications*, pages 277–284. Springer, 1998.

[46] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[47] Saptarsi Das and SK Nandy. A flexible crypto-system based upon the redefine polymorphic asic architecture. *Defence Science Journal*, 62(1):25–31, 2012.

[48] Nicolas Debande, Youssef Souissi, M Abdelaziz El Aabid, Sylvain Guilley, and Jean-Luc Danger. Wavelet transform based pre-processing for side channel analysis. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*, pages 32–38. IEEE, 2012.

[49] Bert den Boer, Kerstin Lemke, and Guntram Wicke. A dpa attack against the modular reduction within a crt implementation of rsa. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 228–243. Springer, 2002.

[50] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International Conference on Applied Cryptography and Network Security*, pages 164–175. Springer, 2005.

[51] Javier Duarte, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, Jennifer Ngadiuba, Maurizio Pierini, Ryan Rivera, Nhan Tran, et al. Fast inference of deep neural networks in fpgas for particle physics. *Journal of Instrumentation*, 13(07):P07027, 2018.

[52] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: Algorithm specifications and supporting documentation. *Submission to the NIST's post-quantum cryptography standardization process*, 35, 2019.

[53] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579*, 2021.

[54] Yunsi Fei, Qiasi Luo, and A Adam Ding. A statistical model for dpa with novel algorithmic confusion analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 233–250. Springer, 2012.

[55] 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(5), 2018.

[56] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In *International Workshop on Selected Areas in Cryptography*, pages 262–280. Springer, 2010.

[57] Philippe Gaubert and Akinobu Teramoto. Carrier mobility in field-effect transistors. *Different Types of Field-Effect Transistors: Theory and Applications*, pages 2–25, 2017.

[58] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 426–442. Springer, 2008.

[59] Richard Gilmore, Neil Hanley, and Maire O'Neill. Neural network based attack on a masked implementation of aes. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111. IEEE, 2015.

[60] Dennis RE Gnad, Jonas Krautter, Mehdi B Tahoori, Falk Schellenberg, and Amir Moradi. Remote electrical-level security threats to multi-tenant fpgas. *IEEE Design & Test*, 37(2):111–119, 2020.

[61] Dennis RE Gnad, Fabian Oboril, Saman Kiamehr, and Mehdi B Tahoori. Analysis of transient voltage fluctuations in fpgas. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 12–19. IEEE, 2016.

[62] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[63] 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*, pages 3–3, 2016.

[64] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 33–48. Springer, 2011.

[65] Hendra Guntur, Jun Ishii, and Akashi Satoh. Side-channel attack user reference architecture board sakura-g. In *Consumer Electronics (GCCE), 2014 IEEE 3rd Global Conference on*, pages 271–274. IEEE, 2014.

[66] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *Journal of Cryptographic Engineering*, 10(2):135–162, 2020.

[67] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. A theoretical study of kolmogorov-smirnov distinguishers. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 9–28. Springer, 2014.

[68] Annelie Heuser and Michael Zohner. Intelligent machine homicide. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 249–264. Springer, 2012.

[69] Ekawat Homsirikamol and Kris Gaj. Can high-level synthesis compete against a hand-written code in the cryptographic domain? a case study. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, pages 1–8. IEEE, 2014.

[70] Yohei Hori, Toshihiro Katashita, Akihiko Sasaki, and Akashi Satoh. Sasebo-giii: A hardware security evaluation board equipped with a 28-nm fpga. In *The 1st IEEE Global Conference on Consumer Electronics 2012*, pages 657–660. IEEE, 2012.

[71] Paul Horowitz, Winfield Hill, and Ian Robinson. *The art of electronics*, volume 2. Cambridge university press Cambridge, 1989.

[72] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293, 2011.

[73] Gerardus TM Hubert. Current source for cryptographic processor, August 4 2009. US Patent 7,571,492.

[74] Stanislav Jeřábek, Jan Schmidt, Martin Novotný, and Vojtěch Miškovský. Dummy rounds as a dpa countermeasure in hardware. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 523–528. IEEE, 2018.

[75] Zhenghong Jiang, Steve Dai, G Edward Suh, and Zhiru Zhang. High-level synthesis with timing-sensitive information flow enforcement. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[76] Zhenghong Jiang, Hanchen Jin, G Edward Suh, and Zhiru Zhang. Designing secure cryptographic accelerators with information flow enforcement: A case study on aes. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[77] Norman L Johnson, Adrienne W Kemp, and Samuel Kotz. *Univariate discrete distributions*, volume 444. John Wiley & Sons, 2005.

[78] Mabin Joseph, Gautham Sekar, and R Balasubramanian. Side channel analysis of speck. *Journal of Computer Security*, 28(6):655–676, 2020.

[79] Najeh Kamoun, Lilian Bossuet, and Adel Ghazel. Correlated power noise generator as a low cost dpa countermeasures to secure hardware aes cipher. In *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*, pages 1–6. IEEE, 2009.

[80] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[81] Ayesha Khalid, Goutam Paul, and Anupam Chattopadhyay. High level synthesis for symmetric key cryptography. In *Domain Specific High-Level Synthesis for Cryptographic Workloads*, pages 51–90. Springer, 2019.

[82] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 206–222. Springer, 1999.

[83] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[84] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[85] ST Choden Konigsmark, Deming Chen, and Martin DF Wong. High-level synthesis for side-channel defense. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 37–44. IEEE, 2017.

[86] Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 286–296, 2007.

[87] Sotiris B Kotsiantis, I Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[88] Jonas Krautter, Dennis RE Gnad, Falk Schellenberg, Amir Moradi, and Mehdi B Tahoori. Active fences against voltage-based side channels in multi-tenant fpgas. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[89] Takaya Kubota, Kota Yoshida, Mitsuru Shiozaki, and Takeshi Fujino. Deep learning side-channel attack against hardware implementations of aes. *Microprocessors and Microsystems*, page 103383, 2020.

[90] Thanh-Ha Le, Jessy Clédière, Cécile Canovas, Bruno Robisson, Christine Servière, and Jean-Louis Lacoume. A proposition for correlation power analysis enhancement. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 174–186. Springer, 2006.

[91] Thanh-Ha Le, Jessy Clédière, Christine Servière, and Jean-Louis Lacoume. Noise reduction in side channel attack using fourth-order cumulant. *IEEE Transactions on Information Forensics and Security*, 2(4):710–720, 2007.

[92] Kerstin Lemke-Rust and Christof Paar. Gaussian mixture models for higher-order side channel analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 14–27. Springer, 2007.

[93] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: an approach based on machine learning. *International Journal of Applied Cryptography*, 3(2):97–115, 2014.

[94] Liran Lerman, Romain Poussier, Olivier Markowitch, and François-Xavier Standaert. Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: extended version. *Journal of Cryptographic Engineering*, 8(4):301–313, 2018.

[95] Oleksiy Lisovets, David Knichel, Thorben Moos, and Amir Moradi. Let's take it offline: Boosting brute-force attacks on iphone's user authentication through sca. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 496–519, 2021.

[96] Hongying Liu, Guoyu Qian, Satoshi Goto, and Yukiyasu Tsunoo. Aes key recovery based on switching distance model. In *2010 Third International Symposium on Electronic Commerce and Security*, pages 218–222. IEEE, 2010.

[97] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[98] Houssem Maghrebi, Olivier Rioul, Sylvain Guilley, and Jean-Luc Danger. Comparison between side-channel analysis distinguishers. In *International Conference on Information and Communications Security*, pages 331–340. Springer, 2012.

[99] Badruddoja Majumder, Sakib Hasan, Mesbah Uddin, and Garrett S Rose. Chaos computing for mitigating side channel attack. In *2018 IEEE international symposium on hardware oriented security and trust (HOST)*, pages 143–146. IEEE, 2018.

[100] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[101] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked aes hardware implementations. In *International workshop on cryptographic hardware and embedded systems*, pages 157–171. Springer, 2005.

[102] Zdenek Martinasek, Lukas Malina, and Krisztina Trasy. Profiling power analysis attack based on multi-layer perceptron network. In *Computational Problems in Science and Engineering*, pages 317–339. Springer, 2015.

[103] Zdenek Martinasek and Vaclav Zeman. Innovative method of the power analysis. *Radioengineering*, 22(2):586–594, 2013.

[104] James L Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, page 204. IEEE, 1994.

[105] Robert Matthews. On the derivation of a "chaotic" encryption algorithm. *Cryptologia*, 13(1):29–42, 1989.

[106] Matthew Mayhew and Radu Muresan. On-chip nanoscale capacitor decoupling architectures for hardware security. *IEEE Transactions on Emerging Topics in Computing*, 2(1):4–15, 2014.

[107] Matthew Mayhew and Radu Muresan. An overview of hardware-level statistical power analysis attack countermeasures. *Journal of Cryptographic Engineering*, 7(3):213–244, 2017.

[108] Nele Mentens, Benedikt Gierlichs, and Ingrid Verbauwhede. Power and fault analysis resistance in hardware through dynamic reconfiguration. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 346–362. Springer, 2008.

[109] Thomas S Messerges. Securing the aes finalists against power analysis attacks. In *International Workshop on Fast Software Encryption*, pages 150–164. Springer, 2000.

[110] Thomas S Messerges. Using second-order power analysis to attack dpa resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 238–251. Springer, 2000.

[111] Thomas S Messerges, Ezzat A Dabbish, and Robert H Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE transactions on computers*, 51(5):541–552, 2002.

[112] Yong Moon and Deog-Kyoon Jeong. An efficient charge recovery logic circuit. *IEICE transactions on electronics*, 79(7):925–933, 1996.

[113] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 256–292, 2019.

[114] Thorben Moos, Felix Wegener, and Amir Moradi. Dl-la: Deep learning leakage assessment. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 552–598, 2021.

[115] Amir Moradi. Advances in side-channel security. *Habilitation, Ruhr-Universität Bochum*, 2015.

[116] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of aes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 69–88. Springer, 2011.

[117] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 209–237, 2018.

[118] Radu Muresan and Stefano Gregori. Protection circuit against differential power analysis attacks for smart cards. *IEEE Transactions on Computers*, 57(11):1540–1549, 2008.

[119] Miguel Angel Murillo-Escobar, César Cruz-Hernández, Fausto Abundiz-Pérez, and Rosa Martha López-Gutiérrez. Implementation of an improved chaotic encryption algorithm for real-time embedded systems by using a 32-bit microcontroller. *Microprocessors and Microsystems*, 45:297–309, 2016.

[120] National Institute of Standards and Technology. Advanced encryption standard. In *Federal Information Processing Standards Publication 197*, 2001.

[121] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *International conference on information and communications security*, pages 529–545. Springer, 2006.

[122] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *Journal of Cryptology*, 24(2):292–321, 2011.

[123] Daniel H Noronha, Bahar Salehpour, and Steven JE Wilton. Leflow: Enabling flexible fpga high-level synthesis of tensorflow deep neural networks. In *FSP Workshop 2018; Fifth International Workshop on FPGAs for Software Programmers*, pages 1–8. VDE, 2018.

[124] Matúš Olekšák and Vojtěch Miškovský. Correlation power analysis of siphash. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 84–87. IEEE, 2022.

[125] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order dpa attacks for masked smart card implementations of block ciphers. In *Cryptographers' Track at the RSA Conference*, pages 192–207. Springer, 2006.

[126] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the aes s-box. In *International workshop on fast software encryption*, pages 413–423. Springer, 2005.

[127] Colin O'Flynn and Zhizhang Chen. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *Journal of Cryptographic Engineering*, 5(1):53–69, 2015.

[128] Sanjay Pant. *Design and Analysis of Power Distribution Networks in VLSI Circuits*. PhD thesis, The University of Michigan, 2008.

[129] Aesun Park, Kyung-Ah Shim, Namhun Koo, and Dong-Guk Han. Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 500–523, 2018.

[130] Bruce B Pedersen. Programmable logic device with improved security, August 28 2012. US Patent 8,255,702.

[131] Albrecht Petzoldt, Stanislav Bulygin, and Johannes Buchmann. Cyclicrainbow–a multivariate signature scheme with a partially cyclic public key. In *International Conference on Cryptology in India*, pages 33–48. Springer, 2010.

[132] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):1–29, 2019.

[133] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2,300 ge. *Journal of Cryptology*, 24(2):322–345, 2011.

[134] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 142–159. Springer, 2013.

[135] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Transactions on computers*, 58(6):799–811, 2009.

[136] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, pages 200–210. Springer, 2001.

[137] Jan M Rabaey, Anantha P Chandrakasan, and Borivoje Nikolić. *Digital integrated circuits: a design perspective*, volume 7. Pearson education Upper Saddle River, NJ, 2003.

[138] Jan M.. Rabaey, Anantha P Chandrakasan, and Borivoje Nikolić. *Digital integrated circuits: a design perspective*. Pearson Education, Incorporated., 2003.

[139] Pqcrainbow.org. Online. URL: `https://www.pqcrainbow.org/`.

[140] Chethan Ramesh, Shivukumar B Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. Fpga side channel attacks without physical access. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.

[141] Girish B Ratanpal, Ronald D Williams, and Travis N Blalock. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*, 1(3):179–189, 2004.

[142] Alin Razafindraibe, Michel Robert, and Philippe Maurine. Improvement of dual rail logic as a countermeasure against dpa. In *2007 IFIP International Conference on Very Large Scale Integration*, pages 270–275. IEEE, 2007.

[143] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In *International Workshop on Information Security Applications*, pages 440–456. Springer, 2004.

[144] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[145] Rajat Sadhukhan, Sayandeep Saha, and Debdeep Mukhopadhyay. Shortest path to secured hardware: Domain oriented masking with high-level-synthesis. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 223–228. IEEE, 2021.

[146] Pradeep Kumar Sana and M Satyam. An energy efficient secure logic to provide resistance against differential power analysis attacks. In *2010 International Symposium on Electronic System Design*, pages 61–65. IEEE, 2010.

[147] Pascal Sasdrich, René Bock, and Amir Moradi. Threshold implementation in software. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 227–244. Springer, 2018.

[148] Pascal Sasdrich, Amir Moradi, Oliver Mischke, and Tim Güneysu. Achieving side-channel protection with dynamic logic reconfiguration on modern fpgas. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 130–136. IEEE, 2015.

[149] Laurent Sauvage, Sylvain Guilley, Jean-Luc Danger, Yves Mathieu, and Maxime Nassar. Successful attack on an fpga-based wddl des cryptoprocessor without place and route constraints. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 640–645. IEEE, 2009.

[150] Falk Schellenberg, Dennis RE Gnad, Amir Moradi, and Mehdi B Tahoori. An inside job: Remote power analysis attacks on fpgas. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1111–1116. IEEE, 2018.

[151] Tobias Schneider and Amir Moradi. Leakage assessment methodology. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 495–513. Springer, 2015.

[152] Tobias Schneider, Amir Moradi, and Tim Güneysu. Robust and one-pass parallel computation of correlation-based attacks at arbitrary order. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 199–217. Springer, 2016.

[153] Adi Shamir. Protecting smart cards from power analysis with detachable power supplies, January 14 2003. US Patent 6,507,913.

[154] Kyung-Ah Shim, Cheol-Min Park, and Yoo-Jin Baek. Lite-rainbow: Lightweight signature schemes based on multivariate quadratic equations and their secure implementations. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology – INDOCRYPT 2015*, pages 45–63, Cham, 2015. Springer International Publishing.

[155] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[156] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer networks*, 76:146–164, 2015.

[157] Bernard W Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26. CRC Press, 1986.

[158] Petr Socha. hls-crypto. URL: `https://github.com/petrsocha/hls-crypto`.

[159] Youssef Souissi, M Abdelaziz Elaabid, Nicolas Debande, Sylvain Guilley, and Jean-Luc Danger. Novel applications of wavelet transforms based side-channel analysis. In *Non-Invasive Attack Testing Workshop*, 2011.

[160] François-Xavier Standaert. Introduction to side-channel attacks. In *Secure integrated circuits and systems*, pages 27–42. Springer, 2010.

[161] François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. In *International Conference on Smart Card Research and Advanced Applications*, pages 65–79. Springer, 2018.

[162] François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected cmos devices. In *International Conference on Information Security and Cryptology*, pages 253–267. Springer, 2008.

[163] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A formal practice-oriented model for the analysis of side-channel attacks. *IACR e-print archive*, 134(2006):2, 2006.

[164] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 443–461. Springer, 2009.

[165] D.A. Suprunenko, K.A. Hirsch, and American Mathematical Society. *Matrix Groups*. Translations of mathematical monographs. American Mathematical Society, 1976. URL: `https://books.google.cz/books?id=5wnvAAAAMAAJ`.

[166] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 107–131, 2019.

[167] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Proceedings of the 28th European solid-state circuits conference*, pages 403–406. IEEE, 2002.

[168] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 246–251. IEEE, 2004.

[169] Carlos Tokunaga and David Blaauw. Securing encryption systems with a switched capacitor current equalizer. *IEEE Journal of Solid-State Circuits*, 45(1):23–31, 2009.

[170] Elena Trichina, Tymur Korkishko, and Kyung Hee Lee. Small size, low power, side channel-immune aes coprocessor: design and synthesis results. In *International Conference on Advanced Encryption Standard*, pages 113–127. Springer, 2004.

[171] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 175–199. Springer, 2020.

[172] Jasper GJ van Woudenberg, Marc F Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In *Cryptographers' Track at the RSA Conference*, pages 104–119. Springer, 2011.

[173] Nicolas Veyrat-Charvillon and François-Xavier Standaert. Mutual information analysis: how, when and why? In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 429–443. Springer, 2009.

[174] Jason Waddle and David Wagner. Towards efficient second-order power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–15. Springer, 2004.

[175] Carolyn Whitnall and Elisabeth Oswald. A comprehensive evaluation of mutual information analysis using a fair evaluation framework. In *Annual Cryptology Conference*, pages 316–334. Springer, 2011.

[176] Carolyn Whitnall and Elisabeth Oswald. A fair evaluation framework for comparing side-channel distinguishers. *Journal of Cryptographic Engineering*, 1(2):145–160, 2011.

[177] Carolyn Whitnall, Elisabeth Oswald, and Luke Mather. An exploration of the kolmogorov-smirnov test as a competitor to mutual information analysis. In *International Conference on Smart Card Research and Advanced Applications*, pages 234–251. Springer, 2011.

[178] Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The myth of generic dpa... and the magic of learning. In *Cryptographers' Track at the RSA Conference*, pages 183–205. Springer, 2014.

[179] Christopher Wolf and Bart Preneel. Equivalent keys in multivariate quadratic public key systems. *Journal of Mathematical Cryptology*, 4(4):375–415, 2011. URL: `https://doi.org/10.1515/jmc.2011.004`, `doi:doi:10.1515/jmc.2011.004`.

[180] Xilinx. Spartan-6 libraries guide for hdl designs. URL: https://www.xilinx.com/support/documentation/sw_\protect\penalty\z@manuals/xilinx14_\protect\penalty\z@5/spartan6_\protect\penalty\z@hdl.pdf.

[181] Xilinx. Vivado design suite user guide: high-level synthesis (ug902), 2018.

[182] Lu Zhang, Wei Hu, Armaiti Ardeshiricham, Yu Tai, Jeremy Blackstone, Dejun Mu, and Ryan Kastner. Examining the consequences of high-level synthesis optimizations on power side-channel. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1167–1170. IEEE, 2018.

[183] Mark Zhao and G Edward Suh. Fpga-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 229–244. IEEE, 2018.

[184] Nianhao Zhu, Yujie Zhou, and Hongming Liu. Counteracting leakage power analysis attack using random ring oscillators. In *Proceedings of 2013 International Conference on Sensor Network Security Technology and Privacy Communication System*, pages 74–77. IEEE, 2013.

Reviewed Publications of the Author Presented in This Thesis

[A.1] Petr Socha, Vojtěch Miškovský and Martin Novotný. A Comprehensive Survey on the Non-Invasive Passive Side-Channel Analysis. *Sensors*, 22(21):3607, 2022.

[A.2] Petr Socha, Jan Brejník, Stanislav Jeřábek, Martin Novotný and Nele Mentens. Dynamic Logic Reconfiguration Based Side-Channel Protection of AES and Serpent. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 277-282. IEEE, 2019.

The paper has been cited 2 times, in:

- Harcha, G.; Introduction d'aléas dans les architectures matérielles pour une contribution à la sécurisation de chiffreurs AES dans un contexte IoT; Doctoral thesis, 2021.
- Wittke, Ch.; Investigation of the effects of layout variants on side channels of accelerators for cryptographic operations; Doctoral thesis, 2021.

[A.3] Petr Socha, Jan Brejník, Josep Balasch, Martin Novotný and Nele Mentens. Side-channel countermeasures utilizing dynamic logic reconfiguration: Protecting AES/Rijndael and Serpent encryption in hardware. *Microprocessors and Microsystems*, 78:103208, 2020.

The paper has been cited 2 times, in:

- Sepúlveda, J.; Secure Cryptography Integration: NoC-Based Microarchitectural Attacks and Countermeasures; Network-on-Chip Security and Privacy, pp. 153 - 179, 2021. ISBN 978-3-030-69131-8.
- Das, N.; Panchanathan, A.; SD-SHO: Security-dominated finite state machine state assignment technique with a satisfactory level of hardware optimization; IET Computers and Digital Techniques, vol. 15, no. 5, pp. 372 - 392, 2021. ISSN 1751-8601.

[A.4] Petr Socha and Martin Novotný. Towards High-Level Synthesis of Polymorphic Side-Channel Countermeasures. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 193-199. IEEE, 2020.

The paper has been cited 2 times, in:

- Sadhukhan, R.; Saha, S.; Mukhopadhyay, D.; Shortest Path to Secured Hardware: Domain Oriented Masking with High-Level-Synthesis; 2021 58TH ACM/IEEE Design Automation Conference (DAC), pp. 223 - 228, 2021. ISSN 0738-100X.

- Bran, C.; Flores, D.; Hernández, C.; Cryptography model to secure IoT device endpoints, based on polymorphic cipher OTP; 2022 IEEE 40th Central America and Panama Convention (CONCAPAN), 2022. ISBN 978-1-7281-6715-2.

[A.5] Petr Socha, Vojtěch Miškovský and Martin Novotný. High-level synthesis, cryptography, and side-channel countermeasures: A comprehensive evaluation. *Microprocessors and Microsystems*, 104311, 2021.

The paper has been cited 2 times, in:

- Koufopoulou, A. A.; Development of hardware countermeasures for embedded systems security using High-Level Synthesis; Master's thesis, 2022.
- Koufopoulou, A.; Xevgeni, K.; Papadimitriou, A.; Psarakis, M.; Hely, D.; Security and Reliability Evaluation of Countermeasures implemented using High-Level Synthesis; 2022 IEEE 28th International Symposium on On-line Testing and Robust System Design (IOLTS 2022), 2022. ISSN 1942-9398.

[A.6] David Pokorný, Petr Socha and Martin Novotný. Side-channel attack on Rainbow post-quantum signature. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1-4. IEEE, 2021.

The paper has been cited 5 times, in:

- Aulbach, T.; Kovats, T.; Krämer, J.; Marzougui, S.; Recovering Rainbow's Secret Key with a First-Order Fault Attack; Progress in Cryptology - AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, 2022.
- Jin, X.; Katsis, C.; Sang, F.; Sun, J.; Kundu, A.; Kompella, R.; Edge Security: Challenges and Issues; arXiv:2206.07164, 2022.
- Gupta, N.; Jati, A.; Chattopadhyay, A.; Jha, G.; Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium; Cryptology ePrint Archive, 2022.
- Zhi, Q.; Jiang, X.; Zhang, H.; Zhou, Z.; Ren, J.; Huang, T.; An Anti-Physical Attack Scheme of ARX Lightweight Algorithms for IoT Applications; Computer Systems Science and Engineering, 2023. ISSN 0267-6192.
- Aulbach, T.; Campos, F.; Krämer, J.; Samardjiska, S.; Stöttinger, M.; Separating Oil and Vinegar with a Single Trace; Cryptology ePrint Archive, 2023.

[A.7] David Pokorný, Petr Socha and Martin Novotný. Equivalent Keys: Side-Channel Countermeasure for Post-Quantum Multivariate Quadratic Signatures. *Electronics*, 11(21):3607, 2022.

Remaining Reviewed Publications of the Author

[A.8] Petr Socha, Vojtěch Miškovský, Hana Kubátová and Martin Novotný. Optimization of Pearson correlation coefficient calculation for DPA and comparison of different approaches. In *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 184-189. IEEE, 2017.

The paper has been cited 14 times, in:

- Wei, Y.; Zhang, Y.; Tong, W.; Form factors of modeling design language with improved entropy weight based on Kaisen engineering; Revista de la Facultad de Ingenieria, vol. 32, pp. 357 - 363, 2017. ISSN 0798-4065.
- Liu, Y. J.; Zhao, Y. Q.; He, J. J.; Liu, A. Q.; Xin, R. S.; SCCA: Side-Channel Correlation Analysis for Detecting Hardware Trojan; Proceedings of 2017 11th IEEE International Conference on Anti-Counterfeiting, Security, and Identification (ASID), pp. 196 - 200, 2017. ISBN 978-1-5386-0533-2. ISSN 2163-5048.
- Nascimento, J. F. O.; Estudo de preços de energia no mercado spot e futuros no mibel; Master's thesis, 2017.
- Shakhari, S.; Verma, A. K.; Ghosh, D.; Bhar, K. K.; Banerjee, I.; Diverse Water Quality Data Pattern Study of the Indian River Ganga: Correlation and Cluster Analysis; 2019 17th International Conference on ICT and Knowledge Engineering (ICT&KE), pp. 93 - 99, 2019. ISBN 978-1-7281-3209-9. ISSN 2157-0981.
- Bos, S.; Secure Machine Learning for Privacy Preserving Genetic Disease Identification; Master's thesis, 2019.
- Randolph, M.; Diehl, W.; Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman; Cryptography, vol. 4, no. 2, 2020.
- Garcia, L. G.; Molina, R. S.; Crespo, M. L.; Carrato, S.; Ramponi, G.; Cicuttin, A.; Morales, I. R.; Perez, H.; Muon-Electron Pulse Shape Discrimination for Water Cherenkov Detectors Based on FPGA/SoC; Electronics, vol. 10, no. 3, 2021.
- Kaushal, A.; Alexander, R.; Rao, P.T.; Prakash, J.; Dasgupta, K.; Artificial neural network, Pareto optimization, and Taguchi analysis for the synthesis of single-walled carbon nanotubes; Carbon Trends, vol. 2, 2021. ISSN 2667-0569.
- Rastoceanu, F.; Rughinis, R.; Ciocirlan, S.; Enache, M.; Sensor-Based Entropy Source Analysis and Validation for Use in IoT Environments; Electronics, vol. 10, no. 10, 2021.
- Blackstone, J.; Using Blinking to Mitigate Passive Side Channel Attacks and Fault Attacks; Doctoral thesis, 2021.
- Abdelhadi, A. M. S.; Sha, E.; Bannon, C.; Steenland, H.; Moshovos, A.; Noema: Hardware-Efficient Template Matching for Neural Population Pattern Detection; MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 522 - 534, 2021.
- Nematirad, R.; Pahwa, A.; Solar Radiation Forecasting Using Artificial Neural Networks Considering Feature Selection; 2022 IEEE Kansas Power and Energy Conference (KPEC 2022), 2022.

- Pham, K. H.; Tran, T. H.; Nguyen, T. P.; Pham, C. K.; An Efficient Masking Method for AES Using Tower Fields; 2022 IEEE Ninth International Conference on Communications and Electronics (ICCE), 2022. ISBN; 978-1-6654-9745-9.
- Saini, A.; Tsokanos, A.; Kirner, R.; CryptoQNRG: a new framework for evaluation of cryptographic strength in quantum and pseudorandom number generation for key-scheduling algorithms; The Journal of Supercomputing, 2023. ISSN 1573-0484.

[A.9] Petr Socha, Vojtěch Miškovský, Hana Kubátová and Martin Novotný. Correlation Power Analysis Distinguisher Based on the Correlation Trace Derivative. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 565-568. IEEE, 2018.

The paper has been cited 4 times, in:

- Wang, M.; Huang, K.; Wang, Yi; Wu, Z.; Du, Z.; A novel side-channel analysis for physical-domain security in cyber-physical systems; International Journal of Distributed Sensor Networks, vol. 15, no. 8, 2019. ISSN 1550-1477.
- Randolph, M.; Diehl, W.; Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman; Cryptography, vol. 4, no. 2, 2020.
- Blackstone, J.; Using Blinking to Mitigate Passive Side Channel Attacks and Fault Attacks; Doctoral thesis, 2021.
- Zonios, C.; Tenentes, V.; REVOLVER: A Zero-Step Execution Emulation Framework for Mitigating Power Side-Channel Attacks on ARM64; 2022 IEEE 28th International Symposium on On-line Testing and Robust System Design (IOLTS 2022), 2022. ISSN 1942-9398.

[A.10] Petr Socha, Jan Brejník and Matěj Bartík. Attacking AES implementations using correlation power analysis on ZYBO Zynq-7000 SoC board. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pages 1-4. IEEE, 2018.

The paper has been cited 10 times, in:

- De Los Reyes, E.M.; Sison, A.M.; Medina, R.P.; Modified AES cipher round and key schedule; Indonesian Journal of Electrical Engineering and Informatics, vol. 7, pp. 29 - 36, 2019. ISSN 2089-3272.
- Gui, Y.; Tamore, S.; Siddiqui, A.; Saqib, F.; A Key Update Scheme for Side-Channel Attack Mitigation; 2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT, IoT and AI (IEEE HONET-ICT 2019), pp. 187 - 188, 2019.
- Prinetto, P.; Roascio, G.; Hardware Security, Vulnerabilities, and Attacks: A Comprehensive Taxonomy; Proceedings of the Fourth Italian Conference on Cyber Security (ITASEC 2020), vol. 2597, pp. 177 - 189, 2020. ISSN 1613-0073.
- Reifler, M.; Zollinger, M.; Embedded Secure Boot Test Suite; Bachelor's thesis, 2020.
- Reynaud, V.; Accès sécurisé aux ressources de test IEEE 1687 et aux crypto-processeurs légers dans le contexte des IoT; Doctoral thesis, 2021.
- Mozipo, A. T.; Acken, J. M.; Power Side Channel Attack of AES FPGA Implementation with Experimental Results using Full Keys; 2021 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS), 2021. ISBN 978-1-6654-2542-1.
- Hafeez, M.; Hazzazi, M.; Tariq, H.; Aljaedi, A.; Javed, A.; Alharbi, Adel R.; A Low-Overhead Countermeasure against Differential Power Analysis for AES Block Cipher; Applied Sciences, vol. 11, no. 21, 2021.
- Trautmann, J.; Teich, J.; Wildermann, S.; Characterization of Side Channels on FPGA-based Off-The-Shelf Boards against Automated Attacks; 2022 IEEE 30th International Symposium On Field-Programmable Custom Computing Machines (FCCM 2022), pp. 168 - 176, 2022. ISSN 2576-2613.
- Bergstedt, M. A.; Malware Detection Using Electromagnetic Side-Channel Analysis; Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2022.
- Talaki, E.; A memory hierarchy protected against side-channel attacks; Doctoral thesis, 2022.

[A.11] Petr Socha, Vojtěch Miškovský and Martin Novotný. SICAK: An open-source SIde-Channel Analysis toolKit. In *2019 8th Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE)*. 2019.

[A.12] Petr Socha, Vojtěch Miškovský and Martin Novotný. First-Order and Higher-Order Power Analysis: Computational Approaches and Aspects. In *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1-5. IEEE, 2019.

The paper has been cited 1 time, in:

- Moucha, P.; Ochrana šifry PRESENT prostřednictvím falešných a vícenásobnýcch rund na FPGA; Bachelor's thesis, 2020.

[A.13] Petr Socha, Vojtěch Miškovský, Hana Kubátová and Martin Novotný. Efficient algorithmic evaluation of correlation power analysis: Key distinguisher based on the correlation trace derivative. *Microprocessors and Microsystems*, 71:102858, 2019.

The paper has been cited 1 time, in:

- Cheng, K.; Song, Z.; Cui, X.; Zhang, L.; Hybrid Denoising Based Correlation Power Analysis for AES; 2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2021. ISBN 978-1-7281-8535-4. ISSN 2693-2776.

[A.14] Petr Socha, Vojtěch Miškovský and Martin Novotný. A fair experimental evaluation of distance correlation side-channel distinguisher. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1-4. IEEE, 2022.

[A.15] Tomáš Přeučil, Petr Socha and Martin Novotný. Implementation of the Rainbow signature scheme on SoC FPGA. In *2022 25st Euromicro Conference on Digital System Design (DSD)*, pages 513-519. IEEE, 2022.

[A.16] Petr Jedlička, Lukáš Malina, Petr Socha, Tomáš Gerlich, Zdeněk Martinásek and Jan Hajný. On Secure and Side-Channel Resistant Hardware Implementations of Post-Quantum Cryptography. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pages 1-9. ACM, 2022.