On the real-world security of cryptographic primitives

From theory to practice

Proefschrift

ter verkrijging van de graad van doctor

aan de Technische Universiteit Delft,

op gezag van de Rector Magnificus prof.dr.ir T.H.J.J. van der Hagen,

voorzitter van het College voor Promoties,

in het openbaar te verdedigen op dinsdag 17 oktober 2023 om 12:30 uur.

door

Zakaria NAJM

Master of Science in Mathematics and Computer Science,
University of Grenoble,

geboren te Bourgoin-Jallieu, France.

Dit proefschrift is goedgekeurd door de promotoren.

Samenstelling promotiecommissie bestaat uit:

Rector magnificus,	voorzitter
Prof. dr. P. Hartel	Technische Universiteit Delft, promotor
Dr. S. Picek,	Technische Universiteit Delft, copromotor
Onafhankelijke leden:	
Prof.dr. P.R. Schaumont	Worcester Polytechnic Inst., USA
Prof.dr.ir. N. Mentens	U. Leiden, NL
Prof.dr. G. Smaragdakis	Technische Universiteit Delft
Prof.dr. M. Conti	TUD/ U. Padua, Italy EEMCS
Prof.dr.ir. R.L. Lagendijk	Technische Universiteit Delft, reservelid

Keywords: Cyber Security, Information Security, Side Channel Attack, Cryptography, Implementations

Printed by: Ipskamp Printing, Enschede

Front & Back: Zakaria Najm

TU Delft PhD Series, Delft 2021

ISBN 978-94-6384-497-0

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Contents

I

Introduction

1

Introduction

1.1. Background

In today's digital age, data has taken on a new level of significance and has even been referred to as the new gold or new oil [1]. This is due to the increasing value placed on data as an indispensable asset in the digital economy. The ability to store, process, and analyze vast amounts of data has made it possible to gain insights and make informed decisions, creating new opportunities and driving innovation across various industries. However, the abundance of data also creates new challenges, including protecting sensitive information from unauthorized access, theft, and manipulation. Cryptography provides a powerful solution for securing data and ensuring stored and transmitted information's confidentiality, integrity, and authenticity. Cryptographic methods are applied at every data lifecycle stage, from collection and storage to transmission and usage. The demand for cryptography and encryption has risen across various industries, such as finance, healthcare, and government, and is expected to continue growing in the future [2]. The proliferation of communication networks and connected devices, such as the Internet of Things (IoT), has increased the risk of attacks on these systems, which store and handle sensitive information. As such the importance of cryptography in ensuring the security of sensitive information in the digital world cannot be overstated. The fundamental components of cryptography, known as cryptographic algorithms, serve to assure the confidentiality, integrity, and authenticity of stored or communicated data. Key generation primitives, hashing, and encryption primitives make up these algorithms. Key generation primitives generate unique digital keys for encryption and decryption processes, often using a random number gener-

ator. Hashing primitives generate a fixed-length hash from a variable-length input, ensuring data integrity by detecting any modifications to the original. Encryption primitives scramble data, rendering it unreadable without the proper decryption key, preserving confidentiality. These primitives provide the secure foundation for cryptography and are utilized in various cryptographic protocols to guard against unauthorized access, tampering, and exposure.

Moreover, cryptographic algorithms were traditionally designed based on the designers' experience and intuition rather than solid mathematical proof. The security of these algorithms was evaluated using generic techniques such as brute force attacks or linear and differential cryptanalysis. However, these techniques may only sometimes give a comprehensive or accurate understanding of an algorithm's security.

The practicality of security is often guided by experience and intuition, while mathematical proofs offer a theoretical sense of security. These two ideas of security can sometimes conflict. For example, two groups emerged with differing approaches to lattice-based schemes during the NIST post-quantum cryptography competition. The first group focused on provably secure schemes that relied on Gaussian sampling. However, these schemes were found to be costly to implement and challenging to protect against side-channel attacks. In contrast, the second group employed uniform sampling-based schemes, which, although lacking provable security, were demonstrated to be less expensive to implement and easier to secure against side-channel attacks [3]. This situation raises a critical question for the field: Should we prioritize the development of systems with ease of security proof, or should we favor schemes that have been empirically shown to be practical and secure?

In the context of cryptographic system design, it is generally preferred to use schemes empirically shown to be secure rather than those with only provable security. However, that is difficult to implement and protect against side-channel attacks. This is because practical security is a critical consideration in real-world applications, and security proof is only one aspect of overall security.

However, using schemes with provable security is still highly valuable in situations where security is paramount, such as in highly sensitive military or government applications. In such cases, the additional security provided by a provably secure scheme may be worth the added implementation complexity and resource usage.

Therefore, the choice of which scheme to use ultimately depends on the application's specific requirements, as well as the available resources and constraints. While schemes with provable security may have their drawbacks, they are still a vital area of research and development in cryptography.

An example of empirically secure ciphers widely used in practice is the arithmetic-based non-linear operation ciphers, such as SHA-1 and CHACHA-poly-20. While these schemes have been demonstrated to be secure in practice and highly efficient, they are challenging to formalize. Therefore, they have not received as much attention from researchers as Boolean Pseudo-Random Functions (PRF) based ciphers [4]. However, the attractiveness of PRF-based ciphers has led to extensive research and publication, which ultimately contributes to an additional level of trust in the security of these schemes.

Cryptographic algorithms are mathematical procedures used for the encryption and decryption of data. They form the foundation of secure communication by transforming plaintext data into ciphertext, rendering it unreadable to unauthorized parties. Examples of cryptographic algorithms include symmetric algorithms like Advanced Encryption Standard (AES) and asymmetric algorithms like RSA.

On the other hand, cryptographic protocols are sets of rules and procedures that govern the secure exchange of information between parties using cryptographic algorithms. Protocols define how the algorithms are to be used, including the generation, distribution, and management of cryptographic keys and handling authentication, confidentiality, and integrity of messages. Examples of cryptographic protocols include Secure Sockets Layer (SSL), Transport Layer Security (TLS), and Pretty Good Privacy (PGP).

Cryptographic algorithms provide the mathematical foundation for encryption and decryption, while cryptographic protocols establish a structured framework for secure communication that leverages these algorithms. By introducing both concepts on page three, readers will clearly understand their distinct roles and how they work together to ensure secure data transmission.

The field of cryptography is in a state of constant evolution. As a result, new weaknesses and security breaches are frequently uncovered that can render cryptographic algorithms and protocols vulnerable.

Regular updates, as well as solid underlying infrastructure, are essential to maintain their security. The recent developments in quantum computing have the potential to break many of the encryption algorithms widely used today, making it essential to research and develop post-quantum cryptographic algorithms that are secure against quantum computers. These post-quantum algorithms are expected to be more robust and secure than current algorithms and will be crucial in maintaining the security of digital communication and transactions in the future [5].

The shift to post-quantum cryptography requires a thorough and well-planned approach that involves a range of stakeholders, including researchers, industry leaders, and government agencies. The transition is a complex and multi-faceted pro-

cess that entails updating the cryptographic algorithms and the underlying hardware, software, and communication protocols that support these algorithms. To ensure a smooth and successful transition, it is imperative to take into consideration the far-reaching implications it will have on digital communication and transactions and to implement measures that will minimize any potential risks and disruptions during the process.

As the field of cryptography evolves, and new attacks and vulnerabilities are discovered, security models for cryptographic algorithms and protocols become more complex. Indeed, a security model is a mathematical representation of the cryptographic algorithm or protocol and the environment in which it is used. It describes the assumptions and requirements that must be met for the algorithm or protocol to be considered secure. As the gap between theory and practice is studied, security models for cryptographic algorithms and protocols become more complex and consider a broader range of attack scenarios and potential vulnerabilities. This can make it more challenging to analyze and evaluate the security of a given algorithm or protocol. However, it is necessary to ensure that security is robust against all known and potential attacks.

It is crucial to develop methods to assess security and identify bugs that could affect the security of an implementation. There are several methods that can be used to assess the security of cryptographic algorithms and protocols. These are mathematical analysis, formal verification, and testing. The mathematical analysis involves using mathematical proofs and models to show that an algorithm or protocol is secure under a given set of assumptions. Formal verification uses mathematical proof systems to prove that an algorithm or protocol satisfies specific properties. Testing involves running the algorithm or protocol in a simulated or natural environment to identify bugs or weaknesses.

It is important to note that no single method can guarantee the security of an algorithm or protocol. Instead, a combination of methods is often used to provide a comprehensive assessment of the security of an algorithm or protocol. In addition, it is also essential to conduct regular security audits of the implementation of cryptographic algorithms and protocols to identify any bugs that may affect security. This can include a manual review of the code and automated testing of the implementation to identify any vulnerabilities or weaknesses. In summary, a methodology that combines mathematical analysis, formal verification, testing, and security audits is essential to ensure the security of cryptographic algorithms and protocols. The implementation of a cryptographic primitive may result in the leakage of secret information during computation through side-channels. Side-channels are unintended information channels that can leak sensitive information about the implementation of an algorithm. Examples of side-channels are power consumption, electromagnetic radiation, and timing information. Attackers can use

these side-channels to extract secret keys or other sensitive information by measuring the physical characteristics of the implementation. Another way to attack an implementation is the active side-channel attack, where an attacker actively disrupts the computation of the algorithm to leak secrets. With the rapid evolution of technology and advances in cryptanalysis research, older cryptographic primitives such as SHA-1 are showing their limitations. SHA-1 is a cryptographic hash function widely used in the past for digital signatures and other applications requiring data integrity and authenticity. Researchers have discovered new attacks that can effectively break SHA-1 and make the cryptographic hash function vulnerable to collisions. As a result, the use of SHA-1 has been deprecated, and experts recommend using more powerful cryptographic hash functions like SHA-2 or SHA-3. It is important to note that cryptographic primitives have a limited lifetime and must be regularly reviewed and updated to maintain their security. The gap between theory and practice in cryptography can create potential security problems. Researchers focus on theoretical topics, while practitioners focus on providing standards-compliant implementations. This can lead to a situation where disproportionate theoretical security margins are used. In contrast, hackers can, in some situations, easily exploit side-channel weaknesses in the implementation to break the cryptosystem, regardless of the theoretical security margins.

The security margin for a cryptographic algorithm is defined by the difference between the current best-known attack against the algorithm and the theoretical maximum security provided by the algorithm, usually expressed in terms of computational steps or resources required for the attack.

In simpler terms, the security margin is a measure of how resistant an algorithm is to attacks, and it is used to gauge the overall security of a cryptographic system. A larger security margin indicates that an algorithm is more secure, as it would require significantly more time, computational power, or other resources for an attacker to break it.

The security margin depends on various factors, including:

Key size: A larger key typically provides a higher security margin, as it increases the number of possible keys, making it more difficult for an attacker to guess the correct Key through brute force.

Algorithm design: Some cryptographic algorithms are inherently more secure than others due to their design and resistance to known attacks, such as differential or linear cryptanalysis.

Implementation: The security of a cryptographic algorithm can also depend on the quality of its implementation, including how well it resists side-channel attacks or other implementation-specific vulnerabilities.

Attack complexity: The amount of time, computational resources, or other resources required for an attacker to successfully break the algorithm. The higher the complexity, the larger the security margin.

State-of-the-art attacks: The current best-known attacks against a specific cryptographic algorithm. As new attacks are discovered, or existing attacks are improved, the security margin of an algorithm may decrease.

In practice, the security margin is often estimated based on the current state of cryptographic research and attack techniques, and it is subject to change as new attacks are discovered or existing attacks are refined. For bloc cipher, it is referred to as the number of extra computation rounds for which the claimed security bound still holds even if they are removed.

For example, AES has a security margin of three to four rounds in single-key setting, excluding the biclique attack, meaning that if one round is attacked, the attacker would still have to attack two to three rounds and start to erode the security claims. A security margin of three to four rounds in the context of a block cipher refers to the number of additional rounds of encryption that would need to be successfully attacked by an adversary beyond any compromised rounds for the cipher's security to be significantly weakened.

For example, if a block cipher has ten rounds, and an attacker successfully compromises one round, the remaining security margin would be nine rounds. In the case of a security margin of three to four rounds, the attacker would need to compromise an additional two to three rounds (in addition to the one already compromised) before the overall security of the cipher is significantly weakened.

This security margin is a measure of the strength of the block cipher against attacks and is an important consideration in the evaluation of cryptographic algorithms. Generally, a higher security margin provides greater confidence in the cipher's security.

It is important to note that a time-memory trade-off determines the security of a block cipher. As a result, a reduction in time complexity without a corresponding decrease in data or memory complexity may not significantly impact the cipher's overall security claims.

A biclique attack is a type of cryptanalytic attack that targets block ciphers, which are symmetric key cryptographic algorithms operating on fixed-size data blocks. The biclique attack was first introduced by Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger in 2011 to improve the known attacks on the Advanced Encryption Standard (AES).

The biclique attack is based on partitioning the cipher into two parts and finding

a so-called "biclique," a set of keys that allows an attacker to bypass some of the intermediate computations in the cipher. By doing this, the attacker can significantly reduce the overall computational complexity of an exhaustive Key search, making the attack more efficient than a brute-force attack.

In the case of AES, the biclique attack reduced the computational complexity of key recovery by a minor factor. However, it is essential to note that the practical impact of the biclique attack on AES is quite limited. The attack still requires enormous computational power to succeed, and more than a reduction in complexity is needed to make AES insecure for most applications. While the biclique attack is a noteworthy development in cryptanalysis, its practical implications are relatively limited for well-designed cryptographic algorithms like AES. As of the time of writing, there are still questions regarding the real-world impact of such attacks in terms of their practical implementation using state-of-the-art technologies.

It is noteworthy that new properties of AES continue to be uncovered. For instance, the novel representations of the AES key schedule by Leurent and Pernot [6] were only recently described, nearly two decades after its initial analysis [7].

The attack serves as a reminder that the cryptographic community must continually research and develop new methods to ensure the long-term security of encryption algorithms ranks depending on the version, and AES 128 requires about 2^{128} evaluations to be attacked under a single key, which is considered infeasible for the foreseeable future. However, a cache-timing attack and other forms of side-channel attacks can recover the secret Key of a vulnerable AES implementation in just a few minutes [7].

Professionals frequently need to utilize obsolete standards despite the lack of proof concerning any associated risks. Deprecated primitives that are unused but remain in implementations for compatibility purposes can be detected. This practice exposes a vulnerability that hackers could exploit in downgrade attacks, given how algorithm types are used in libraries. Understanding the hazards of using such algorithms is crucial for advancing the field. One way to achieve this is by demonstrating the feasibility and cost of potential attacks. This highlights the importance of bridging the gap between practitioners and theorists of cryptography. Theoretical research needs to be more closely linked to practical implementation, and practitioners need to understand the theoretical underpinnings of cryptography better to ensure secure implementations. One way to bridge this gap is to involve practitioners in developing and analyzing cryptographic algorithms and protocols at an early stage. This will align theoretical research more closely with practical implementation and ensure greater security of applications. Another way to bridge the gap is to include more practical evaluation and testing of cryptographic implementations, including side-channel vulnerability testing, in the development

process. This will ensure that implementations are secure not only in theory but also in practice.

To conclude, bridging the divide between cryptographic practitioners and theorists is crucial to ensure that cryptographic implementations are secure in practice and theory.

This study adopts a practical and pragmatic approach to enhancing the security of real-world systems. By examining the costs associated with various attack types, including the complexity of the attack and the resources required, the study aims to establish a better balance between security, performance, and cost in cryptography. Moreover, the study evaluates several countermeasure schemes designed to enhance the security of embedded systems and connected devices while considering real-world limitations and cost-effectiveness. Additionally, the thesis provides a comprehensive review of the current state of cryptography, including cryptographic primitives, protocols, and the challenges and constraints of existing cryptographic algorithms. We also discuss the post-quantum cryptography topic and its challenges and opportunities, offering recommendations for organizations and individuals in the transition. Overall, this thesis seeks to advance the understanding of practical trade-offs and security margins in cryptographic standards, ultimately contributing to developing robust and secure systems that balance performance and cost.

1.2. Problem statement and Research questions

This thesis focuses on the gap between cryptography theory and practice. Understanding the differences in how cryptographic algorithms are implemented and executed is critical to designing secure cryptosystems, especially in today's digital age, where the use of technology such as IoT devices, Edge computing, decentralized cloud, and blockchains is widespread. Efficient cryptosystem design is crucial as even small variations at the device or node level can have a significant impact when billions of devices or nodes are connected and exchanging data.

In this thesis, various aspects of Cryptanalysis (in Part II), Passive Side-Channel Attacks and Countermeasures (in Part III), Active Side-Channel Attacks (also known as Fault Attacks) and Countermeasures (in Part IV), and Secure Lightweight Implementations (in Part VI) are examined.

The objective is to address key questions in the field and provide a comprehensive conclusion that takes a broad view of security issues in cryptographic algorithms and suggests a framework for incorporating security into the product design process.

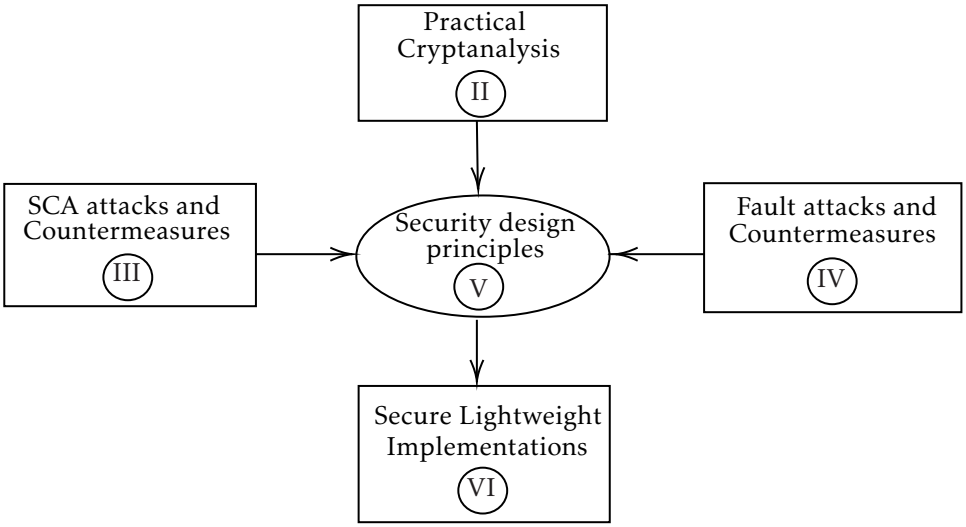


Figure 1.1: Thesis structure chart

The vastness of security means that not all aspects can be covered in this Ph.D. thesis. Nevertheless, we can redefine security from a designer's perspective by studying various topics and evaluating the current state-of-the-art. This can lead to developing a security-aware design flow that designers can utilize to achieve their desired security objectives.

1.2.1. Scope

The Figure 1.1 illustrates the scope of this Ph.D. thesis. Please note that the ellipse at the center labeled "Security design principles" pertains to establishing a set of guidelines and design principles based on a thorough security analysis from a designer's perspective. The edges of the graph depict the security constraints. The Ph.D. thesis considers three constraints: Cryptanalysis-based attacks, Passive Side-Channel Attacks, and Active Side-Channel Attacks.

In Part II, Part III, Part IV of this doctoral dissertation, a dual approach is taken to scrutinize various attacks and countermeasures from both the attacker and designer perspectives. The investigation assesses the efficacy and constraints of these measures, with particular emphasis on distinguishing between their theoretical and practical implications.

Design guidelines are established based on the three considered constraints and a review of the existing literature. These guidelines form the basis for a proposed security-aware design flow aimed at introducing more rigor in the design process

for applications that require protection against the three attack vectors.

The proposed security-aware design flow is demonstrated in a real-world scenario, specifically implementing and evaluating various contenders in the NIST Lightweight Cipher competition. The methodology was initially applied to the microcontroller implementation of the GIFT family of block ciphers and was later extended to AES [8], demonstrating its genericity.

Having outlined the thesis's scope, we will elaborate on each aspect and present the research questions.

1.2.2. The Practical Cryptanalysis (Part II)

This section explores the practical aspects of differential cryptanalysis attacks. It sheds light on the cost of a differential cryptanalysis attack when the cipher's security is near or below the 80-bit security threshold. We recognize that cryptographic methods are constantly evolving, and determining the level of security provided by encryption is an ever-changing challenge. While 80-bit encryption is generally sufficient for most applications, highly sensitive or critical applications may require higher security, such as a minimum of 128-bit encryption. Despite its claimed security level of 80 bits, the SHA-1 algorithm was still widely used in various applications and website certificates at the start of this Ph.D. thesis. As shown by Leurent et al. (2020), about 30% of the certificates on the internet still supported SHA-1, highlighting the need for regular security assessments and updates to ensure adequate protection. Understanding the cost of a chosen-prefix collision attack on the SHA-1 algorithm through ASICs and GPUs provides insight into the cost of differential cryptanalysis attacks on ciphers with 80-bit security or lower, which are still widely used in low latency and resource-constrained applications such as IoT and sensor networks where side-channel attacks pose a risk. In Chapter chapter 3, we enhance the understanding of the cost of a collision attack on SHA-1 by executing the first chosen-prefix collision on SHA-1 with the use of GPUs and designing the first SHA-1 Chosen-prefix collision ASIC, providing a precise estimate of the attack cost when using ASICs. This knowledge enables more informed choices in selecting secure cryptographic methods and protocols for real-world applications with strict constraints by setting more meaningful security margins. The following research question is addressed in this part:

RQ1: What is the actual cost of a differential cryptanalysis attack when the cipher's security is near or below the 80-bit security threshold?

1.2.3. Passive Side-Channel Attacks and Countermeasures (Part III)

This part explores the practical aspects of preventing side-channel attacks in embedded systems and IoT devices. The security of these devices is not only dependent on their ability to withstand cryptanalysis but also on their resistance to passive and active side-channel attacks. In situations where the device is low on power or requires low latency, side-channel attacks may pose a greater risk than more complex attack methods. Various countermeasures can be employed to prevent passive side-channel attacks, including masking. However, implementing a countermeasure requires balancing execution time and memory usage and preserving the security properties of the masking scheme. The masking order and the signal-to-noise ratio determine a masking scheme's security.

Linear operations in symmetric encryption have been shown to be highly susceptible to side-channel attacks due to their confusion properties, while non-linear operations are fundamental to symmetric encryption. Implementing boolean S-boxes in software can be expensive, and table-based implementations are vulnerable to timing side-channel attacks. Although boolean S-boxes can be naturally masked, arithmetic S-boxes may provide additional security benefits over boolean S-boxes like those used in AES. Masking arithmetic operations such as addition, subtraction, multiplication, and division can be computationally expensive and negatively impact cryptographic implementation performance, particularly for those requiring many arithmetic operations. Masking requires converting arithmetic operations to boolean operations, a resource-intensive process involving performing boolean operations on individual bits of operands and merging the boolean results to produce the masked output.

ChaCha20, an arithmetic addition-based cipher, has emerged as an AES alternative due to its ease of software implementation and ability to deliver constant-time implementation advantages. It would be interesting to explore whether ChaCha20 provides any advantage against side-channel attacks without masking.

Therefore, this part addresses the question:

RQ2: How can we practically prevent side-channel attacks?

1.2.4. Active Side-Channel Attacks and Countermeasures (Part IV)

The Part IV is dedicated to fault injection attacks and their impact on block ciphers and asymmetric cryptography. What methods or techniques can be employed to detect and prevent practical fault attacks? The precision of the attacker plays a crucial role in determining the success of these attacks. The two most common methods for inducing faults on a cryptosystem are using a laser beam or power glitches. However, using lasers may only sometimes be possible due to counter-

measures like light sensors. Power glitches can also be prevented with an on-chip DC-DC converter. Data path and time redundancy, achieved through codes or duplication, is another option, but it has limitations and can be costly. This is why the focus has shifted towards Electromagnetic Fault Injection (EMFI) attacks, which can be carried out with good time-space accuracy without opening the chip, and can induce complex high-order fault models.

Can electromagnetic fault injection (EMFI) detection and prevention be achieved without data-path time-space redundancy? An unreported behavior of the phase-locked loop (PLL) phase detector sensor exposed to EMFI was discovered through experimentation. A novel countermeasure is examined and assessed based on this behavior against EMFI.

Once a realistic fault model has been established, the identification of potential associated attacks by the cryptographer becomes necessary. These attacks are cipher-specific and are investigated on a case-by-case basis. To the best of our knowledge, no general framework exists to assess a cipher's vulnerability to faults provided a fault model.

A Differential Fault Attack (DFA) is a type of fault injection attack that involves intentionally introducing faults into a cryptographic implementation by altering its input or internal state. The attacker then observes the output of the faulty implementation to infer secret information, such as the secret Key used in a symmetric cipher or the private Key used in a public-key cryptosystem.

Given a fault model, how can we assess the inherent resistance of a block cipher against fault attacks? Understanding the factors that make a cipher more resilient to these attacks is valuable for improving system protection. This thesis seeks to contribute to finding an answer to this question by conducting a case study on DFA.

While physical countermeasures can enhance system security, combining them with logical countermeasures can be even more effective. However, logical countermeasures that are proven based on simple fault models like DFA may not be sufficient in real-world scenarios. Implementing duplication-based countermeasures in Elliptic Curve Cryptography can also be costly.

How effective is a real-world secure, proven implementation of elliptic curve scalar multiplication against electromagnetic fault injection attacks?

A technology feature developed to mitigate side-channel attacks or enhance system performance may inadvertently create opportunities for other types of attacks. Therefore, a thorough assessment of a system's security and consideration of all possible risk factors before its release to the market is crucial. An example of this is demonstrated by using a reconfigurable lookup table (RLUT) in an FPGA, which

can increase a design's resistance against SCA but also offers a potential means for active adversaries to insert trojans into the design.

Therefore, this part addresses the question:

RQ3: What methods or techniques, can be employed to detect and prevent practical fault attacks?

1.2.5. Design considerations and guidelines (Part V)

In Part V, the utilization and selection of security algorithms are discussed. The definition of security from a designer's standpoint can be described as an additional constraint, precisely, as the fourth architectural design constraint, in addition to the three well-known existing ones: Power, Speed, and Price. A proposed security-aware design flow that considers the three security constraints investigated in this thesis, namely Cryptanalysis, Side-Channel Attacks, and Fault Injection Attacks, can be followed to determine the optimal trade-off for different security applications.

In this part (Part V), we will discuss the utilization and selection of security algorithms. From a designer's perspective, security can be considered an additional constraint that must be considered alongside the three well-known architectural design constraints of Power, Speed, and Price. We want to demonstrate that security can be considered the fourth design constraint for any system through the study of state of the art in different security domains, including Cryptanalysis, Side-Channel Attacks, and Fault Injection Attacks.

From our observation in the previous chapters and the study of the state-of-the-art, how can we introduce more rigor in the design process of a system subject to different security constraints?

To ensure optimal security, we propose a security-aware design flow that considers the three security constraints studied in this thesis to determine the optimal trade-off for a targeted application.

The proposed security-aware design flow could include the following high-level steps:

- Threat Model and Risk Assessment: Identify potential threats to the system and evaluate associated risks. This step provides a comprehensive understanding of the system's security requirements.
- Selection of Security Mechanisms: The appropriate security mechanisms can be selected once the security requirements are understood. These mechanisms can be chosen based on their ability to mitigate the identified threats

and reduce the associated risks.

- Algorithm Selection: The appropriate cryptographic algorithms can be chosen after selecting the security mechanisms.
- The selection process should consider the security strengths and weaknesses of the different algorithms and ensure they are suitable for the specific application.
- Implementation and Testing: The selected algorithms should be implemented in the system and thoroughly tested to ensure that they provide the necessary security levels.
- Verification and Validation: The implemented system should be verified and validated to ensure that it meets the specified security requirements.

Overall, in this part, we explore potential guidelines or principles that designers could utilize to enhance the level of rigor in the design process of a security-critical application. We address the following research question: **RQ4:** What potential guidelines or principles could designers utilize to enhance the level of rigor in the design process of a security-critical application?

1.2.6. Application of design guidelines: Lightweight cipher secure implementations (Part VI)

In Part VI, the focus is on implementing the proposed security-aware design flow to address the challenge of creating secure embedded software implementations for lightweight ciphers. The proliferation of IoT devices has increased the use of general-purpose microcontrollers with integrated hardware cryptographic modules and software cryptographic stack implementations. The selection of one of these solutions is primarily influenced by cost considerations, with hardware solutions providing better power performance but being more costly to replace if a vulnerability is identified. FPGA-based solutions can provide the benefits of both approaches but at a higher cost.

In 2021, the NIST launched a competition to select a future lightweight authenticated cipher cryptographic standard. This competition aims to find a cipher that offers a better trade-off than AES, considering the latest advancements in cryptography and hardware security. Designers face a challenge in finding a lightweight cipher that is efficient in software and hardware. Lightweight ciphers with low S-Box gate counts are Side-Channel masking friendly, but their linear parts can be costly to process in software.

The objective is to find an algorithm that surpasses AES regarding gate count

and software cycle count while providing additional security properties suitable for the widespread deployment of IoT and Edge devices.

The goal was to assess the performance of NIST lightweight cryptography competition candidates versus AES regarding the hardware-software secure implementation trade-off. The GIFT family of block ciphers utilized by several NIST Lightweight Competition contenders was selected for evaluation due to its advantageous hardware implementation characteristics. Despite the favorable hardware implementation, the software implementation of GIFT is considered complex and inefficient because of the bit permutation in its linear layer. To address this issue, the second chapter of the thesis presents a generic bit-slicing approach for efficient, constant-time software implementation of SPN applied to the GIFT family of block ciphers. This implementation methodology is also adaptable to other ciphers, including AES, and aims to provide a fair basis for comparing secure cipher implementations that are constant in time.

Automated formal proof verification for bit-slice implementations is advantageous when utilizing first and second-order logic proof verifiers. Nevertheless, the proof process generally requires a simplified model and relies on extensive assumptions concerning the executing processor's architecture. Various formally proven implementation methodologies will be examined and validated on real microcontrollers used in the Internet of Things to investigate the disparity between theory and practicality.

1

1.2.7. Research questions summary

In summary, here are the questions addressed in this Ph.D. thesis :

*RQ*1

What is the actual cost of a differential cryptanalysis attack when the cipher's security is near or below the 80-bit security threshold?

*RQ*2

How can we practically prevent side-channel attacks?

*RQ*3

What methods or techniques, can be employed to detect and prevent practical fault attacks?

*RQ*4

What are some potential guidelines or principles that designers could utilize to enhance the level of rigor in the design process of a security-critical application?

1.3. Outline and Contributions

The structure of the thesis is as follows:

Each chapter of the thesis includes at least one publication referenced at the end of the chapter. The different chapters of this thesis are integral versions of published papers, and hence, the terminology and notations might vary across chapters.

1.3.1. Part II: The Practical Cryptanalysis

In chapter 2, we provide a preliminary survey on the history of hardware crackers. Cryptanalysis is a crucial aspect of cryptography. It serves the purpose of breaking ciphers for malicious purposes and forming the foundation for constructing secure ones. The most commonly used ciphers are considered secure because they have been continuously tested and not successfully broken by cryptanalysts. Although successful cryptanalysis may prove a cipher's vulnerability, the attack is often only theoretical due to its

difficulty in efficiently executing it. For instance, the vulnerability behind the SHA-1 collision attack discovered in 2017 was already known in 2005. However, due to a lack of resources and algorithms, it took until 2016 for it to be addressed by the Internet and IT industries. Advancements in attack algorithms, implementation techniques, and hardware fabrication have allowed for the practical execution of cryptanalysis. This survey covers these efforts and delves into quantum computers' impact on cryptography and cryptanalysis. The survey is divided into three sections: cryptanalytic attacks with specific implementation needs, previous cryptanalytic machines, and quantum computers.

This chapter is an integral copy of the paper "Crack Me if you can: hardware acceleration bridging the Gap between Practical and theoretical cryptanalysis?: A Survey. SAMOS 2018: 167-172".

Then in chapter 3, we study the cost of the SHA-2 chosen prefix collision on ASIC. In 2017, the SHA-1 hash algorithm was practically broken using an identical-prefix collision attack implemented on a GPU cluster, and in 2020, a chosen-prefix collision was first computed with practical implications for various security protocols. This raised questions about the cost of performing such attacks and the best software/hardware cryptanalysis technology. This paper addresses these questions by examining the challenges and costs of building an ASIC cluster for attacking a hash function. The study considers different scenarios and includes two cryptanalytic strategies - a classical birthday search and a differential attack using neutral bits for SHA-1.

The results show that for generic attacks, GPUs and ASICs pose a severe threat to primitives with 64-bit security, with rented GPUs being a good solution for a one-time attack and ASICs being more efficient for multiple attacks. ASICs also pose a significant security risk for primitives with 80-bit security. For differential attacks, GPUs (purchased or rented) are often a cost-effective choice. However, ASIC provides an alternative for an organization with the financial means to afford the initial cost and look for a compact, energy-efficient solution. In the case of SHA-1, we show that an ASIC cluster costing several million dollars could generate chosen-prefix collisions in a day or even a minute, extending the attack surface to TLS and SSH.

This chapter is an integral copy of the paper "On the Cost of ASIC Hardware Crackers: A SHA-1 Case Study. CT-RSA 2021: 657-681".

1.3.2. Part III: Passive Side-Channel Attacks and Countermeasures

The threat of side-channel attacks is a significant concern for secure systems. In chapter 4, we study the resistance of two ciphers commonly used in the automotive industry - AES and ChaCha20 - against such attacks. The focus is on the analysis of the resistance of the non-linear component. ChaCha20 is naturally resistant to timing-based side-channel attacks, making it suitable for vulnerable applications. However, protecting it against power side-channel attacks is more challenging and requires higher overhead than AES.

This chapter is an integral copy of the paper "On Comparing Side-channel Properties of AES and ChaCha20 on Microcontrollers. APCCAS 2018: 552-5558".

In chapter 5, we took a deeper dive into the security properties of the masked table recomputation, which is commonly used in memory-constrained devices. We show that a higher order of masking does not necessarily mean more security once the sensitive point of interest is known. This work is an integral copy of the paper "Multivariate High-Order Attacks of Shuffled Tables Recomputation" published in Journal of Cryptology 2018 p351-393.

Identifying points of interest in a grey-box or black-box setting is crucial in Side Channel Attacks and Common Criteria evaluations. ECC (Elliptic Curve Cryptography) is a widely used public key cryptosystem for various real-world applications. The rise of side-channel attacks has raised concerns about the security of ECC implementations, as these attacks can exploit physical leaks to break even theoretically secure ciphers. Non-profiled side-channel attacks, which can work in almost a black-box setting, are considered more severe than profiled attacks. The challenge in such attacks is the selection of relevant features from the side-channel signal, as the measurement data often contain irrelevant points that can negatively impact the attack's effectiveness. This issue is particularly pronounced in non-profiled black-box scenarios.

In chapter 6, we explore different feature selection approaches to improve the accuracy of non-profiled attacks on ECC. The proposed methods are tested on actual measurements from FPGA and microcontroller targets and achieve accuracy comparable to the profiled case.

This chapter is an integral copy of the paper "Feature selection methods for non-profiled side-channel attacks on ECC" published in DSP 2018: 1-5".

1.3.3. Part IV : Active Side-Channel Attacks and Countermeasures

Countermeasures against fault attacks are an essential aspect of cryptographic system design. Fault attacks are a type of attack that takes advantage of errors or faults introduced in a cryptographic system to extract sensitive information. Fault attacks can be applied to both symmetric and asymmetric cryptographic systems and can be particularly devastating in cases where the keys used in the system are compromised.

Various countermeasures can be used to protect against fault attacks on both symmetric and asymmetric cryptographic systems. In asymmetric cryptography, one approach is to use error-correcting codes to implement the system. This helps ensure that any faults or errors introduced during the computation process are detected and corrected before attackers can exploit them. Another approach is to use redundancy in the implementation of the system, such as using multiple implementations to verify the correctness of the results.

Countermeasures against fault attacks typically involve redundancy and randomization techniques in symmetric cryptography. For example, one approach uses redundant computations, where multiple instances of the cryptographic function are computed, and the results are combined using error detection and correction techniques. Another approach is to use randomization, where random values are introduced into the cryptographic function to make it more difficult for an attacker to exploit faults.

However, it is essential to note that implementing these countermeasures comes with practical trade-offs. The addition of error detection and correction codes or redundancy can result in increased computational overhead, which can impact the performance of the cryptographic system. Additionally, using randomization techniques can increase the system's computational complexity. Therefore, designers must carefully balance the need for security against the practical considerations of performance and resource utilization.

Substitution-Permutation Network (SPN) is a standard method for designing block ciphers in cryptography. Differential fault cryptanalysis attack or DFA can be used to break every known existing block cipher. The complexity of the DFA depends on the precision of the fault model. The ultimate attacker can fault a single bit, and the single-bit flip can be induced at a different location in time and space, leading to various key-extraction complexities. The complexity of the DFA depends as well on the bloc cipher structure.

In chapter 7, we propose a generic method for evaluating the resistance of

any SPN block cipher against DFA. This work is an integral copy of the paper "SoK: On DFA Vulnerabilities of Substitution-Permutation Networks in ACM AsiaCCS 2019 p403-414". The aim of this work is to identify feasible compromises when designing fault-resistant implementations of block ciphers.

Private Circuits II is a known provable defense against strong attackers who have the ability to access a limited number of internal nodes. The ultimate DFA attacker model is an attacker able to access and arbitrarily flip the value of any internal node. The resistance against such an attacker is done by introducing gate-level redundancy. It should be noted that targeting specific nodes is difficult in practice. Therefore, studying the cost and effectiveness of such countermeasures in practice is essential.

In chapter 8, we present the first implementation of Private Circuits II on an FPGA, which is secure against the reading or reset of one wire chosen by the attacker. The implementation uses a Spartan 6 Xilinx FPGA and features a throughput of 142 Mbit/s. The security of the design is analyzed, revealing that despite the countermeasure, some exploitable ciphertexts can still be outputted. This is due to correlated faults that result in a differential fault attack, but only if a steady fault injection setup is used. Otherwise, the faults result in non-exploitable ciphertexts.

This work is an integral copy of the paper: "Private circuits II versus fault injection attacks" published in "ReConFig 2015: 1-9".

In chapter 9, we address the issue of fault injection attacks on asymmetric cryptography and present countermeasures that ensure the reliability of computation results against these attacks. We focus on studying the modular extension protection scheme and its variants applied to elliptic curve scalar multiplication (ECSM) algorithms. We discovered a flaw in an existing countermeasure and proposed a new, test-free variant to fix it. Our results showed that this new method provides improved security as the fault non-detection probability decreases with increased security parameters. Furthermore, we implemented this countermeasure on an ECSM algorithm for Edwards and twisted Edwards curves on an ARM Cortex-M4 microcontroller to demonstrate its efficiency.

This work is an integral copy of the paper:" Using modular extension to provably protect Edwards curves against fault attacks" published in "The Journal of Cryptographic Engineering. 7(4): 321-330 (2017)".

The evaluation of the ability of attackers to break a cryptosystem using DFA is often done using lasers by evaluation labs. Lasers offer exceptional time-space accuracy for DFA, leading most hardware security modules to imple-

ment time and space redundancy and light sensors as a defense mechanism. However, light sensors can be bypassed through the use of Electromagnetic Fault Injection (EMFI), which is currently the subject of active research in the community.

EMFI has a reasonably good spatial and perfect time precision, bypassing time-redundancy-based countermeasures. For cost reasons, it is only sometimes possible to include spatial redundancy. Hence, a low-cost physical sensor-based countermeasure is investigated in chapter 10. This work is an integral copy of the paper "PLL to the Rescue: a novel EM fault countermeasure" in ACM/IEEE Design Automation Conference (DAC'2016)".

Finally, in chapter 11, the use of the Dynamic Reconfigurable LUT (RLUT) feature in contemporary FPGAs for secure applications is explored. The fundamental functionality and potential of RLUT for security are described, followed by an investigation of its application through case studies examined from both design and hacking perspectives.

This work is an integral copy of the paper "Reconfigurable LUT: A Double-Edged Sword for Security-Critical Applications published in "SPACE 2015: 248-268".

1.3.4. Part V: Design considerations and guidelines

The topic of Chapter 12 centers on the application and selection of security algorithms, which involves redefining the concept of security as a fourth architectural design constraint for designers to consider. In addition to the three well-known constraints of power, speed, and price, we introduce security as an essential aspect of the design process. To address this, we propose a security-aware design flow, which begins with selecting cryptographic primitives, protocols, and overall system design.

It is worth noting that this work is based on the paper "Security is an Architectural Design Constraint," which was published in the "Journal of Microprocessors and Microsystems, Volume 68." As such, the proposed design flow is built on previous research and aims to provide a comprehensive approach to security in microprocessor and microcontroller designs and implementations.

Overall, in this chapter, we explore potential guidelines or principles that designers could utilize to enhance the level of rigor in the design process of a security-critical application. We address the following research question: **RQ4:** What potential guidelines or principles could designers utilize to enhance the level of rigor in the design process of a security-critical application?

1.3.5. Part VI: Application of design guidelines: Lightweight cipher secure implementations

The chapter 13 focuses on the implementation performance of lightweight ciphers in software.

The fixslicing implementation technique has been applied to GIFT, a fundamental building block utilized by several NIST Lightweight Cryptography (LWC) finalist candidates. This study reveals that lightweight ciphers with exceptional hardware-oriented designs can also achieve impressive performance in software, surpassing AES. The methodology employed in this research proves to be valuable in comparing different cipher implementations on a fair and unbiased basis.

This work is an integral copy of the paper "Fixslicing: A New GIFT Representation Fast Constant-Time Implementations of GIFT and GIFT-COFB on ARM Cortex-M" published at TCHES 2020.

This section serves as an application of our design guidelines and contributes to answering the research question RQ4. Through this work, we aim to demonstrate that incorporating more rigor in the implementation process of security primitives enables a fair comparison of implementation performance for different ciphers.

II

The Practical Cryptanalysis

The main focus of this section is on the first constraint shown in Figure 1.1, which is the ability of cryptographic algorithms to withstand cryptanalysis. Specifically, we will be examining the cost of a real-world differential cryptanalysis attacks on the widely used cryptographic hash function, SHA-1. The goal is to provide an estimate of the cost of an attack that has a complexity below the 80-bit security bound, which is the current standard for secure cryptographic algorithms. By analyzing the cost of such attacks, we can better understand the overall strength of SHA-1 and other similar lightweight cryptographic algorithms, and identify any potential vulnerabilities that need to be addressed to ensure their continued security.

2

Hardware Acceleration Bridging the Gap between Practical and Theoretical Cryptanalysis

Cryptanalysis is an essential part of cryptology. Not just is it useful to break ciphers for malicious applications, but it is also the basis for building secure ones. In fact almost all the ciphers still in use are trusted to be secure mainly due to the fact that many cryptanalysts are trying hard to break them publicly and failing. However, most of the time successful cryptanalytic results end up violating the cipher designers claims, but the attack itself remains theoretical due to the lack of enough resources/algorithms to efficiently implement it. For example, while the first practical SHA-1 collision was found in 2017, most of the ideas and vulnerabilities behind the attack had been discovered in 2005. The internet and IT industries didn't give much attention to the early theoretical results and it wasn't until 2016 that internet browsers starting getting rid of SHA-1. The leap from 2005 to 2017 was due to advancements in the attack algorithms, implementation techniques and hardware fabrication technologies. While hardware fabrication so far keeps on improving according to Moore's law, the other two aspects require a lot of research effort. In this survey, we touch on several examples of these efforts over the years. The survey is divided into three parts, cryptanalytic attacks

designed with specific implementation requirements, previous cryptanalytic machines and quantum computers, the technology that promises to change how we think about cryptography and cryptanalysis.

2

2.1. Introduction

While computing machines, hardware acceleration technologies and cryptography have always been closely related, the use of hardware digital circuits to accelerate cracking secure ciphers even pre-dates the invention of the Silicone transistor. The events surrounding World War II led to the emergence of Modern Cryptography, where it changed from an art exclusive to military and intelligence personnel, to a mathematical discipline practiced and studied by mathematicians and computer scientists, as well. Moreover, as it was discovered many years later, it was during the World War II at Bletchley Park in England that the first automated hardware cryptanalytic machines was built. In 1939, Alan Turing designed a mechanical machine called "Bombe", which performed statistical attacks on encrypted German messages, using Enigma [9], intercepted by the British Navy. Later, between 1943 and 1945, British code breakers built another machine called "Colossus", in order to break another German cipher, Lorenz. Unlike Bombe, which was an electro-mechanical machine, Colossus used vacuum tubes and Boolean functions and is considered as the first programmable, electronic and digital computer [10].

In this survey, the modern and potentially future hardware machines for cryptanalytical purposes are discussed. We only consider cases where a single machine is built in order to either break a cipher directly or perform a huge computational task that is considered a milestone towards breaking one. Distributed cryptanalysis projects over many locations are not considered. Besides, we consider only logical attacks that don't include assumptions on the cipher implementations. Hence, side-channel and fault-injection attacks are also not considered, regardless of their costs. In Section 2.2, a set of cryptanalytic techniques that require specific assumption when it comes to the execution platform are discussed. In Section 12.4.1, several examples of implementations or designs of cryptanalytic hardware machines are provided, showing how new and advancing technologies push the limits of what was previously considered impractical. In Section 2.4, a brief discussion on the effect of quantum computing and quantum attacks on existing ciphers is provided, while the survey is concluded in Section 10.5.

2.2. Cryptanalytic Attacks with Tight Hardware Requirements

Most of the cryptanalytic attacks on ciphers, especially symmetric key ciphers (block ciphers, stream ciphers, hash functions, etc) consist of at least one phase of executing a complex search algorithm. With the huge input and output spaces of the involved functions, this step is very expensive in terms of time and/or memory consumption. Hence, acceleration algorithms and machines usually target efficient and parallelisable implementations of this step. In [11], the authors provided three assumptions that cover a wide variety of attacks and help develop efficient accelerators:

1. Cryptanalytic algorithms are parallelisable.
2. Different nodes need to communicate with each other only for a very limited amount of time.
3. Since the target algorithms are computationally intensive, the communication with the host is very limited compared to the time spent on the computational tasks.

In this section, we describe how some of the costly attacks can be adjusted in order to satisfy these conditions and lead to efficient hardware accelerators. For a wider exploration of different cryptanalytic techniques, we refer the interested reader to any of these resources: [12–14].

2.2.1. Brute-Force Attacks

Brute-Force attacks play an important role in the security of ciphers, especially in the field of symmetric key cryptography. Brute-force attacks refer to attacks where all the possible values of a secret variable are tried until the correct value (or a set of valid values) is reached. This type of attack is applicable to any cryptosystem and provides an upper bound on the computational complexity of breaking a cipher. For example, since the Data Encryption Standard (DES) uses a secret key of 56 bits, it requires at most 2^{56} encryptions/decryptions in order to significantly narrow down the space of possible secret keys. Hence, it was believed when DES was introduced, that it has a security level of 56 bits. Any attack that requires less than 2^{56} encryptions/decryptions is considered a genuine threat to the cipher security. Moreover, while 2^{56} operations was considered beyond the realm of possibility in the 1980s and early 1990s, the NIST organization has recently announced that any security level below 112 will be considered insecure from now on [15]. However, implementing brute force attacks is not as straight-

forward as it sounds and it's practicality is not just subject to the availability of resources. A lot of challenges face the attackers when it comes to memory management, parallelisation and data sorting/searching.

2.2.2. Time-Memory-Data Trade-off Attacks

In order to overcome the high time complexity required by most attacks, there is a trade-off to be made between the time and space requirements. For example, considering a block cipher $E_K(p)$, which represents a family of bijective permutations parametrized by the key value K, the attacker can choose on one end of the trade-off to ask for $E_K(0)$ and use brute-force to try all possible keys until he finds the correct key (or set of keys, as depending on the size of p and K it may not be possible to find a unique solution using only a single encryption), or the attacker can pre-compute and sort $E_{K'}(0)\ \forall K^{'} \in 2^{|K|}$, then for any instance of the block cipher the key recovery takes $\mathcal{O}(1)$ as it involves only one memory access. However, the space complexity of the later case is $\mathcal{O}(2^{|K|})$. TMDT attacks try to find a sweet spot between these two extremes, making complex attacks more practical for implementations. In this section we describe two of the famous examples for such attacks.

Meet in the Middle Attacks The MitM attack was introduced by Diffie and Hellman in 1977 [16]. It applies to scenarios were an intermediate value during a function execution can be represented as the output of two independent random mappings. The most famous example for a cipher vulnerable to this attack is 2DES, which consists of applying the Data Encryption Standard (DES) twice using two independent keys, i.e.,

$$C = E_{K_2}(E_{K_1}(P))$$

A straightforward brute force attack would require 2^{112} operations, since DES has a key size of 56 bits. However, the MitM attack requires only 2^{57} operations and works as follows. First, we notice that

$$I = E_{K_1}(P) = E^{-1}_{K_2}(C)$$

Second, we notice that I is a collision between two random mappings. Hence, finding a pair (K_1, K_2), such that $E_{K_1}(P) = E^{-1}_{K_2}(C)$, falls under the birthday problem and requires only $2^{\frac{|K_1|+|K_2|}{2}}$, i.e., 2^{56} iterations, on average. Since, every iterations consists of one encryption and one decryption operations,

the overall number of function calls is 2^{57}, which is only twice the brute force complexity against DES. On the other hand, as most birthday attacks, the MitM requires a huge memory space, since every execution has to be stored and compared to all previous executions until the collision is found.

Hellman Time-Space Trade-off This attack, first proposed by Hellman in 1980 [17], targets accelerating brute force attacks. For simplicity, we consider only the case where the key size equals the plaintext size. However, the same attack can be generalized to other cases. The attacker chooses a plain-text P, which he knows will be encrypted at some point in the future. Then, he selects a set of possible keys as the starting point of his computation. For example, he can use the set $\{K_i | K_i \in [0, N]\}$, where N is one of the parameters of the time-space trade-off. The next step is to compute $C_i^0 = E_{K_i}(P)$. The attacker iterates over the computation $C_i^j = E_{C_i^{j-1}}(P)$, where $j \in [1, S]$ (S is the second parameter of the trade-off). All the previous step are done offline, without communicating with the target user. The attacker at this point ends up with N chains, each has $S+1$ nodes, as follows:

$$\begin{aligned} C_0^0 \to C_0^1 \to C_0^2 \cdots \to C_0^S \\ C_1^0 \to C_1^1 \to C_1^2 \cdots \to C_1^S \\ \vdots \\ C_N^0 \to C_N^1 \to C_N^2 \cdots \to C_N^S \end{aligned}$$

The previous lists include $(S+1)*(N+1)$ Keys. In order to save memory, the attacker stores only the initial key K_i and the final value C_i^S. Next, in the online phase, the attacker intercepts a ciphertext $C = E_{K^*}(P)$. First, he compares C to the $N+1$ final values in his pre-computed table. If $C = C_i^S$, since $C = E_{K^*}(P)$, then $E_{K^*}(P) = E_{C_i^{S-1}}(P)$. The attacker returns $K^* = C_i^{S-1}$. Otherwise, the attacker sets $C^0 = C$ and computes the list in the same manner as the lists generated during the offline phase:

$$C^0 \to C^1 \to C^2 \cdots \to C^S$$

If $C^l = C_i^S$, then $E_{K^*}(P) = E_{C_i^{S-l-1}}(P)$. In both cases, the attacker needs to re-compute the chain where the collision occurred until the key value is found. On average, the online phase requires $S/2$ operations to find the collision and $S/2$ operations to find the key. However, since the attack covers only

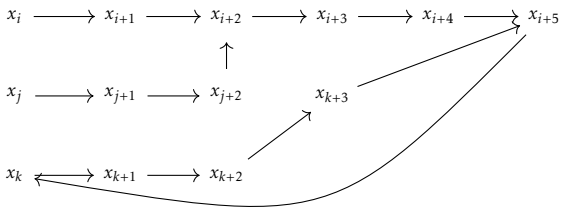


Figure 2.1: A simplified functional graph example. An edge goes from vertex a to vertex $H^{'}(a)$

$(S+1)*(N+1)$ key candidates, it has a success probability of $\frac{(S+1)*(N+1)}{|\mathcal{KS}|}$, where $|\mathcal{KS}|$ is the size of the full key space.

2.2.3. Parallel Birthday Search Algorithms

So far, we have described three different generic attacks against ciphers. All these attacks have one feature in common. A cipher is considered as a random mapping $C = E_{K^*}(P) : \{0,1\}^{|P|} \times \{0,1\}^{|K|} \rightarrow \{0,1\}^{|C|}$. In this section, we consider a wider class of functions; collision-resistant compression functions $T = H(x) : \{0,1\}^n \rightarrow \{0,1\}^t$, such that $n \gg t$ and it is computationally hard to find a pair (x_1, x_2) such that $H(x_1) = H(x_2)$. This class of functions has several applications in the construction of secure hash functions and the cryptanalysis of symmetric key ciphers. For examples, the problem of finding such a collision is helpful to the meet in the middle attack described earlier. It is known that the computational complexity of finding a collision for such a function is upper bounded by the birthday bound $2^{t/2}$. However, the efficient design of a collision search algorithm is not a trivial task, specially if the attacker wants to make use of parallelisation over a set of computing machines. This issue is discussed in details in [18]. First, we look at the problem of designing an efficient algorithm for the birthday search problem on a single processing unit. A straightforward approach is to compute $2^{t/2}$ random instances. With high probability, a colliding pair exists in the list formed by these instances. However, such approach requires $\mathcal{O}(2^{t/2})$ memory locations and $\mathcal{O}(2^t)$ memory accesses/comparisons. In order to overcome these tight requirements, several attacks with different trade-offs have been proposed, almost all of them share the same property; the function in question is reduced to $T = H^{'}(x) : \{0,1\}^t \rightarrow \{0,1\}^t$, which is treated as a pseudo-random function (PRF). One of the useful ways to represent such a function is using a functional graph, which is a directed graph with 2^t vertices and two vertices x and y with an edge from x to y are connected if $y = H(x)$, as shown in Figure 2.1.

The collision search problem can be treated as a graph search problem, where he attacker is looking for two edges with the same endpoint but with different start-points. Pollard's rho method [19] helps find a collision in the functional graph with a small memory requirement. The underlying idea is to start at any vertex and perform a random walk in the graph until a cycle is found. Unless the attacker is unlucky to have chosen a starting point that is part of the cycle, he ends up with a graph that resembles the Greek letter ρ and the collision is detected.

Nonetheless, Pollard's rho method cannot be efficiently parallelized without modifications. If an attacker tries to run many instances of the algorithms on several machines, independently, each machine will try to look for a cycle in a specific part of the functional graph. However, there is no guarantee that the first colliding pair will be found using a single machine, as each member of the pair can be found using a different machine. For example, two machines can enter the same cycle, but the attacker will not detect this event. Hence, if he uses m machines, he will need $\mathcal{O}(\frac{2^t}{\sqrt{m}})$ time to find the collision, as opposed to his original target of $\mathcal{O}(\frac{2^t}{m})$.

In [18], the authors proposed a method to achieve $\mathcal{O}(\frac{2^t}{m})$ speed-up, using limited memory and communication requirements. First, the attacker defines a distinguished point to be a point $H^{'}(x)$ that has a special property which can be easily checked, e.g. the first d bits are equal to 0. Second, the attacker chooses m random messages $x_0^1, x_0^2, \cdots, x_0^m$ and assigns one of them to each machine. Third, each machine i computes a trace $(x_0^i, x_1^i, \cdots, x_d^i)$, where $x_j^i = H^{'}(x_{j-1}^i)$ and x_d^i is a distinguished point. This is a random walk in the functional graph. If the probability of the condition $x_d^i = H^{'}(x_{d-1}^i)$ is θ, the average length of the trace d is $\frac{1}{\theta}$. Hence, if θ is too large, the traces are too short and the total number of traces $\approx \frac{2^k}{\theta}$ is in the same order of magnitude of 2^k. This means that the attacker does not observe a significant reduction in the memory usage compared to a random-search based algorithm. On the other hand, if θ is too small, the traces become so long, and there is a risk of hitting a cycle within the trace, without ever hitting a distinguished point. Hence the choice of θ is crucial for the attack efficiency. Moreover, it is a good practice to abort the trace after it becomes too long, e.g. $\frac{20}{\theta}$. The previous two steps are repeated $\frac{2^{t/2}\theta}{m}$ times. Each trace is specified in the memory as (x_0^i, d, x_d^i). Finally, a central server needs to sort these traces efficiently, according to the values x_d^i and find the traces that have the same endpoint. Once two similar traces are found, it is easy to find a collision within them that looks like in Figure 2.2.

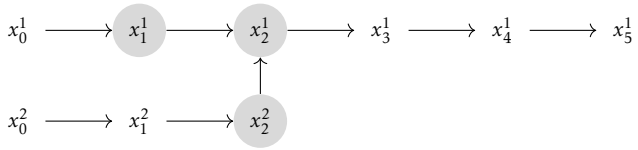


Figure 2.2: An example of two colliding traces in the functional graph

2.3. Hardware Machines for Breaking Ciphers

2.3.1. Brute Force Machines

In this section we describe cases where engineers have been able to build machines to efficiently execute brute force attacks against certain ciphers.

Deep Crack In 1998, the Electronic Frontier Foundation (EFF) built a dedicated hardware machine consisting of 1856 ASIC chips connected to a single PC. The machine was able to test over 90 billions DES keys per second, which means that it can go over all the 2^{56} DES keys in 9 days. It was able to solve one of the RSA security DES challenges in 56 hours. The project costed 250,0000 US Dollars and was motivated by the discrepancy between the estimates of academia and government officials regarding the cost and time required to break DES [20].

COPACOBANA In CHES 2006, COPACOBANA [11] was introduced as an FPGA cluster architecture consisting on 120 FPGAs controlled by a host computer. It is considered to be the first publicly reported configurable platform built specifically for cryptanalysis as its main purpose. The design philosophy behind the architecture depends on the three main assumptions in Section 2.2. These assumptions are satisfied by both brute force and cryptanalytic attacks. Hence, the COPACABANA has been used to accelerate several attacks [21]. We sum up some of these attacks in Table 2.1. Some of the attacks performed on the COPACOBANA platform, e.g. guess and determine attack on A5/1, were specially designed in the first place to make use of hardware acceleration [22].

WindsorGreen In 2016, a document was release accidentally on the New York University server, describing a custom made supercomputer designed by the NSA and IBM, which is believed to have mainly two applications, cracking ciphers and forging cryptographic signatures [23]. However, limited information is available publicly on the project.

Table 2.1: Some of the attacks performed using the COPACOBANA platform

Cipher	Attack Type	Time Consumed
DES	Brute Force	6.4 days
A5/1	Guess and Determine [22]	6 hours
ECC (k=79)	Discrete-Log	3.06 hours
ECC (k=97)	Discrete-Log	93.4 days

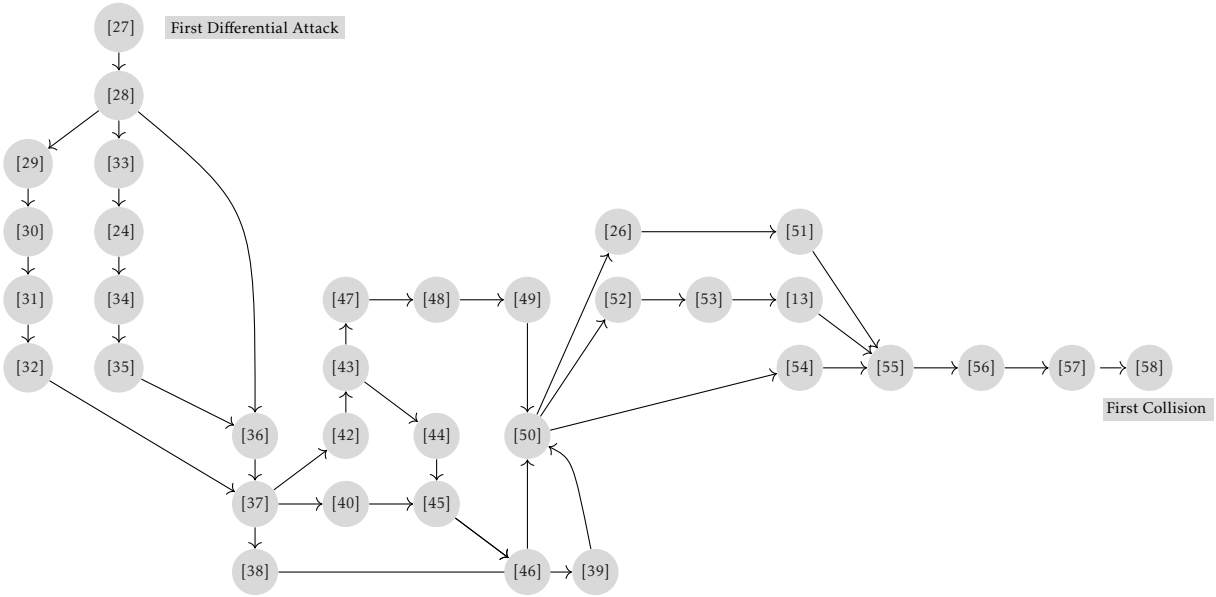


Figure 2.3: Summary of the improvement and efforts towards accelerating the collision attacks against SHA-1 between 2005 and 2017

2.3.2. Acceleration of Collision Attacks on Hash Functions

In 2005, the first theoretical collision attack on SHA-1 was published by Wang et al [24]. Since then, a lot of efforts have been targeted towards making the attack more efficient. These efforts are summarized in Figure 2.3. In 2015, the authors of [25] provided an estimation for finding near collisions on SHA-1, which is a critical step in the collision attacks. The authors provided a design of an Application-Specific Instruction-set Processor (ASIP), named Cracken, which executes specific parts of the attack. It was estimated that to execute the free-start collision and real collision attacks from [26], the attacks will take 46 and 65 days and cost 15 and 121 million Euros respectively.

In 2016, the first attack was practically executed using a cluster of 64 GPUs

and took 10 days [56]. Later in 2017, the first real SHA-1 collision was computed [58], using a combination of CPUs and GPUs, taking 6500 CPU years and 100 GPU years. However, the exact details of the machine used were not revealed in the paper.

2.3.3. The Factoring Machine

The problem of efficiently finding the prime factors of an integer is one of the oldest mathematical problems in the field of Number Theory. It is also the basis of some of the Public Key Cryptosystems (PKC), such as RSA. If a computer can factor $n = pq$ into p and q, it would lead to breaking RSA systems of key size $|n|$. The Number Field Sieve (NFS) algorithm is one of the famous algorithms for solving the factoring problem. However, efficient and cheap implementations of this algorithm are non-trivial. In [59], the author describes three different architectures for machine to perform the NFS algorithm, the cheapest of which costs 400,000 US Dollars and is estimated to take under one year to break RSA-768 in under one year. However, the results are highly speculative and are not supported by any actual implementation.

2.3.4. Molecular Computers

In [60], Boneh et al. describe an attack on DES that is estimated to take one day to recover the key. It is based on a theorized underlying DNA computer and can be extended to any cipher of key size ≤ 64 bits. However, The attack has not been implemented in real life and remains a theoretical idea, until the required DNA computer becomes available.

2.3.5. Blockchain Mining

While the topic of blockchain mining is not directly related to cryptanalysis or breaking ciphers (specifically hash functions), it is closely related to the acceleration of brute force attacks. The mining operation involves finding an input block to a secure hash function such that the output tag is less than a specific value, i.e. has a certain number of leading 0's. The number of required leading 0's defines the complexity of the problem, which is equivalent to a pre-image attack against a truncated version of the hash function. If that version of the hash function is pre-image resistant, then the mining step is equivalent to a brute force pre-image attack. As the blockchain gets older, the mining step gets more complex. Hence, several industrial players have been interested into accelerating these computations. In 2016, Intel applied for a patent for a Bitcoin mining hardware accelerator [61], which consists

of a processor and a coupled hardware accelerator that uses SHA-256 as the main underlying hash function. It is claimed that this system can reduce the power consumption involved in Bitcoin mining by 35%. In April 2018, Samsung has also confirmed that it is building ASIC chips to mine Bitcoin. The new chips utilize the technology and expertise of Samsung's high memory capacity GPUs and can be designed for a specific hash function, to give customers the freedom to choose the target blockchain, not being limited to Bitcoin. However, once fabricated the chip is hash-function specific. It is supposed to increase the power efficiency by 30% and to execute 16 Tera hashes per second [62].

2.4. Quantum Computers

In recent years, there has been a substantial amount of research on quantum computers. If large-scale quantum computers are ever built, they will be able to break many of the public-key cryptosystems currently in use.

The basis of this problem is that the hidden subgroup problem (HSP) is solvable in finite Abelian groups in quantum polynomial time. Thus, factorization, discrete logarithm, discrete logarithm in elliptic curves are solvable in quantum polynomial time as well [63].

The question of when a large-scale quantum computer will be built is a not obvious. While in the past it was less clear that large quantum computers are a physical possibility, many scientists now believe it to be merely a significant engineering challenge. It has taken almost two decade to deploy the currently used public key cryptography standards. That's why, to anticipate, the NIST recently published a call for the post-quantum cryptography standard.

One of the challenge to solve is to find mechanisms for controlling and manipulating the quantum bit easier. Currently a laser is used to physically move each individual qbit stored in the form of ion from one location to another. Breaking cryptograhic standards used today would require to move million of ions at the same time, so millions of lasers, making it impractical for current technologies. But recent advances showed that with a new technic from the university of Sussex called blueprint [64] allows to manipulate thousands of qubits with currently available technoligies. With this technology, a large scale quantum computer consisting of millions of ions would occupy a space the size of a football field, costing upwards of $120 million .

2.5. Conclusion

Breaking a real world cipher in practice is a scientific and technological challenge. When both advances in science and technology gives signs that a cipher can be broken in practice, new cryptographic standards are pulled from the current knowledge. This should be done way before the attack is made practical to absorb the inertia needed to deploy a new cryptographic standard. From a theoretical attack and a practical one, there is still a gap, where new discoveries can be made, that can push further the knowledge that we have to make even better cryptographic standards. When practical attacks are not well anticipated, this inertia can make possible, practical attacks on cryptographic standards that are still in use, which can be catastrophic.

This study is funded by Temasek Laboratories @ NTU.

3

On The Cost of ASIC Hardware Crackers: A SHA-1 Case Study

In February 2017, the SHA-1 *hashing algorithm was practically broken using an identical-prefix collision attack implemented on a GPU cluster, and in January 2020 a chosen-prefix collision was first computed with practical implications on various security protocols. These advances opened the door for several research questions, such as the minimal cost to perform these attacks in practice. In particular, one may wonder what is the best technology for software/hardware cryptanalysis of such primitives. In this paper, we address some of these questions by studying the challenges and costs of building an ASIC cluster for performing attacks against a hash function. Our study takes into account different scenarios and includes two cryptanalytic strategies that can be used to find such collisions: a classical generic birthday search, and a state-of-the-art differential attack using neutral bits for* SHA-1.

We show that for generic attacks, GPU and ASIC poses a serious practical threat to primitives with security level ~ 64 *bits, with rented GPU a good solution for a one-off attack, and ASICs more efficient if the attack has to be run a few times. ASICs also pose a non-negligible security risk for primitives with* 80*-bit security. For differential attacks, GPUs (purchased or rented) are often a very cost-effective choice, but ASIC provides an alternative for organizations that can afford the initial cost and look for a compact, energy-efficient, reusable solution. In the case*

of SHA-1, *we show that an ASIC cluster costing a few millions would be able to generate chosen-prefix collisions in a day or even in a minute. This extends the attack surface to TLS and SSH, for which the chosen-prefix collision would need to be generated very quickly.*

3.1. Introduction

Hardware cryptanalysis has always been an important part of modern cryptography. It studies building application-specific electronic machines for performing cryptanalytic attacks. These machines can use different technologies, starting from mechanical computers during World War II, to FPGA, GPU or ASIC in the modern days. A full discussion of the history and state of the art of this field can be found in [65]. A widely held belief is that FPGAs and GPUs are suited for small-scale or low-budget computations, while ASIC is predicted to be better for heavy computational tasks or if the attacker has an important budget to spend. It is intuitive that a chip that is designed for a specific task is much more efficient than a general-purpose chip for the same task. However, since ASIC design has a huge non-recurring cost for fabrication, it is only competitive when a huge amount of chips is required. Besides, unlike the cryptographic algorithms themselves, which are usually optimized for hardware implementations, the cryptanalytic algorithms are usually designed for general-purpose computing machines. Hence, it is not necessarily true that ASIC implementations of such algorithms are more efficient. In other words, ASIC can always be at least as efficient as general-purpose CPUs or GPUS, as in the worst case the ASIC designer can simply design a circuit that is similar to the general-purpose one, but the gap in efficiency between the ASIC and the general-purpose circuit depends on the algorithm being implemented.

In general, ASIC provides an unfair advantage to players with bigger budgets. This has led to speculation that large intelligence entities may already possess ASIC hardware crackers that can break some of the widely used cryptographic schemes. In this paper, we address the question of the feasibility of such machines and whether it is more beneficial to use ASIC for cryptanalysis. The answer to this question is yes, but only for generic attacks of very large complexities, e.g. $> 2^{64}$. For low scale or more complicated cryptanalytic attacks, GPUs provide a very competitive option, due to re-usability, mass production and/or the possibility of renting them.

A relevant topic to our study is blockchain mining. As discussed earlier, big players can gain a huge advantage by using expensive ASICs. This has been a trend for Bitcoin specifically, where the introduction of a new ASIC

machine lowers the profitability of older machines significantly. To maintain fairness of blockchain and cryptocurrency mining, memory-bound and ASIC-resistant hashing algorithms have been used, such as Ethash [66] for the Ethereum cryptocurrency and the X16R algorithm [67].

Related Work COPACOBANA [11] was introduced in CHES 2006 as an FPGA cluster consisting on 120 FPGAs. It is considered to be the first publicly reported configurable platform built specifically for cryptanalysis. The design philosophy behind the architecture depends on three assumptions:

1. Cryptanalytic algorithms are parallelisable.
2. Different nodes need to communicate with each other only for a very limited amount of time.
3. Since the target algorithms are computationally intensive, the communication with the host is very limited compared to the time spent on the computational tasks.

These assumptions are satisfied by both brute force (generic) and a lot of cryptanalytic attacks. Hence, the COPACOBANA has been used to accelerate several attacks [21]. In our study we follow the same assumptions and add one more assumption:

4. Each node requires a constant/low amount of storage. The overall attack can be implemented using an almost memory-less algorithm.

This assumption needs to be satisfied by the attack algorithm in order to make sure that the efficiency due to parallelisation is not lost due to memory operations. For example, a naive approach to implementing a generic birthday collision search on m nodes, can lead to only $\sqrt{m}$ speed up compared to a single node if the algorithm doesn't satisfy this assumption.

Our Contributions. This paper is an attempt at answering three important research questions:

– *Can the cost of the collision attacks against* SHA-1 *be reduced?* There has been major breakthroughs in the cryptanalysis of SHA-1 over the past few years, with the first practical identical-prefix collision (IPC) found in February 2017 [58] and the first chosen-prefix collision (CPC) found in January 2020 [68]. While these attacks are practical on general-purpose GPUs, they still take a few months to generate one collision, by both academic and industrial entities. Interestingly, the authors of [68] remarked that TLS and SSH connections using SHA-1 signatures to au-

thenticate the handshake could be attacked with the SLOTH attack [69] if the chosen-prefix collision can be generated quickly. Hence, we would like to check if ASIC can provide a better alternative to speed up the attacks, using larger budgets. We actually show that chosen-prefix collisions could be generated within a day or even a minute using an ASIC cluster costing a few dozen Million USD (the amortized cost per chosen-prefix collision is then much lower).

– *What is the difference between generic attacks and cryptanalytic attacks in terms of cost and implementation?* When analyzing a new cipher, any algorithm that has a theoretical time complexity lower than the generic attacks is considered a successful attack and the cipher is considered broken. For example, an n-bit hash function that is collision resistant up to the birthday bound is considered insecure if there is a cryptanalytic attack that requires less than $2^{0.9n/2}$ hash calls. Most of the time, researchers only measure time complexity in terms of function calls and ignore other operations required to perform the attack if they are much smaller. However, in practice, it can be a lot harder to implement a cryptanalytic attack compared to a generic attack, even with lower theoretical complexity. There are countless attacks published every year with a complexity very close to the generic one, but a natural example of such scenarios is the biclique attack against AES [70], where the brute force complexity is reduced only by a small factor from 2^{128} to $2^{126.1}$. However, one can question if implementing the simple brute force attack would actually be much less complex in practice. In this paper, we compare the generic 64-bit birthday CPC attack over a 128-bit hash function to the cryptanalytic CPC attack against SHA-1 (which costs close to $2^{63.6}$ operations on GPUs, and of a lower complexity in theory) showing that in practice, the generic attack cost is more than 5 times cheaper than the ad-hoc CPC attack. Attacks like biclique or complex cryptanalysis are even more difficult to implement than the ad-hoc CPC attack and might require a huge memory, which probably makes the gap even larger. Hence, we argue that for a cryptanalytic attack to be competitive against a generic algorithm in practice, one must ensure a sufficiently large gap, at least of a factor 5, if not more (only an actual hardware implementation testing or estimation could give accurate bounds on that factor).

– *How secure is an 80-bit collision-resistant hash function?* In the NIST Lightweight Cryptography Workshop 2019, Tom Brostöm proposed an application for lightweight cryptography where the SIMON cipher [71] is used in the Davis-Meyer construction as a secure compression function

which is collision-resistant at most up to 2^{64} computations [72]. Besides, it remains a common belief that SHA-1 is insecure due to the cryptanalytic attacks against it, but it would have still been acceptable otherwise. Actually, it is only since 2011 that 80-bit security is not recommended anymore by the NIST, and 80-bit security for data already encrypted with this level of protection is deemed acceptable as a legacy feature, accepting some inherent risk. Hence, we study the cost of implementing the generic 2^{80} birthday collision attack against SHA-1, showing that it is within our reach in the near future, costing ≈ 61 million USD to implement the attack in 1 month, which is not out of reach of large budget players, *e.g.* large government entities, and with the decreasing cost of ASICs, this will even be within reach of academic/industrial entities in the near future.

Finally, we argue that ASIC provides the most efficient technology for implementing high complexity and generic attacks, while GPU provides a competitive option for cryptanalytic and medium/low cost attacks.

3.2. Hash Functions and Cryptanalysis

Cryptographic hash functions are one of the main and most widely used primitives in symmetric key cryptography. One of their key applications is to provide data integrity by ensuring each message will lead to a seemingly random digital *fingerprint*. They are also used as building blocks of some digital signature and authentication schemes. A cryptographic hash function takes a message of arbitrary length as input and returns a fixed-size string, which is called the hash value/tag. In order for the function to be considered secure, it must be hard to find collisions, i.e. two or more different messages that have the same tag. More specifically, a n-bit cryptographically secure hash function must satisfy at least the security notion of collision resistance, *i.e.* finding a pair (M_1, M_2) of distinct messages, such that $H(M_1) = H(M_2)$ must require about $2^{n/2}$ computations.

3.2.1. SHA-1 and Related Attacks.

The SHA-1 hash function defines a generalized-Feistel-based compression function used inside the Merkle-Damgård (MD) algorithm. It was selected in 1995 as a replacement for the SHA-0 hash function after some weaknesses have been discovered in the latter. While the two functions are relatively similar, SHA-1 was considered collision resistant till 2005, when Wang *et al.* proposed the first cryptanalytic attack on SHA-1 [33]. Since then, a lot of efforts

have been targeted towards making the attack more efficient. In 2015, the authors of [25] provided an estimation for finding near collisions on SHA-1, which is a critical step in the collision attacks. The authors provided a design of an Application-Specific Instruction-set Processor (ASIP), named Cracken, which executes specific parts of the attack. It was estimated that to execute the free-start collision and real collision attacks from [26], the attacks will take 46 and 65 days and cost 15 and 121 Million Euros respectively. At Eurocrypt 2019, Leurent and Peyrin [73] provided a chosen-prefix attacks which uses two parts: first a birthday search to reach an acceptable set of differences in the chaining variable, and then a differential cryptanalysis part that successively generate near-collision blocks to eventually reach the final collision. The attack was implemented on GPUs and a first chosen-prefix collision was published in January 2020 [68].

3.2.2. Birthday Search in Practice.

The efficient design of a collision search algorithm is not a trivial task, especially if the attacker wants to use parallelization over a set of computing machines. This issue is discussed in details in [18]. The collision search problem can be treated as a graph search problem, where the attacker is looking for two edges with the same endpoint but with different starting points. Pollard's rho method [19] helps finding a collision in the functional graph with a small memory requirement. The underlying idea is to start at any vertex and perform a random walk in the graph until a cycle is found. Unless the attacker is unlucky to have chosen a starting point that is part of the cycle, he ends up with a graph that resembles the Greek letter ρ and the collision is detected. Unfortunately, this method is not efficiently parallelizable, as it provides only $\mathcal{O}(\sqrt{m})$ speed-up when m cores are used. In [18], the authors proposed a method to achieve $\mathcal{O}(m)$ speed-up, using limited memory and communication requirements. This algorithm leads to very efficient parallel implementations, and is the basis for our study.

However, in the chosen-prefix collision attack against SHA-1, it is not applied directly to the compression function of SHA-1, but to a helper function. Let IV_i represent a chaining value to the compression function (reached after processing a prefix), x a message block, and $H(IV_i, x)$ the application of the SHA-1 compression function. The goal of the birthday phase of CPC attack is to find many solutions x_1 and x_2 such that $L(H(IV_1, x_1)) = L(H(IV_2, x_2))$, where $L(x)$ is a linear function applied to a word x, in order to select some of the output bits of the compression function. The helper function is defined

as:

$$f(x) = \begin{cases} L(H(IV_1, x)), & \text{if } x = 1 \pmod 2 \\ L(H(IV_2, x)), & \text{otherwise.} \end{cases} \tag{3.1}$$

When a collision $f(x_1) = f(x_2)$ is found, we have $x_1 \neq x_2 \pmod 2$ with probability one half, and in this case we obtain $L(H(IV_1, x_1)) = L(H(IV_2, x_2))$.

3.2.3. Differential Cryptanalysis.

In this section we briefly describe the algorithms involved in the second part of the chosen-prefix collision attack: the generation of successive near-collision blocks to reach the final collision. The details of this differential attack can be found in [26, 33, 55, 56, 58, 68, 73]. For each new near-collision block, the attacker has to go through three main steps:

1. Preparing a fully defined differential path for the SHA-1 compression function (in particular a non-linear part has to be generated for the first few steps of the SHA-1 compression function)
2. Find base solutions for the first few steps of this differential path (a base solution is simply two messages inputs that verify the planned differential path in the internal state up to the starting step of the neutral bits).
3. Expand those solutions into many solutions using what is known as neutral bits (in order to amortize the cost of the base solution), and check whether any of these solutions verify the differential path until the output of the compression function.

A neutral bit for a step i is a bit (or a combination of bits) of the message such that when its value is flipped on a base solution valid until step i, the differential path is still satisfied with high probability until step i. Most of the time, a neutral bit is a single bit, but it can sometimes be composed of a combination of bits. A neutral bit for a step i allows to amortize the cost of finding a solution to the differential path until step i.

The hardware cluster we consider consists of one master node and many slave nodes. The master builds a proper differential path for the compression function steps, based on the incoming chaining values, and generates base solutions based on this path. The slave is then required to expand these base solutions into a wider set of potential solutions and find out which of them satisfy the differential path until a certain step r (we selected $r = 40$ for ASIC for implementation efficiency purposes, but we remark that $r = 61$ was selected for GPU even though it does not have much impact) in the SHA-1

compression function. The master then aggregates all the solutions that are valid up to step r and exhaustively search for solutions that are valid up to step 80. This is repeated several times until a valid solution for the differential path is found. Consequently, we define a slave as a dedicated core that is responsible for extending a base solution found by the master into a set of potential solutions by traversing the tree of solutions defined by the neutral bits.

Unfortunately, this attack is not hardware-friendly and needs a lot of control logic. The master has to send to the slave:

1. A base solution, which consists of two message blocks M_1 and M_2.
2. A set $[DP]$ of differential specifications for the slave to check conformance.
3. A group of neutral bit sets N_i, where the neutral bits in N_i are supposed to be neutral up to step i.

Combining a base solution (M_1, M_2) valid at step i and the set N_i, we get about $2^{|N_i|}$ new solutions that are valid up to step i, simply by trying all the possible combinations of the neutral bits in the set. In a naive approach, each of these partial solutions is expended to $2^{|N_{i+1}|}$ by applying combinations of the next set. Eventually, we would end up with $2^{\sum_i N_i}$ partial solutions, organised in a tree as shown in Figure 3.1. However, the neutral bits N_{i+1} are defined such that they don't impact the path up to step $i+1$. Therefore, if the partial solution does *not* satisfy the conditions at step $i+1$, there is no need to apply the neutral bits N_{i+1}, and we can instead cut the corresponding branch from the tree. Indeed, there is a certain probability that a solution valid at step i will be valid at step $i+1$, according to the SHA-1 differential path selected. With the parameters used in SHA-1 collision attacks, most subtrees fail.

We can generate the partial solutions using a graph search algorithm to start navigating the tree from its root, and neglect complete subtrees that are failing. In this paper we choose Depth-First Search (DFS) graph search, with some modifications to suit our specific problems, in order to satisfy our assumptions for the cryptanalytic algorithm, as DFS has low memory requirements.

Our attack scenarios. In this paper we consider three attack scenarios:

1. A plain 2^{64} birthday search: a generic birthday attack against a 128-bit hash function, constructed by selecting only 128 bits out of the 160

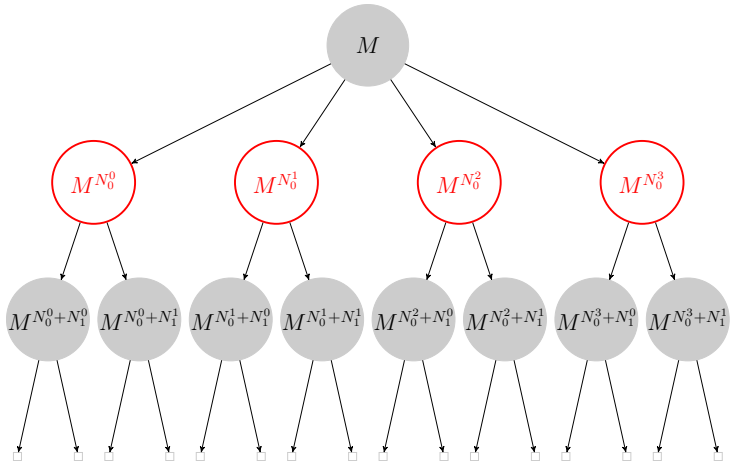


Figure 3.1: Building partial solutions with neutral bits

output bits of the SHA-1 compression function.

2. A plain 2^{80} birthday search: a generic birthday search over the full space of the SHA-1 compression function.

3. The chosen-prefix collision attack on SHA-1 from Leurent and Peyrin [68, 73].

These three scenarios cover two generic attacks against two security levels used in practice and one cryptanalytic attack.

3.3. Hardware Birthday Cluster

In this section, we describe the hardware core that handles the birthday attack. First, we define the nodes used in the proposed cluster. Then, we describe the design of the slave nodes and the communication requirements.

3.3.1. Cluster Nodes

The cluster used to apply the parallel birthday search attack consists of two types of nodes:

1. *Master*: a software-based CPU that manages the attack from high level and performs some jobs including choosing the initial prefixes, distributing the attack loads among slaves, sorting of the outputs and identifying colliding traces.

2. *Birthday Slaves*: dedicated cores that can perform different parts of the parallel birthday search. Specifically, it compute traces in the functional graph of the function in question, and once the master has identified colliding traces, the core also can locate the exact collision in these traces.

3.3.2. Hardware Design of Birthday Slaves

The design of the proposed birthday slave is shown in Figure 3.2. It's main role is to iterate the helper function of Equation 3.1. It consists of a reconfigurable ROM, where the initial trace value x_0, IV_1 and IV_2 are loaded, a logic SHA-1 core which performs the step function of SHA-1, a comparator to compute $L(x)$, $x \pmod 2$ and check whether a given x is a distinguished point (see [18]) or not, a memory to store distinguished points and a control unit to handle the communications with the master, and measure the lengths of different traces.

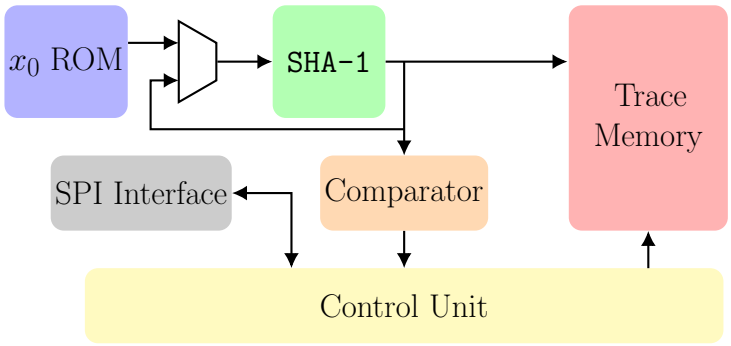


Figure 3.2: Birthday slave for the parallel collision algorithm

In order to estimate the cost of the proposed core, the area and speed are compared to a single, step-based SHA-1 core, which is a standard practice in estimating the cost of SHA-1 cryptanalytic attacks. We have implemented a full SHA-1 core and it has an area of 6.2 KGE and 0.21 ns critical path. The implementation of the core in Figure 3.2 using a step-based SHA-1 core requires at least twice this area. Moreover, its critical path is dominated by the memory and counters in the control logic. Besides, it is not expected that a huge ASIC cluster will run at a speed higher than 1 GHz, due to the power consumption. Hence, in order to regain the efficiency lost due to the extra control logic and memories, it is a good approach to try to use this logic with as many SHA-1 steps as possible. Given these experiments and the huge cost of the control overhead, we increase the efficiency by cascading 4 SHA-1 steps instead of one in the SHA-1 core. This makes the critical path around

1ns, but a full SHA-1 computations takes only 20 cycles instead of 80, and the overhead 25% instead of 100%.

3.4. Verification

We have verified the attack by finding collisions on a small number of bits using functional simulation of the hardware implementations. Specifically, we found collisions on $20 \sim 330$ bits of the output. We have also generated traces for larger number of bits and compared them to traces generated using software implementations.

3.5. Hardware Differential Attack Cluster Design

In this section we discuss the challenges and different trade-offs when implementing the neutral bit search algorithm in ASIC and give a description of the circuit. The cluster architecture uses 3 types of nodes: master nodes, birthday slaves (BC), and neutral bit slaves (NB).

3.5.1. Neutral Bits

One of the trade-offs when implementing the attack is whether to consider neutral bits as only single-bits or to use the more general sets of multi-bits. The first approach leads to a very small circuit, but it strongly limits the number of usable neutral bits. This increases the overall work load, as more base solutions need to be generated, and more time is spend applying neutral bits. The second approach is more complex, because multi-bit neutral bits must be represented by a bit-vector. However, the single-bit neutral bits are not sufficient to implement an efficient attack, and we have to use the second option:

1. Our simulation results show that the success probability of single-bits is very low. Hence, any gain achieved by using them is offset due to the huge work load and high communication cost between the master and slave.

2. In order to achieve significant results, multi-bits are inevitable. In particular, *boomerangs* [74] (which can be seen as multi-bit neutral bits with extra conditions to reach a later step) are crucial cryptanalytic tools for a low-complexity attack against SHA-1. Hence, avoiding multi-bits can lead to a drastic loss in terms of attack efficiency.

3.5.2. Storage

Each multi-bit neutral bit is represented by a 512-bit vector, which indicates the location of the involved bits in the message block (a SHA-1 message block is indeed 512-bit long). However, we noticed that almost all the neutral bits involve bits only in the last 6 32-bit words of the message block. Therefore, we reduced the representation to only 192 bits. Yet, since the original chosen-prefix collision attack against SHA-1 uses ~ 60 neutral bits, including boomerangs, this requires a representation of $\sim 11,520$ bits. Besides, the last few levels of the tree requires 320 bits per neutral bits as the boomerangs can be located as early as step 6. In addition, for each level of the tree search we need a counter to trace which node we are testing. The tree used in the attack has ~ 10 levels, and our experiments show that the maximum number of neutral bits in one level is ~ 26 bits. Hence, the overall size of the counters is ~ 260 bits. In order to design the circuit that handles this tree search algorithm, we tried out four different approaches:

1. Generic approach: we assume that each tree level can have ~ 28 neutral bits (slightly higher than our experiments for tolerance). Also, assume that these levels can be related to any step of the SHA-1 compression function between 10 and 26, i.e. 16 possible steps. In total, this requires $\sim 63,670$ memory locations (Flip-Flops).

2. Statistical approach: from the software experiments and simulations, we identified an average number of neutral bits per level. In the design, we use the maximum number of neutral bits we observe for each level (in addition to two extra bits for tolerance). We observed that only the first few levels require such a huge storage, while the later levels usually have $3 \sim 7$ bits per level. In addition, boomerangs are usually $3 \sim 4$ per level. This reduces the memory requirement by about 50%. However, it remains a huge requirement.

3. Configurable approach: our experiments showed that not only the number of neutral bits per level can be predicted, but also the values of these bits. In other words, very few bits have different values for different blocks. Hence, we can fix each neutral bit to two or three choices and use flip-flops to configure which choice is selected during execution. This reduces the cost significantly. However, the cost is still high as a multiplexer has an area only $\sim 50\%$ of a flip-flop. Besides, we still need flip-flops for configuring these multiplexers.

4. Another approach is to reduce the cost by fixing the the neutral bit values to a set of statistically dominant values. Indeed, [68] reports using the same neutral bits for each near-collision bloc. This eliminates the

need to store the neutral-bit reference values.

At the end, we chose the third approach, since our analysis shows that it captures the reality, while allowing some level of freedom for the attacker to adjust the attack parameters after fabrication.

3.5.3. Architecture

Figure 3.3 shows the architecture of the neutral-bit slave. It consists of a register file to store the differential path for comparison, a configurable ROM to store the base solution, a unit to enumerate the different neutral bit patterns and maintains the tree level for the graph search algorithm, and the SHA-1 step logic.

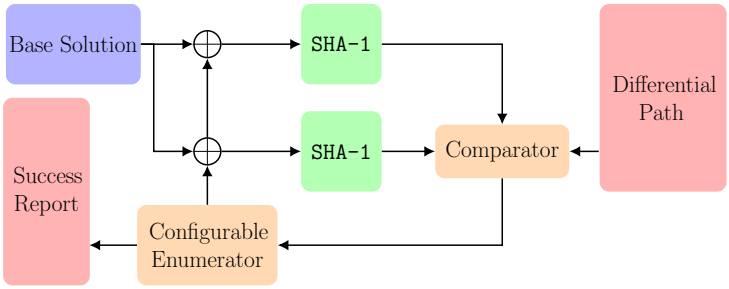


Figure 3.3: Neutral-bit slave hardware architecture

3.6. Chip Design

In this section, we describe our process for simulating the proposed chips and the results in terms of power, area and performance for each.

3.6.1. Chip Architecture

A challenge when designing this cluster is the communication overhead between the master and the slaves. A 100MHz SPI bus interface is used as a one-to-one communication interface with the attack server. A set of ASICs can also be daisy-chained, thanks to this interface, in such a way to lower the number of interconnects with the master. It provides enough bandwidth to handle the data exchanges between the BD/NB slave cluster and the attack server. The CU (Control Unit) is responsible for dispatching the 32-bit deserialized packets sent by the attack sever to configure the BD/NB slaves. It is also responsible for daisy-chaining and demultiplexing the output traces

of the different BD/NB slaves to the SPI bus interface before the serialization. Each ASIC also outputs an asynchronous interrupt signal. The interrupt signal is 1 when at least one BD/NB slave is done, and an output trace is available. Those interrupt signals are managed by a set of ZYNQ board cluster interfaces.

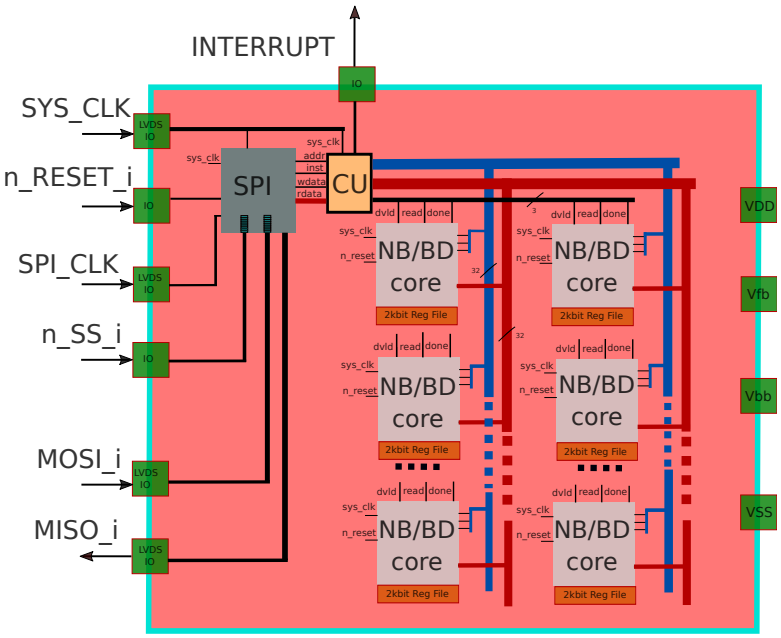


Figure 3.4: System architecture of the ASIC cluster chip

3.6.2. ASIC Fabrication and Running Cost

Estimating the cost of fabricating and running an ASIC cluster can be challenging as many parameters are confidential to the fabs. In order to estimate the costs of the attacks considered, we developed a methodology based on the information available publicly. We considered the FD-SOI 28nm technology from ST-Microelectronics. For small scale academic projects, the price of a small batch of up to 100 die, the fabrication cost in US $ can be estimated by:

$$p_{100} = \begin{cases} 125400 + (A-12) * 7700, & \text{if } A > 12mm^2 \\ 20900 + (A-2) * 9900, & \text{if } 2mm^2 \leq A \leq 12mm^2 \end{cases}$$

where A is the die area in mm^2 and p_{100} is the price of the first 100 die in US Dollars (USD). For small scale projects with more than 100 die, the price for a lot of 100 extra die is between 21,120 and 38,500 USD depending on the die area and the number of reticules in a wafer. MPW runs uses Multi Layer Reticule technic to reduce the overall cost of the mask and additional dies. For our purposes, we consider a small scale project to be a project with at most 25 wafers [75] . For large scale projects, a market study published at the FDSOI Forum in 2018 showed that the die manufacturing cost per 40 mm^2 is 0.9 USD for the 28nm technology [76]. Hence, our methodology for estimating the costs consists of the three parts we explained. In reality, a more accurate methodology is probably available for the fabs to fill in the gaps. However, we believe that the overall cost will be in the same range.

On top of the fabrication cost, we need to consider the running cost of the ASIC cluster, which includes the energy consumption and cooling. We have performed post-layout extraction and simulation in order to estimate the power consumption of the different chips. In order to simplify the cost analysis, we use a figure of 18 cents/KWh, which is higher than the electricity consumption price in most countries [77]. Hence, we only consider the energy consumption of the chips and not the cooling cost or other factors that will be added after fabrication. The performances and power result are provided in Table 3.1.

Early studies [78] demonstrated the effectiveness of body biasing in reducing leakage, improving performances, and worst case power consumption. This is an interesting feature for high performance computing, and practical cryptanalysis. Indeed this feature allows to get the best possible performance at given desired energy point. For a single targeted attack, the energy cost is not the critical factor in the overall attack cost. However it has a direct impact on the complexity of the cooling infrastructure when the attack complexity gets high. Moreover, for multiple attacks scenario, the energy becomes a critical factor.

The STMicroelectronics CMOS FD-SOI 28nm technology has been chosen for our simulations for its very good *power* × *performance* × *cost* product capability compared to the its earlier predecessors CMOS 40nm and 65nm, its availability in our testing environment and the availability of enough public information regarding its pricing. The ASIC chip in Figure 3.4 is composed of slave cores, which can either be birthday or neutral-bit slaves. Our digital design flow is shown in below Figure 3.5. Each slave has been synthesized with a top-down strategy using cadence RTL-compiler v14.8, while placement and routing were done using Cadence Innovus. A Power-Aware Synthesis and Placement-And-Routing are used. Power simulations are per-

formed with the pre- and post-placement and routing back-annotated netlist using Cadence Voltus. The slave is then imported as hard macro in Cadence Virtuoso and instantiated from the top-level RTL. The slave and the interface are then placed and routed in Virtuoso GXL.

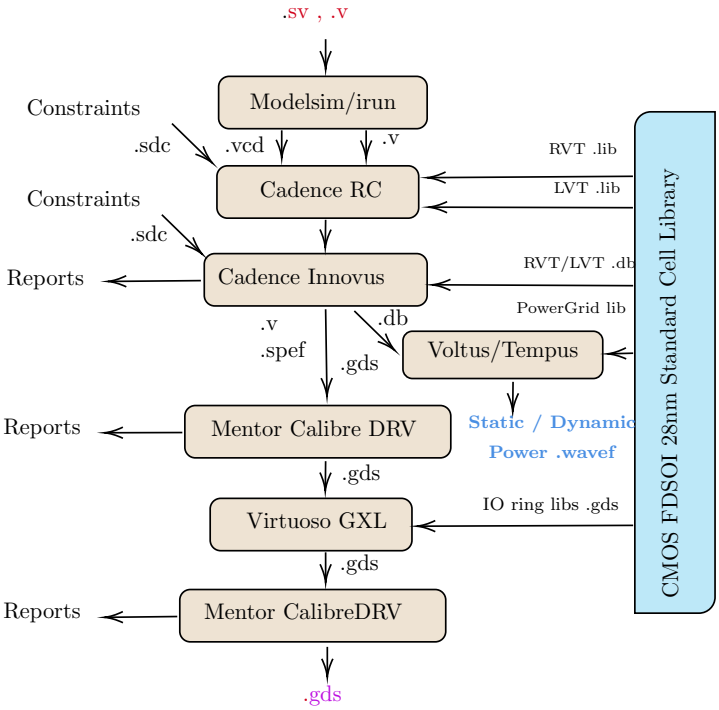


Figure 3.5: Our Bottom-Up ASIC digital design flow

The power rail and clock tree are routed with large tracks from the closest power supply and clock pins so as to reduce local voltage drop effect. The RC parasitic extraction of the NB/BD core GDS and final layout is done using Cadence QRC.

Mentor CalibreDRV is then used for the sign-off DRC and LVS checks. Our design mixes both Regular-Vt (RVT) and Low-Vt (LVT) cells. LVT cells are used without poly-biasing (PB0) for the critical path. RVT cells with poly-biasing up to PB16 are used for the rest of the circuit in order to minimize the leakage power.

Nominal process variation for both PMOS and NMOS for the pre/post-placement-and-routing power simulations with 0.92V supply voltage at 25 degree Celcius are used as parameter for the high performance version of our design. The circuit is first synthesized to reach the maximal operating frequency. Our high speed version reaches 909MHz with V_{fbb}=0V and 1262MHz with V_{fbb}=+2V. LVT cells have then been chosen for the critical path of the NB/BD core. The rest of the circuit have been synthesized with RVT cells so that to balance the performance and power consumption. Each slave is isolated

using triple-well isolation to reduce parasitic substrate noise between the slaves that reduces the overall performances. Power simulations show that our $16mm^2$ die requires 140 power supply pins and a plastic-ceramic package to dissipate the power. The effect of body biasing on power and delay after place and route is simulated using Cadence Genus and Voltus. Parasitic extraction with QRC is done with typical parameters. The performances and power result are provided in Table 3.1.

3.7. Chip layout

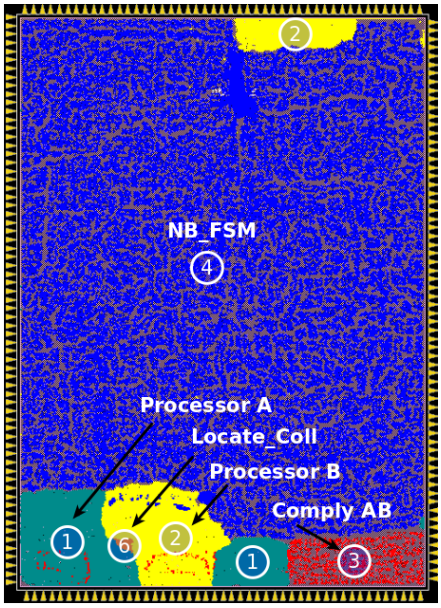


Figure 3.6: NB Slave ASIC CMOS 28nm FD-SOI layout.

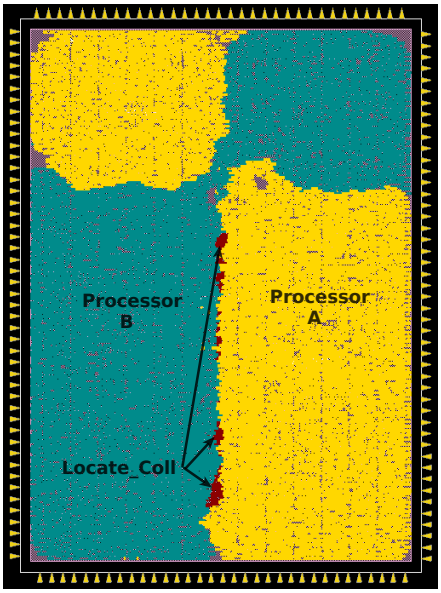


Figure 3.7: Layout Birthday core.

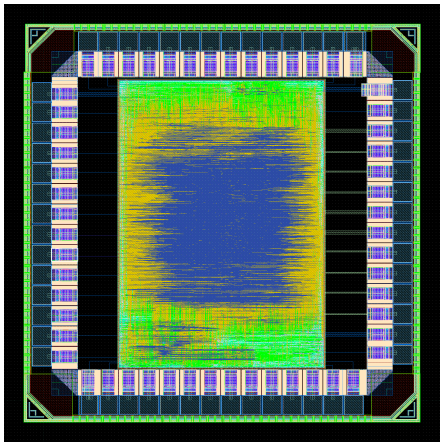


Figure 3.8: Sample $1mm^2$. ASIC layout with 1 NB core.

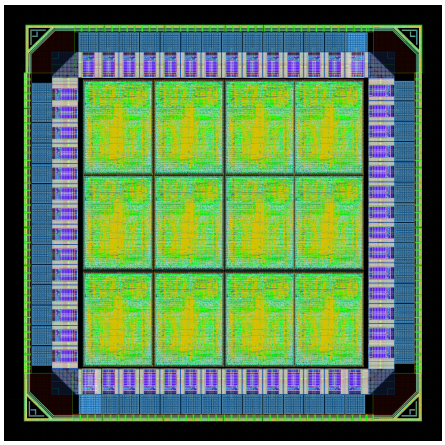


Figure 3.9: Sample $1mm^2$ ASIC layout with 12 BD core.

3.8. Verification

In order to verify the functionality of the ASIC implementation of the neutral bit algorithm, we have implemented it also in software and we have checked that the outputs and the intermediate values from the two implementations match. This process is described in more detail next.

In Algorithm 1, we give a description of the neutral bit search algorithm in pseudo-code as implemented in software. It follows the high-level description given in Section 3.2. In more detail, all neutral bits are provided as an input to the algorithm in the form of 512-bit masks. Each mask selects one or more bits from the original 16 byte message, resulting resp. in a single neutral bit or a set of neutral bits (multibits). Note that the latter also include the boomerangs.

The set of all neutral bit (NB) masks is partitioned in n_k subsets, such that all NB from the n_k-th subset are neutral up to step k inclusive. Such order allows to apply the neutral bits recursively in a breadth-first manner from n_k to n_{k+1}. If a neutral (multi)bit from subset n_k fails (i.e. results in a message that does not follow the differential path), then the search does not explore any neutral bits from subset n_{k+1} for the particular failing combination at n_k. In this way failing branches of the search tree (Fig. 3.1) are abandoned early during the search.

An equivalent implementation as the one described above was developed also in hardware. The verification of the equivalence of the two implementations was performed as follows. The execution of the software program is stored in the form of a trace containing the following information: step k, neutral bit mask m_k from subset n_k applied at step k and a list of all differential pairs of modified internal states (A_i, A'_i) at steps $i = k, k+1, \ldots, k'$, where step k' is the step at which the pair of internal states $(A_{k'}, A'_{k'})$ has failed to follow the differential path. The hardware implementation takes as input the trace produced by the software together with a list of all neutral bit masks and verifies that the values of all internal state pairs (A_k, A'_k) match the ones produced by the hardware and fail at the same step k' given in the software trace.

Algorithm 1 Apply neutral bits.

Input: —

i: Message step $i \geq 13$ (correspond to internal state step $i+1$)
P: Path (composed of internal state A and expanded message W)
S_i: Base solution with fully instantiated first 16 message words $W_0, \ldots, W_{15}$ and following path P up to and including message step $i \geq 13$
$N_k[0 \ldots n_k - 1]$: an array of n_k 512-bit masks. Each mask is a single bit or a multibit neutral bit (NBit) combination that is neutral up to and including message step $k : 13 \leq k \leq 18$. (Note: a multibit is a collection of several bits that have to be flipped together)

Output: —

S_j: Base solution following path P up to and including message step $j > i$

apply_neutral_multibits(i, S_i, P)
// If no more NBits to assign, keep computing step by step until solution fails P
if $i > 18$ **then**
    while S_i follows P up to step i inclusive **do**
        compute S_{i+1}
        $i \leftarrow i + 1$
    end while
    return S_{i-1}
end if
// Get the original 16 message words to be modified by the NBbits
$(W_0||W_1||...||W_{15}) \leftarrow S_i$
// For all 2^{n_i} combinations of (multi)bits neutral up to step i inclusive
for all $l = 0, 1 \ldots, (2^{n_i} - 1)$ combinations of NBits up to message step i **do**
    // Apply the l-th combination of NBits up to step i by XOR-ing all masks
    // that compose it to the initial 16 message words
    for all $(N_i[q] : 0 \leq q < n_i)$ that belong to combination l **do**
        $(W_0||W_1||...||W_{15}) \leftarrow N_i[q] \oplus (W_0||W_1||...||W_{15})$
    end for
    // Store the modified 16 message words back to the solution
    $S_i \leftarrow (W_0||W_1||...||W_{15})$
    // Neutral bit probability
    if S_i follows P up to message step i inclusive **then**
        // Compute next message step and call recursively the function
        compute S_{i+1}
        // Differential step probability
        if S_{i+1} follows P up to message step $i+1$ inclusive **then**
            apply_neutral_multibits$(i+1, S_{i+1}, P)$
        end if
    end if
end for

3.8.1. Results

Two different architectures of SHA-1 crackers are compared here. The first architecture is based on 2 separate ASIC slaves that handle the two parts of the attack, *i.e.,* the birthday search (BD) and the neutral bits part (NB). The two phases are performed sequentially. Figure 3.10 depicts the overall cost required to build the machine and find the first chosen-prefix collision depending on the time ratio between the two phases. For ASIC, the overall

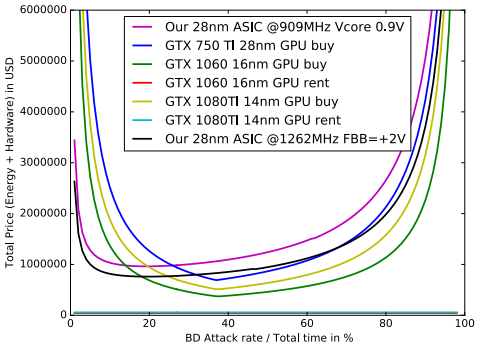


Figure 3.10: Impact of the BD/NB time ratio on the cost

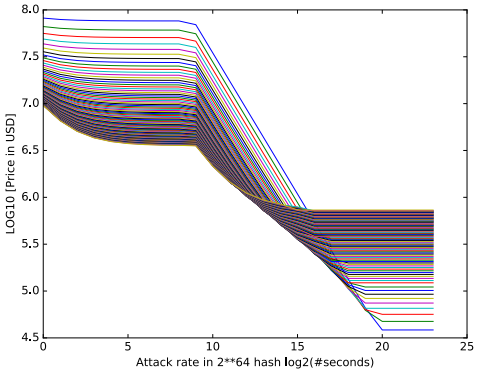


Figure 3.11: Impact of the die size and latency on the HW cost (4 to 100 mm^2 28nm FD-SOI). The top left line in blue represents 4 mm^2 and the bottom left is 100 mm^2.

minimum cost is not perfectly at 50% ratio. Hence, we consider a two-stage pipeline architecture at the cost of slightly more hardware to balance the birthday and neutral-bit parts.

Our birthday (BD) core uses 16927.1 gate equivalents (GEs) per SHA-1 rounds., while our neutral-bit core (NB) uses 170442.7 GEs. Our best implementation is a 4-round SHA-1 unrolled compression function that can be clocked at 900 MHz at V_{core}=0.92V and V_{fbb}=0V. Using body biasing and LVT transistors for the critical path, we can further decrease the threshold voltage and increase the running frequency. With V_{fbb}=+2.0V we can increase the running frequency of our fastest core by 40%, reaching 1262 MHz with a 2% increase in dissipated power. The chip can be further over-clocked by increasing V_{core} but at the cost of a quadratic increase in the dissipated power, so a more costly cooling system. The results of our implementations are shown in Table 3.1. As shown in Figure 3.8 , a BD slave contains up to ∼ 15 BD cores per

mm^2 while an NB slave contains ~ 1.5 NB cores per mm^2.

Version	900 MHz V_{fbb}=0V		1262 MHz V_{fbb}=+2V	
	BD	NB	BD	NB
Power (in mW)	71.1	289	72.6	294
CP delay (in ps)	1110	1110	792	792
Area (in mm^2)	0.0650	0.6545	0.0650	0.6545

Table 3.1: ASIC implementation performances for 2 corners cases : high performance at 900 MHz and high performance with FBB at 1262 MHz.

In our study, the overall cost is calculated without the cooling and infrastructure. Note that as shown in Figure 3.11, the total cost required to build an ASIC-based cracker greatly depends on the die size. This is due to the fact that the initial cost is predominant when the die size is large. The overall hardware cost tends to the same for any die size when the attack is fast.

3.8.2. Attack Rates and Execution Time

As shown in Table 3.2, a single NB slave of $16mm^2$ contains up to 24 NB cores and can generate up to 976 solutions up to step 40 of SHA-1 per second. Each solution A_{40} requires 31 Million cycles, on average. A single BD slave of $16mm^2$ contains up to 245 BD cores and provides a hash rate of 20.6 GH/s for the fastest version of our design. As a comparison, as shown in Table 3.3 and taken from [68], a single GTX 1060 GPU provides a hash rate of 4.0GH/s and can generate 2000 A_{40} solutions per second. If we take the birthday part of the attack as a reference, the neutral bit part is ten time less efficient in hardware than on GPU.

The second architecture is based on GPU. For GPU, it is cost-wise more interesting to take advantage of its reconfigurability to minimize the cost. Hence, we consider in our cost analysis that the chosen-prefix collision is performed serially by reusing the same GPU for the two attack phases. In Table 3.4, the cost of the three attack scenarios is provided. We give in this table the cost to build the ASIC- and GPU-based clusters for 3 different speeds, *i.e.*, one attack per month, one attack per day and one attack per minute. The latency corresponds to the delay to get the first collision. For instance, a two-stage ASIC-based machine able to generate one SHA-1 collision every months, will generate the first collision in two months. A GPU-based machine generates

Parameter	900 MHz	1262 MHz
SHA-1/core/sec	$2^{25.8}$	$2^{26.3}$
SHA-1/core/month	$2^{47.1}$	$2^{47.6}$
SHA-1/chip/month	$2^{55.1}$	$2^{55.6}$
A_{40} Solutions/core/sec	$2^{4.9}$	$2^{5.3}$
A_{40} Solutions/core/month	$2^{26.1}$	$2^{26.7}$
A_{40} Solutions/chip/month	$2^{30.8}$	$2^{31.2}$

Table 3.2: Our best $16mm^2$ ASIC implementation performances for 2 corners

GPU	arch	Hash Rate	A_{33} rate	A_{40} rate	Price	Power	Rental
GTX 750 Ti	Maxwell	0.9GH/s	62k/s	250/s	$144	60W	
GTX 1060	Pascal	4.0GH/s	470k/s	2k/s	$300	120W	$35/month
GTX 1080 Ti	Pascal	12.8GH/s	1500k/s	6.2k/s	$1300	250W	

Table 3.3: SHA-1 hash rate from hashcat for various GPU models, as well as measured rate of solutions at step 33 (A_{33}-solutions). Data taken from [68].

the first collision in one month for the same attack rate. Our ASIC-based two stage pipelined architecture has twice the latency of a sequential GPU-based machine for the same attack rate. Our benchmark (Figures 3.20 and 3.21) provides a comparison between our ASIC cluster and two of the most widely spread GPU based machines, *i.e.*, the GTX 1080TI (CMOS 14nm) and the GTX 1060 (CMOS 16nm) for different attack rates. The numbers for the GTX 750 TI (CMOS 28nm technology) are also added to the benchmark as it provides an idea of the performance obtained with a GPU based on a similar technology node as our ASIC.

Note on the use of FPGAs Our ASIC design have been tested on FPGA platform. FPGA can be considered as a good alternative to ASIC thanks to its reconfigurability property. However, one of the largest FPGAs from Xilinx, namely the Virtex 7 xc7vx330t-3ffg1157 can fit only 20 instances of the Birthday core running at 135MHz in one chip. The same FPGA can fit only 16 instances of the Neutral Bit core running at 133MHz. In order to do the

Platform	ASIC			GPU rent			GPU buy		
Attack	64	CPC	80	64	CPC	80	64	CPC	80
Energy Cost	$776	$1.6k	$50.9M	-	-	-	$18k	$12k	$1.2B
Cluster for 1 attack per month									
Latency (month)	1	2	1	1	1	1	1	1	1
Hardware Cost	$257k	$1.1M	$11M	-	-	-	$715k	$490k	$47B
First Attack Cost	$257k	$1.1M	$61.9M	$61k	$43k	$4B	$733k	$502k	$48B
Amortized Cost	$7.9k	$32.1k	$51.2M	$61k	$43k	$4B	$38k	$26k	$2.5B
Cluster for 1 attack per day									
Latency (day)	1	2	1	1	1	1	1	1	1
Hardware Cost	$1.4M	$3.7M	$218M	-	-	-	$22M	$15M	$1.4T
First Attack Cost	$1.4M	$3.7M	$269M	$61k	$43k	$4B	$22M	$15M	$1.4T
Amortized Cost	$2k	$5k	$51.1M	$61k	$43k	$4B	$38k	$26k	$2.5B
Cluster for 1 attack per minute									
Latency (minute)	1	2	1	1	1	1	1	1	1
Hardware Cost	$8.5M	$48M	$263B	-	-	-	$31B	$21B	$2Q
First Attack Cost	$8.5M	$48M	$263B	$61k	$43k	$4B	$31B	$21B	$2Q
Amortized Cost	$781	$1.6k	$51M	$61k	$43k	$4B	$38k	$26k	$2.5B

Table 3.4: Comparison of attack costs with various parameters. Costs are given in USD (k stands for thousand, M for Million, B for Billion, T for Trillion, Q for Quadrillion). Amortized cost is the cost per attack assuming that the hardware is used continuously during three years. Note that it is possible to get slightly more energy efficient platforms and implementations at the cost of more expensive hardware. We list the cheapest platform after one attack, energy included.

2^{64} generic birthday search, we need $2^{36.6}$ FPGA-seconds, *i.e.*, in order to do it in one month we need $2^{15.3}$ FPGAs. As a single FPGA costs around 8000 USD, this attack would cost around 319 Million USD. This is more than one thousand times the cost of the same attack on ASIC and 440 times the cost on GPU, making it irrelevant for the purpose of analyzing SHA-1. Even if FPGAs can be rented, a similar factor is expected compared to renting GPUs.

3.9. Cost Analysis and Comparisons

As explained throughout the paper, we have performed several experiments to identify the different implementation trade-offs for the attack scenarios we consider. In this section, we analyze the cost estimates of implementing these attacks in ASIC vs. consumer GPU. We consider three attack scenarios that fall into two categories: generic birthday attacks and differential cryptanalysis of SHA-1. Before discussing the analysis in more details, here are a few general conclusions that we reached through our experiments, which can be helpful for building future hardware crackers:

1. The cost of implementing memoryless generic attacks, such as the parallel collision search of [18], in hardware can range from 20% to 50% of the overall ASIC implementation, while the rest is dedicated to the attacked primitive, e.g. the SHA-1 hash function.
2. For iterative cryptographic algorithms, such as hash functions and block ciphers, a way to reduce the attack cost is to use unrolling. This approach is similar to using memoryless algorithms. Instead of computing one step of the function every clock cycle, we compute several steps in the same cycle. This amortizes the costs of the attack logic among several steps. For example, implementing the birthday attack using a single-step iterative SHA-1 core leads to a circuits where only 20% of the area is used by the SHA-1 logic and 80% of the area is due to the attack logic, registers and comparisons. On the other hand, using a core that computes 4 steps every clock cycles leads to a circuit with a 50%/50% ratio. While this technique may increase the critical path of the circuit and reduce the frequency, it also reduces the overall number of cycles, so the overall time to compute a single SHA-1 per core is almost constant.
3. For cryptanalytic attacks, the cost is dominated by the attack logic, which may include a huge number of comparisons, modifications and registers. These extra operations are usually different from one step to another, so they consume a huge area. Besides, the state machine of these attacks can be very costly. In such scenarios, the advantage of using ASICs becomes diminished compared to consumer GPUs, except for very high budgets, especially as the GPUs are reusable and can be rented.

3.9.1. 2^{64} Birthday Attack

The first attack scenario we consider is attacking a hash function with 2^{64} birthday collision complexity. The hash function used is the SHA-1 compres-

sion function reduced to only 128 output bits, as explained in Section 3.2. A single ASIC core is described in Section 3.3. The time to finish such an attack depends on the number of chips fabricated and the size of each chip. A single ASIC core running at 1262 MHz contributes $2^{26.33}$ SHA-1 computations per second. The attack costs $2^{37.67}$ core-seconds. To reach this complexity, Figure 3.11 shows the price required vs. the estimated time needed to finish the attack, including the fabrication cost of chips of different sizes and the energy consumption.

To put these numbers into perspective, the NVIDIA GeForce GTX 1080 TI GPU (14nm technology) can do about $2^{33.6}$ SHA-1 computations per second, so implementing the attack on GPU would require $2^{30.4}$ GPU-seconds. In order to implement this attack in one month, we need to buy around 550 GPUs costing around 715k USD and around 18k USD in energy. As shown in Figure 3.12, a GTX 1060-based machine is a bit less expensive, costing 525k USD but consuming around 28k in energy for the same job (using 1750 GPUs).

Besides, as shown in Figure 3.12, for any attack rate it is cheaper to buy an ASIC cluster than a GPU-based cluster. The difference reaches 1 order of magnitude from a rate of 1 attack per week. Furthermore, the ASIC-based cluster consumes 1 to 2 order of magnitude less energy than any GPU-based solution. As shown in Table 3.4, the minimum cost in energy per attack on ASIC is as low as 776 USD. An ASIC-based cracker able to generate one collision per month would cost 257k USD. For an attack rate of 1 attack per minute, it would cost 8.5 million USD.

An alternative option is to rent the GPUs. This would cost around $61k per attack, assuming a rental price of $209/month for a machine with 6 GTX 1060 GPUs. This makes the GPU rental very competitive for a single attack, around 4 times cheaper than an ASIC cluster. However, the ASIC cluster quickly become much more cost effective when the attack is repeated (see Figure 3.14).

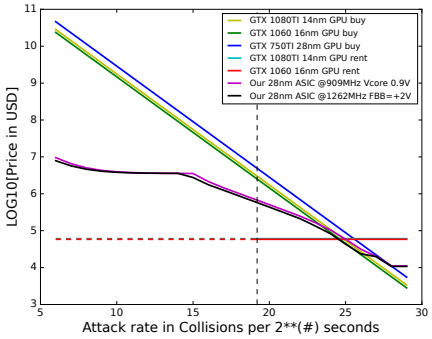


Figure 3.12: 2^{64} BD machine price for different attack rates: ASIC vs GPU

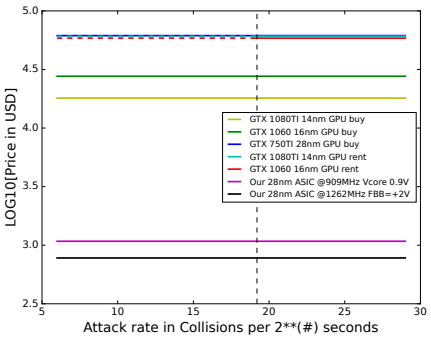


Figure 3.13: Energy cost per 2^{64} BD attack: ASIC vs GPU

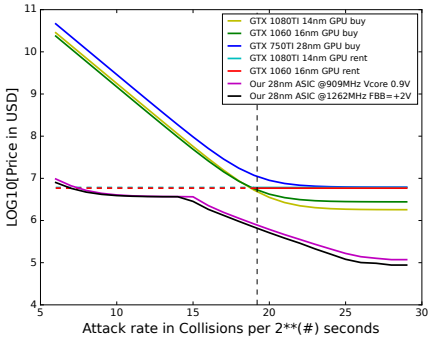


Figure 3.14: Total cost (HW+E) for 100 2^{64} BD attack at a given attack rate: ASIC vs GPU

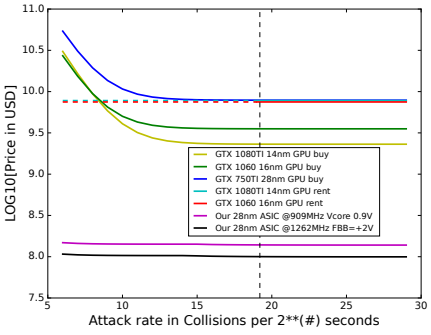


Figure 3.15: Total cost (HW+E) for 100k 2^{64} BD attack at a given attack rate: ASIC vs GPU

3.9.2. 2^{80} Birthday Attack

In this section, we look at the cost of implementing a generic birthday collision search for the full SHA-1 output, which requires around 2^{80} SHA-1 computations. The algorithm is the same as the previous attack, except that we use the full output of the SHA-1 compression function. Since a single ASIC core performs $2^{26.33}$ SHA-1 computations per second, the birthday collision search costs $2^{53.67}$ core-seconds, or around 454 million years on a single core. Fortunately, for a powerful attacker with enough money, the cost for producing ASICs grows slowly for large number of chips. The fabrication cost of a hardware cluster to perform the attack in one month costs only 11 million USD, as opposed to around 34 billion USD for GTX 1060. Hence in this case, for any attack rate as shown in Graphs 3.18 and 3.19 the only realistic option is to build an ASIC cluster.

Running the attack costs around 50.9 million USD in energy, which matches

the order of magnitude estimated from the bitcoin network: the network currently computes about $2^{70.2}$ SHA-256 every ten minutes, for a reward of 12.5 bitcoin, or roughly $85k at the time of writing. This would price a 2^{80} computation at 75 million USD.

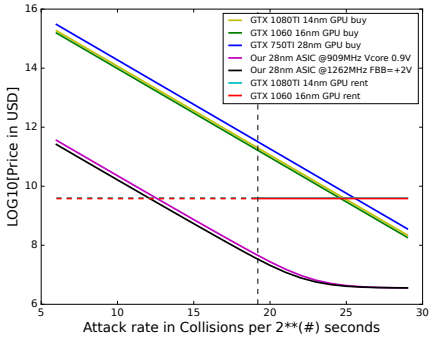


Figure 3.16: 2^{80} BD machine price for different attack rates: ASIC vs GPU

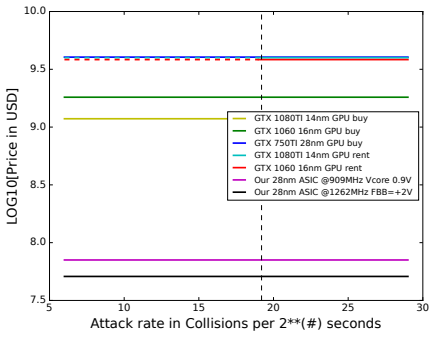


Figure 3.17: Energy cost per 2^{80} BD attack: ASIC vs GPU

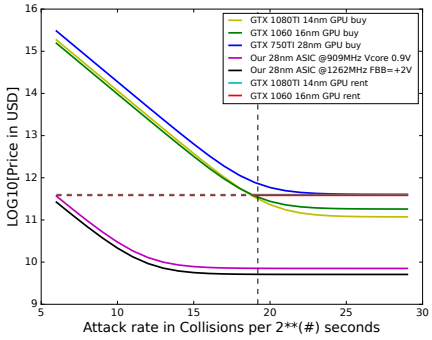


Figure 3.18: Total cost (HW+E) for 100 2^{80} BD attack at a given attack rate: ASIC vs GPU

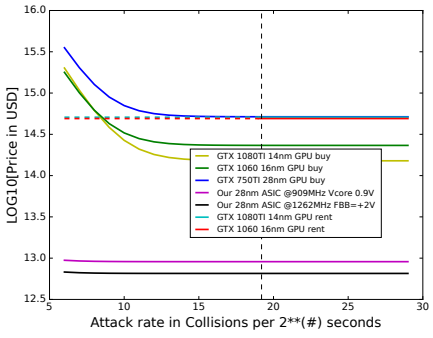


Figure 3.19: Total cost (HW+E) for 100k 2^{80} attack at a given attack rate: ASIC vs GPU

3.9.3. Chosen Prefix Differential Collision Attack

The chosen-prefix collision attack proposed by Leurent and Peyrin [68] consists of two main parts: a birthday search attack, and a differential collision attack. The authors provide different trade-offs between the complexity of the two parts. In their paper, the number of solutions required for the neutral bits up to step 33 is provided. This number of solutions corresponds to the number of solutions required to get a valid solution with high probability. Step 33 is chosen because there is a zero difference at this state, so there is a single path at this step, and solutions are generated fast enough to measure the rate easily. This configuration requires to generate about $2^{62.05}$ SHA-1

computations for the birthday part and $2^{49.78}$ solutions up to step 33. In this paper, it is cost-wise more interesting for ASIC to generate solutions for the neutral bits up to step A_{40}.There is a factor $2^{7.91}$ difference in the number of solutions to generate between step A_{33} and step A_{40}. Hence a chosen-prefix collision requires to generate $2^{41.87}$ solutions. Table 3.3 provides the hash rates and solution rates numbers used in our estimate for the cost on GPU. This gives 38 GPU-years for the birthday, and 65 years for the neutral bits. The estimated cost per attack using GTX 1060 GPU, assuming 209 USD per month for 6 GPU is about 43k USD. The cost of running the attack in GPU is dominated by the energy consumption. ASIC is much more energy efficient, as shown in Figure 3.21. It can be up to 2 order of magnitude less than using common consumer GPU. As shown in Figure 3.20, ASIC-based SHA-1 cracker that generate one collision per month, costs about 1.1 million USD, about the same as the cheapest GPU-based cracker from our benchmark. However, a single attack on GPU costs about 19000 USD in energy. Hence from 100 attacks as shown in Figure 3.22 and 3.23 as well as for attack rates greater than 1 attack per week, an ASIC-based SHA-1 cracker is the only realistic option.

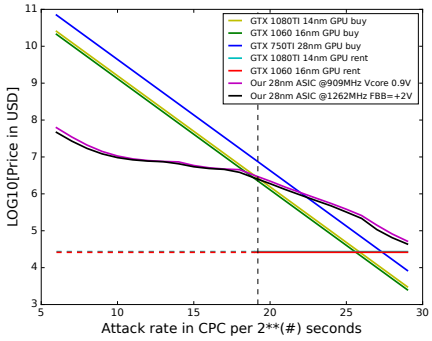


Figure 3.20: CPC machine price for different attack rates: ASIC vs GPU

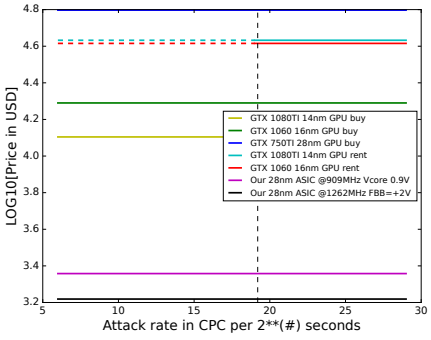


Figure 3.21: Energy cost per CPC attack: ASIC vs GPU

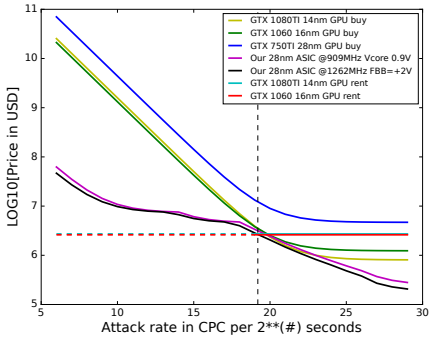


Figure 3.22: Total cost (HW+E) for 100 CPC attack at a given attack rate: ASIC vs GPU

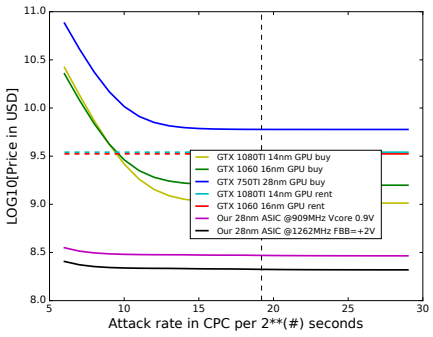


Figure 3.23: Total cost (HW+E) for 100k CPC attack at a given attack rate: ASIC vs GPU

3.9.4. Limitations

While we did our best to estimate the price of the attacks as accurately as possible, our figures should only be considered as orders of magnitude because the pricing of hardware and energy can vary significantly. ASIC pricing is not completely public, and energy prices depend on the country. Moreover, our estimate only include hardware cost and energy, neglecting other operating costs such as cooling and servers to control the cluster (however, the energy price we use is somewhat high, so it can be considered as including some operating costs).

Another caveat is that we only consider the computation part of the attacks. In reality, there is some need for communication between the nodes, and some steps of the attacks must be done sequentially. Concretely, the generic birthday attacks must sort the data after computing all the chains, and the CPC attack must compute several near-collision blocks sequentially. This will likely add some latency to the computation, and running the attack in one minute will be be a huge challenge, even when the required computational power is available.

3.10. Conclusion

Our paper provides a precise comparison between ASIC-based and GPU-based solutions for cryptanalysis, with a case study on generic birthday search and a case study on the recent chosen-prefix collision on SHA-1. For the former, we show that generic birthday attacks can be performed very easily with ASICs against a 128-bit hash function, and that even a 160-bit hash function would not stand against a huge, yet potentially affordable, ASIC cluster. For

the latter, we created two independent ASICs that handle the two parts separately. Our comparisons with GPU-based solutions show a clear advantage of ASIC-based solutions. In particular, we remark that the chosen-prefix collisions for SHA-1 can be generated in under a minute, with an ASIC cluster that costs a few dozen Millions dollars. Such ability would allow an attacker to apply the SLOTH attack [69] on TLS or SSH connections using SHA-1.

In the introduction, we posed three research questions; the first question is related to the cost of attacks on SHA-1. Our study showed that ASIC is clearly the best choice for very high complexities attacks, or for attacks that need to be performed in a short amount of time. However, for proof-of- concept or cryptographic research in general, where complexities of 2^{64} or less can be computed in a month or so, renting a set of GPUs seems to be the best solution. If the attack needs to be repeated multiple times, or if the speed of the attack is critical, then the initial hardware cost might be amortized and the energy cost per attack might become important. We note that the energy cost will be very high on GPU compared to a dedicated ASIC solution. For a chosen-prefix collision on SHA-1, the energy cost per attack for our speed-optimized ASIC is 1.6k USD. The best GPU based solution from our benchmarks consumes about 12k USD per attack. Hence, the cost of the ASIC-based solution is amortized. Furthermore, when the CPC attack rate becomes higher than 100 attacks per month, the ASIC solution is cheaper than any GPU-based solution in our benchmarks. In this case, the cost of the GPU rent is prohibitive and the ASIC is the only realistic threat.

In the second question, we target the comparison between generic attacks and cryptanalytic attacks for similar theoretical level of numeric complexity. In our study, we show that for a similar level of $\sim 2^{64}$ computations, it is $\sim 75 \sim 82\%$ cheaper to implement a generic birthday search, compared to the differential CPC attack on SHA-1. This means that for these two attacks, the generic attack has an advantage of 5×. We need to study more cases, such as the biclique attacks on block ciphers compared to the generic brute force attacks.

Last but not least, the third question is whether the 80-bit security level is still adequate for practical use in less demanding applications. Our study is a warning, showing that not only SHA-1 is indeed practically fully broken, but also that search-based and memory-less generic attacks with complexity $\leq 2^{80}$ are within practical reach.

III

Passive Side-Channel Attacks on implementations

This section focuses on the second constraint considered in Figure 1.1, which is the resistance of cryptographic algorithms against side-channel attacks. Side-channel attacks are a type of attack that exploits information leaked by a cryptographic system during its computation, such as power consumption or electromagnetic radiation.

Resisting side-channel attacks is a critical consideration in cryptographic system design, particularly for applications that require high levels of security. This is because side-channel attacks can reveal sensitive information about the cryptographic key or plaintext, even if the cipher itself is considered secure.

There are several techniques that can be used to enhance the resistance of cryptographic algorithms against side-channel attacks, including masking, hiding, and shuffling. These techniques involve adding noise or randomness to the computation process, in order to make it more difficult for an attacker to extract sensitive information.

In addition to these techniques, the choice of hardware and software implementation can also impact the resistance of cryptographic algorithms against side-channel attacks. For example, using specialized hardware or implementing the algorithm in a way that minimizes power consumption can help reduce the information leakage that can be exploited by an attacker.

Overall, the resistance of cryptographic algorithms against side-channel attacks is a critical aspect of cryptographic system design, and requires careful consideration and evaluation to ensure that the resulting system is secure against a wide range of potential attack vectors.

4

On Comparing Side-channel Properties of AES and ChaCha20 on Microcontrollers

Side-channel attacks are a real threat to many secure systems. In this paper, we consider two ciphers used in the automotive industry – AES and ChaCha20 and we evaluate their resistance against side-channel attacks. In particular, the focus is laid upon the main non-linear component in these ciphers. Owing to the design of ChaCha20, it offers natural timing side-channel resistance and thus is suitable for affected applications. However, attacks exploiting the power side-channel are somewhat more difficult on ChaCha20 as compared to AES, but the overhead to protect ChaCha20 against such attack is considerably higher.

4.1. Introduction

The automotive industry has seen a major digital transformation in recent years. Nowadays, electronics can make up to 60-70% of a car's development cost and consequently, many electronic control units (ECUs) communicate with each other or to external units, to monitor and manage tasks of various criticality. Any manipulation of the communicated data can have a direct ef-

fect on the functional safety of the vehicle and thus, can lead to serious consequences. To protect against such threats, suites of standard cryptographic algorithms are used to ensure the integrity of the data.

For securing the data, standards for symmetric encryption like AES [79] and ChaCha20 [80] are used in the automotive industry. While many lightweight ciphers have been developed in recent years, only AES and ChaCha20 are usually used in automotive application. One of many threats for automotive applications are side-channel attacks (SCA [81]), which target implementations of cryptographic algorithms and lead to key recovery in negligible computation time. While AES is a standard for the last two decades, ChaCha20 is seeing a rapid adoption due to its software-friendly design and natural resistance to timing side-channel attacks.

AES uses a substitution box (S-box) as the only nonlinear component, which can be either pre-computed or calculated on the fly which involves the inversion in the Galois field GF(2^8). ChaCha20 uses a modular addition instead, which is natively supported on a wide range of general purpose microcontrollers. Since SCA primarily targets the nonlinear component, these two operations become the focus of our study. To perform a detailed analysis, we use profiled machine learning attacks and non-profiled correlation power analysis, some of the most commonly used SCA techniques.

When considering countermeasures, it is also the nonlinear operation which is expensive to implement protected against side-channel attacks, unlike other linear components of the cipher. Thus, we also study and compare the cost of protecting these two basic functions (S-box and modular addition) against SCA. Other physical attacks like fault attacks and hardware Trojan are considered out of scope due to different attack model and modus operandi.

The rest of the paper is organized as follow. Section 7.2 recalls basics of target algorithms and side-channel attacks. Section 4.3 compares the susceptibility of the AES S-box and the modular addition to profiled machine learning-based attacks and non-profiled correlation power analysis. In this paper, we use simulated measurements for the analysis. Section 4.4 analyzes the two nonlinear operation in terms of the side-channel protected implementation overhead. Finally, conclusions are drawn in Section 13.7.

4.2. Background

4.2.1. Target Algorithms

AES [79]

The Advanced Encryption Standard (AES) is the NIST standard for block ciphers [82]. AES follows the Substitution Permutation Network (SPN) construction and operates on 128 bit data blocks with a 128/192/256 bit secret key in 10/12/14 rounds [79]. Four distinct operations comprise a round, i.e., SubBytes (SB), ShiftRows (SR), MixColumns (MC), and AddRoundKeys. The operations are applied on the 128-bit data state that is organized as a 4×4 matrix of bytes. SubBytes is a non-linear permutation on individual bytes. ShiftRows and MixColumns are linear permutations of the state and AddRoundKeys performs exclusive-or (XOR) between round keys and the data state. A key expansion algorithm is used to derive round keys from the master key. All rounds are identical except the last one, which skips MixColumns. The SubBytes operation is based on the computation of the inverse of an element of GF(2^8), followed by an affine transformation. Often, the computation is performed using a precomputed table. The implementation of an S-box is not easy and often represents the main source of a large time or memory footprint. Hardware support for AES is currently available in a large range of processors from vendors like Intel, AMD, or ARM. With the built-in support, recent applications have seen significant performance improvements.

ChaCha20 [80]

The ChaCha20 stream cipher is a part of ChaCha20-Poly1305 Authenticated Encryption with Associated Data (AEAD) suite. It follows the ARX construction, i.e., it is based only on addition, rotation, and XOR. Its design is optimized for efficiency and is easily implemented as a time-constant software. It produces a key stream with the internal state initialized using 4×32 bit constant values $c_0, \ldots, c_3$, an 8×32 bit width secret key $k_0, \ldots, k_7$, 3×32 bit nonce values n_0, n_1, n_2, and a 32 bit message block counter value count. Then, the complete state is copied, before the double-round function is called 10 times. The plaintext is XORed with the key stream to get the ciphertext. The only nonlinear function in ChaCha20 is the modular addition, which is very efficient in both hardware and software. Being fairly new, currently no hardware support is available, but ChaCha20 is anyhow adopted in many commonly used cryptographic libraries.

4.2.2. Side-Channel Attacks and Metrics

Profiled Side Channel Attacks

Profiled SCA assumes a strong attacker who has access to a clone device allowing detailed characterization of the leakage. While earlier proposals used template attacks [83], machine learning classifiers were shown to outperform template attacks in certain scenarios [84].

Random Forest (RF) is a well-known ensemble decision tree learner [85]. Decision trees choose their splitting attributes from a random subset of k attributes at each internal node. The best split is taken among these randomly chosen attributes and the trees are built without pruning, RF is a parametric algorithm with respect to the number of trees in the forest. Random Forest is a stochastic algorithm because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting. Learning time complexity for RF is approximately $O(I \cdot k \cdot NlogN)$. We use $I = [10, 50, 100, 200, 500, 1\,000]$ trees in the tuning phase, with no limit to the tree size. To evaluate the performance of RF, we use accuracy as the metric. Accuracy is defined as:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4.1}$$

Here, TP refers to true positive (correctly classified positive), TN to true negative (correctly classified negative), FP to false positive (falsely classified positive), and FN to false negative (falsely classified negative) instances.

We divide the traces into training and testing sets in a 2:1 ratio. On the training set, we conduct a 5-fold cross-validation where we use the averaged results of individual folds to select the best classifier parameters. We report results from the testing phase only, as these results are more relevant than the training set results in assessing the actual classification performance of the constructed models. All the machine learning experiments are done with the scikit-learn library [86] for Python.

Non-Profiled Side Channel Attacks

Non-profiled SCA assumes a weaker attacker who has no access to a clone device, but can observe side-channel activity for known inputs or outputs. Based on a leakage model (like the Hamming Weight or the Hamming Distance), the attacker can estimate the leakage based on key hypotheses. Next, statistical means like correlation are used to find a dependency between the observed leakage and the estimated leakage. The correlation maximizes for the estimated leakage with correct key, thus revealing its value to the attacker. We use the Pearson correlation coefficient as the statistical distin-

guisher. The divide and conquer approach allows to treat small parts of the key (for example, one byte) at a time, making the complexity practical for both profiled and non-profiled SCA.

Attack Metric

We use guessing entropy as the attack metric. The Guessing entropy measures the average number of key candidates to test after the attack. A guessing entropy of 1 signifies a successful attack. The Guessing entropy of the adversary $A_{E_k,L}$ against a key class variable S is defined as:

$$GE_{A_{E_K,L}}(\tau, m, k^*) \quad = \quad \mathsf{E}[\mathrm{Exp}_{A_{E_K,L}}].$$

4.3. Side-channel Analysis of Target Algorithms

In this section, we analyze and compare the side-channel susceptibility of the basic nonlinear functions of AES and ChaCha20. To perform the worst case analysis, we consider the strongest attacker with the profiling capability. Further, we use a machine learning classifier for the attack as such techniques have shown to perform better than template attacks in certain cases [84].

We consider a 32 bit microcontroller as our target as it is seeing rapid adoption in automotive applications. The simulated device is considered to be leaking in the perfect Hamming weight model with some added environmental (Gaussian) noise of standard deviation σ. Two levels of noise are tested. In one experiment, the AES S-box is implemented as a lookup table in memory, operating over 8-bits. Although the underlying architecture is 32 bit, due to memory restrictions, an 8 bit lookup is normally used and thus, we consider only S-box look-ups over 8 bit in our analysis.

For the case of the modular addition, our simulation is computed over 32 bits owing to the native support in the instruction set. An attack over 32 bits might be sometimes slow on normal system. However, the attacker has the freedom to observe it over any granularity from 1 bit to 32 bit. This is represented as (x/y/z) in Table 4.1. Here, x signifies the bit width of observation or the side-channel signal, while the remaining of 32 bit ($y = 32 - x$) contribute to algorithmic noise. $z = x + 1$ is the total number of classes in the attack.

Our results are shown in Table 4.1. With the low environmental noise and no algorithmic noise, the attack accuracy is 100%. For other cases, the accuracy reduces with increased noise and increased number of classes (z). In general, modular addition is marginally harder to attack as compared to the AES

Table 4.1: Classification Accuracy (%)

Noise	AES	ChaCha20			
σ	(8/0/9)	(4/28/5)	(8/24/9)	(16/16/17)	(32/0/33)
0.1	100	30.19	23.02	20.04	100
1	34.01	29.50	22.43	18.78	29.58

Table 4.2: Guessing Entropy for RF

σ: 0.1												
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	5.3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	10.0	6.7	6.3	3.0	2.0	2.0	1.7	2.0	1.0	1.0	1.0	1.0
Add 8	181.0	126.5	66.3	76.1	48.6	42.0	5.0	3.0	1.3	1.0	1.0	1.0
Add 16	21 075.8	12 505.3	23 298.3	11 935.6	809.7	37.3	103.0	4.3	3.0	1.0	1.0	1.0
Add 32	1 846.0	35.7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
σ: 1												
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	84.0	20.1	10.3	8.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	11.0	6.7	6.7	12.0	5.6	7.0	8.0	2.0	1.3	1.0	1.0	1.0
Add 8	134.0	69.0	36.2	49.7	38.0	57.0	86.0	7.6	16.0	3.0	1.0	1.0
Add 16	33 926.0	23 171.3	32 294.7	14 162.7	4 512.0	11 324.3	688.3	11.0	1.0	1.0	1.0	1.0
Add 32	14 305.3	3 134.3	5 736.3	22.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

S-box, because of the higher nonlinearity of the latter. Note that the accuracy for scenario (32/0/33) has 100% accuracy when $\sigma = 0.1$, which is much higher than for (16/16/17) case. Although this may sound counterintuitive, since the number of classes is much higher in the first case, there is a simple explanation of such a behavior. When the level of noise is small, there are clear boundaries between classes and RF does not have any problems when classifying. However, when we reduce the number of classes (granularity), then we actually see an overlap among certain measurements since now they can belong to several classes, which results in a wrong classification.

In addition to the accuracy metric, we use the guessing entropy as a metric for the SCA evaluation. In the attack scenario, the attacker is more interested in obtaining the correct secret key, rather than exact classification. The attacker can easily calculate the maximum likelihood for each key candidate, and derive the guessing entropy, i.e., the average rank of the correct key.

In Table 4.2, the guessing entropies for RF are presented. It shows that for

Table 4.3: Guessing Entropy for CPA

					σ: 0.1							
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	155.0	3.3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	5.7	10.0	15.0	10.3	5.6	2.3	1.0	1.0	1.0	1.0	1.0	1.0
Add 8	180.7	67.0	18.6	85.0	8.7	2.0	1.3	1.0	1.0	1.0	1.0	1.0
Add 16	40753.0	10 288.0	2 501.0	34.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 32	42 121.0	9.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
					σ: 1							
	No of traces (2^x)											
	1	2	3	4	5	6	7	8	9	10	11	12
S-box	184.4	56.7	35.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 4	13.0	12.0	7.3	3.6	6.4	2.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 8	226.0	104.0	71.1	9.7	9.0	1.3	1.0	1.0	1.0	1.0	1.0	1.0
Add 16	40 644.0	4 819.0	15 098.0	1 349.0	112.0	3.0	1.0	1.0	1.0	1.0	1.0	1.0
Add 32	43 313.0	1 840.0	114.0	6.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

$\sigma = 0.1$, with 2^{10} traces, it is enough to recover the correct key, whereas for $\sigma = 1$, it will require 2^{11} traces. In general, it can be observed that even though some of the labels are not predicted correctly, by taking the maximum likelihood, it is still possible to recover the secret key.

When considering non-profiled SCA, the results are presented in Table 4.3. The attack model here is quite different from the profiled attacks, thus a direct comparison is not possible. However, it can be clearly seen that in both cases, addition offers slightly higher resistance to SCA as compared to the S-box.

4.4. Towards Side-channel Protection

4.4.1. Preventing Timing Side-channels

In most cases, timing side-channels can be prevented by removing data-dependent branches and memory accesses. For ChaCha20, this is trivial, because no operation in the algorithm uses operations that are susceptible to such behavior [80].

However, a fast AES implementation uses lookup tables to implement either the S-boxes or so-called T-tables. Such table lookups usually do not have a constant time behavior, and thus, a naive AES implementation is vulnerable

to cache timing attacks [87]. Using bit-slicing, the data-dependent behavior can be removed, but at the cost of a reduced performance. An alternative countermeasure is to disable caches during the execution of AES, if possible.

Another possibility is to use hardware accelerators that many modern microcontrollers implement. For AES, this results in a much faster, constant-time computation. The downside is, that many AES accelerators are not resistant to power side-channel attacks and therefore, a software implementation is needed, if such a protection is required.

No modern microcontroller has a ChaCha20 accelerator so far and therefore, no fair comparison is possible. However, due to the construction of ChaCha20 as an ARX cipher, we believe that at least for low-end and mid-end processors, it is unlikely that accelerators will be developed. For high-end processors the usage of already existing SIMD instructions can lead to high speed ups.

When we look at our main target platforms, i.e., ARM, most of the time AES and ChaCha20 perform with similar speed. However, if hardware acceleration for AES is available, it becomes a significant advantage for AES.

4.4.2. Preventing Power Side-channels

Power side-channels are much harder to mitigate, because the overhead of protected implementations is much higher than for the protection against timing side-channels.

One of the most effective countermeasures against power analysis attacks is masking. In software, many masking schemes of nonlinear gates are often based on the Trichina gate [88] or optimized variants [89].

The common factor among all masking countermeasures is their high overhead compared to implementations hardened against timing side-channels. For our comparison, we outline the costs for masking ChaCha20 and AES.

For ARX ciphers such as ChaCha20, there are two possible approaches. First, it is possible to mask the addition operation with low overhead with arithmetic masking and apply Boolean masking to linear operation. The drawback of this approach is a costly conversion between the different types of masks [90]. Second, a Boolean masking can be applied to all operations. Then, instead of having an expensive mask conversion step, masking of the addition itself becomes expensive [91]. Currently, the most efficient approach for Cortex-M3/M4 ARM microcontrollers needs 78 instructions for one addition [92]. Therefore, it is necessary to spend at least $16 \times 21 \times 78 = 26\,208$

instructions to mask all addition operations of ChaCha20.

For AES, Boolean masking is usually implemented in a very different way. Schwabe et al. [93] proposed a bit-sliced approach to implement the SubBytes layer, which needs 688 instructions for each SubBytes layer. Since AES has 10 rounds, we only need to spend 6 880 instructions to implement the SubBytes layer of AES. A similar bit-slicing optimization is less beneficial for ChaCha20, because the operations used in the algorithm presented in [92] are already working with 32 bit in parallel. Therefore, the hardware utilization cannot be improved significantly by reorganizing the internal state.

Of course, we also need additional instructions for the linear operations of both ciphers. However, With the exception of added loads and stores, the amount of instructions only grows linearly with the protection order. Therefore, AES seems to be much faster, when protections against power side-channels are needed. Currently, the most efficient first-order protected implementation of AES-128 and ChaCha20 known in the literature take 463.9 [93] and 947.2 [92] clock cycles per byte, respectively, if long messages are processed. However, for short messages, such as those typically transmitted over the CAN bus, ChaCha20 is much worse than AES, since the minimum block size is 512 bit, while AES processes only 128 bit blocks, hence a significant slowdown is expected.

4.5. Conclusions

We compare the AES S-box and the modular addition used by ChaCha20, the main nonlinear components of two cryptographic algorithms regarding their side-channel vulnerability. While the addition operation is slightly harder to attack as compared to the S-box, the overhead for protecting an addition is much higher. Thus, if power side-channel attacks are not a concern, like in remote applications, ChaCha20 may have some advantages over AES owing to its software performance and natural resistance to timing attacks. However, if resistance against power side-channel attacks is required, the currently best known implementations favor AES.

5

Multi-Variate High-Order Attacks of Shuffled Tables Recomputation

Masking schemes based on tables recomputation are classical countermeasures against high-order side-channel attacks. Still, they are known to be attackable at order d in the case the masking involves d shares. In this work, we mathematically show that an attack of order strictly greater than d can be more successful than an attack at order d. To do so, we leverage the idea presented by Tunstall, Whitnall and Oswald at FSE 2013: we exhibit attacks which exploit the multiple leakages linked to one mask during the recomputation of tables. Specifically, regarding first-order table recomputation, improved by a shuffled execution, we show that there is a window of opportunity, in terms of noise variance, where a novel highly multivariate third-order attack is more efficient than a classical bivariate second-order attack. Moreover, we show on the example of the high-order secure table computation presented by Coron at EUROCRYPT 2014 that the window of opportunity enlarges linearly with the security order d. These results extend that of the CHES '15 eponymous paper. Here, we also investigate the case of degree one leakage models, and formally show that the Hamming weight model is the less favorable to the attacker. Eventually, we validate our attack on a real ATMEL smartcard.

5.1. Introduction

For more than 16 years now Side-Channel Attacks (SCA [94]) have been a threat against cryptographic algorithms in embedded systems. To protect cryptographic implementations against these attacks several countermeasures have been developed. Data masking schemes [95] are widely used since their security can be formally grounded.

The rationale of masking schemes goes as follows: each sensitive variable is randomly splitted in d shares (using $d-1$ masks), in such a way that any tuple of $d-1$ shares manipulated during the masked algorithm is independent from any sensitive variable. Masking schemes are the target of higher-order SCA [96–99]. A dth-order attack combines the leakages of d shares. In the implementation of masking schemes, it is particularly challenging to compute non-linear parts of the algorithm, such as for example the S-Box of AES (a function from n bits to n bits). To solve this difficulty different methods have been proposed which can be classified in three categories [100].

5

- Algebraic methods [101, 102]. The outputs of the S-Box will be computed using the algebraic representation of the S-Box.
- Global Look-up Table [103, 104] method. A table is precomputed offline for each possible input and output masks.
- Table recomputation methods which precompute a masked S-Box stored in a table [96, 105, 106]. Here, the full table is recomputed despite not all entries will be called. Such tables can be recomputed only once per encryption to reach first-order security. More recently, Coron presented at EUROCRYPT 2014 [107] a table recomputation scheme secure against dth-order attacks. Since this countermeasure aims at high-order security ($d > 1$), it requires one full table precomputation before every S-Box call.

These methods provide security against Differential Power Analysis [108] (DPA) or Higher-Order DPA (HODPA). Still, whatever the protection order, there is *at least one* leakage associated to each share; in practice, shares (typically masks) can leak *more than once*. For example attacks exploiting the multiplicity of leakages of the same mask during the table recomputation have been presented by Pan et al. in [109] and more recently by Tunstall et al. in [110]. Such attacks consist in guessing the mask in a first order horizontal Correlation Power Analysis [111, 112] (CPA) and then conducting a first-order vertical CPA knowing the mask. We refer to these attacks as Horizontal-Vertical attacks (HV attacks).

Shuffling the table recomputation makes the HV attacks more difficult. Still shuffling can be bypassed if the random permutation is generated from a seed with low

entropy, since both the mask and the shuffling seed can be guessed [110].

Our contributions.

Our first contribution is to describe a new HODPA tailored to target the table recomputation despite a highly entropic masking (unexploitable by exhaustive search). More precisely, we propose an innovative combination function, which has the specificity to be highly multivariate. We relate attacks based on the combination function of state-of-the-art and our new HODPA attack to their success rate, which allows for a straightforward comparison.

We build a theoretical analysis of their success rate. Our analysis reveals that there is a window of opportunity, when the noise variance is smaller than a threshold, where our new HODPA is more successful than a straightforward HODPA, despite it is of higher-order. Specifically, our analysis allows to derive mathematically that the previously known attacks require up to three times more traces than our new attack to extract the key. In addition, the impact of the leakage functions (Hamming weight, weighted sum of bits, etc.) is identified, and as a consequence the best and the worst cases for our new attack are found.

For instance in this paper we attack a first-order masking scheme based on table recomputation with a $(2^{n+1}+1)$-variate third-order attack more efficiently than with a classical bivariate second-order attack. In this case HV attacks could not be applied. This is the first time that a non minimal order attack is proved better (in terms of success rate) than the attack of minimal order. Actually, this non intuitive result arises from a relevant selection of leaking samples — this question is seldom addressed in the side-channel literature. We generalize our attack to a higher-order masking scheme based on tables recomputation (Coron, EUROCRYPT 2014), and prove that it remains better than a classical attack, with a window of opportunity that actually grows linearly with the masking order d.

Finally we propose a new innovative countermeasure in order to protect masking schemes based on tables recomputation against our new attack.

Outline of the paper.

The rest of the paper is organized as follows. Sect. 5.2 introduces the notations used in this article. Sect. 5.3 provides a reminder on table recomputation algorithms and on the way to defeat and protect this algorithm using random permutations. In Sect. 5.4 we propose a new attack against the "protected" implementation of the table recomputation, prove theoretically the soundness of the attack and validate these results by simulation. In Sect. 5.5 we apply this attack on a higher-order masking scheme. Sect. 5.6 extends our results to the case where the leakage function is affine in the bits of the targeted sensitive variable. In Sect. 5.7 we validate

our results on real traces. Finally in Sect. 9.2 we present a countermeasure to mitigate the impact of our new attack.

5.2. Preliminary and notations

In this article capital letters (e.g., U) denote random variables and lowercase letters denote their realizations (e.g., u).

Let $k^\star$ be the secret key of the cryptographic algorithm. T denotes the input or the ciphertext. We suppose that the computations are done on n-bit words which means that these words can be seen as elements of $\mathbb{F}_2^n$. As a consequence both $k^\star$ and T belong to $\mathbb{F}_2^n$. Moreover as we study protected implementations of cryptographic algorithms these algorithms also take as input a set of uniform independent random variables (not known by an attacker). Let denote by $\mathcal{R}$ this set.

5

Let g be a mapping which maps the input data to a *sensitive variable*. A *sensitive variable* is an internal variable proceeded by the cryptographic algorithm which depends on a subset of the inputs not known by the attacker (e.g. the secret key but also the secret random value). A measured leakage is modeled by:

$$X = \Psi\Big(g\big(k^\star, T, \mathcal{R}\big)\Big) + N, \tag{5.1}$$

where $\Psi : \mathbb{F}_2^n \rightarrow$ denotes the leakage function. This leakage function is a specific characteristic of the target device. The leakage function could be for example the Hamming Weight (denoted by HW in this article), or a weighted sum of bits (investigated in greater details in Sect. 5.6). The random variable N denotes an independent additive noise. In order to conduct a dth-order attack an attacker should combine the leakages of d shares. To combine these leakages an attacker will use a *combination function* [96, 113, 114]. The degree of this combination function must be at least d for the attack to succeed. The *combination function* will then be applied both on the measured leakages and on the model (this is the optimal HODPA). As a consequence, an HODPA is completely defined by the *combination function* used.

In the rest of the paper the is given by the following definition:

Definition 1 (Signal to noise ratio) *The Signal to Noise Ratio of a leakage denoted by a random variable L depending on informative part denoted I is given by:*

$$\mathrm{SNR}\left[L, I\right] = \frac{Var\left[\mathbb{E}\left[L|I\right]\right]}{\mathbb{E}\left[Var\left[L|I\right]\right]} \ . \tag{5.2}$$

An attack is said *sound* when it allows to recover the key $k^\star$ with success probability which tends to one when the number of measurements tends to the infinity.

5.3. Masking scheme with table recomputation

5.3.1. Algorithm

In this article we consider Boolean masking schemes. In particular, we focus on schemes based on table recomputation where the masked S-Box is stored in a table and fully recomputed each time.

This algorithm begins by a key addition phase where one word of the plaintext t, one word the key $k^\star$ and a random mask word m, are Xored together.

Then, these values are passed through a non linear function (stored in a table). The output of this operation can be masked by a different mask m'. Some linear operations can follow the non linear function. Of course, in the whole algorithm, all the data are masked (exclusive-ored) with a random mask, to ensure the protection against first order attacks.

Masking the linear parts is straightforward but passing through the non linear one is less obvious. To realize this operation the table is recomputed. For all the elements of $\frac{n}{2}$ the input mask is removed and then the output is masked by the output mask. In this step the key is never manipulated so all the leakages concern the mask. It can also be noticed that a new table S' of size $2^n \times n$ bits, is required for this step.

5.3.2. Classical attacks

As any masking scheme, table recomputation can be defeated without the leakage of the table recomputation. Indeed an attacker can use:

- Second order attacks [96, 97] such as second-order CPA (). It can be noticed that for such attacks, the adversary can also exploit the leakage of the mask during the table recomputation.
- Collisions attacks. If several S-Boxes are masked by the same mask the Collisions attacks may be practicable [115].

However these attacks do not take into account all the leakages due to table recomputation stage. An approach to exploit these leakages is to combine all of them with a leakage depending on the key. This method has been presented in [110] where an "horizontal" attack is performed on the table recomputation to recover the mask.

In such "horizontal" attacks two different steps can be targeted:

- An attacker could try to recover the output masks. In this case he should first recover the address in the table. In this case it is not necessary to

recover the input mask but only the address value.

– An attacker could also try to recover the input masks.

The second step consists in a vertical attack which recover the key. In this second step the mask is now a known value. It can be noticed that the exact knowledge of the mask is not required to recover the key. Indeed if the probability to recover the mask is higher than $\frac{1}{2^n}$ then a first order attack is possible (because the mask distribution is biased).

Recently, the optimal distinguisher in the case of masking has been studied in [116]: it is applied to the precomputation phase of masked table without shuffling in section 5. This attack can be extended to the case of shuffled table recomputation but would require an enumeration of all shuffles, which is computationally unfeasible.

5.3.3. Classical countermeasure

The strategy to protect the table recomputation against HV attacks and the distinguisher presented in [116] is to shuffle the recomputation, i.e., do the recomputation in a random order, as illustrated in Alg. 2.

Different methods to randomize the order are presented in [110]. One of the methods presented is based on a random permutation on a subset of $\frac{n}{2}$.

Let S_{2^n} the symmetric group of 2^n elements, which represents all the ways to shuffle the set $\{0,\ldots,2^n-1\}$. If the random permutation over $\frac{n}{2}$ is randomly drawn from a set of permutation $S \subset S_{2^n}$, where $card(S) \ll card(S_{2^n})$, it is still possible for an attacker to take advantage of the table recomputation. Indeed as it is shown in [110] attacks could be built by including all the possible permutations alongside with the key hypothesis. If the permutation is drawn uniformly over the S_{2^n} the number of added hypothesis is $2^n!$ which can be too much for attacks. For instance, for $n = 8$, we have $2^8! \approx 2^{1684}$.

By generating a highly entropic permutation, such as defined in [110] or any pseudo random permutation generator (RC4 key scheduler...), a designer could protect table recomputation against HV attacks. Indeed using for example five or six bytes of entropy as seed for the permutation generator could be enough to prevent an attacker from guessing all the possible permutations.

5.4. Totally random permutation and attack

In this section we present a new attack against shuffled table recomputation. The success of this attack will not be impacted by the entropy used to generate the shuffle. As a consequence this attack will succeed when the HV attacks will fail because

Algorithm 2: Shuffled masked table recomputation

input : Genuine SubBytes $S : {}_2^n \to {}_2^n$ bijection
output: Masked SubBytes $S' : {}_2^n \to {}_2^n$ bijection

1 $m \leftarrow_{\mathcal{R}} {}_2^n$, $m' \leftarrow_{\mathcal{R}} {}_2^n$ // Draw of random input and output masks
2 $\phi \leftarrow_{\mathcal{R}} {}_2^n \to {}_2^n$ // Draw of random permutation of ${}_2^n$
3 **for** $\omega \in \{0, 1, \ldots, 2^n - 1\}$ **do** // S-Box recomputation loop
4 $\quad z \leftarrow \phi(\omega) \oplus m$ // Masked input
5 $\quad z' \leftarrow S[\phi(\omega)] \oplus m'$ // Masked output
6 $\quad S'[z] = z'$ // Creating the masked S-Box entry
7 **end**
8 **return** S'

the quantity of entropy used to generate the shuffle is too large to be exhaustively enumerated. We then express the condition where this attack will outperform the state of the art second order attack.

5

5.4.1. Defeating the countermeasure

As the permutation ϕ is completely random, the value of the current index in the **for** loop (line 3 to line 7) is unknown. But it can be noticed that this current index $\boldsymbol{\phi(\omega)}$, printed in boldface for clarity, is manipulated twice at each step of the loop (line 4, line 5):

$$z \leftarrow \boldsymbol{\phi(\omega)} \oplus m \ , \tag{5.3}$$

$$z' \leftarrow S[\boldsymbol{\phi(\omega)}] \oplus m' \ . \tag{5.4}$$

Let U a random variable uniformly drawn over ${}_2^n$ and $m \in {}_2^n$ a constant. Then, it is shown in [99] that:

$$\mathbb{E}\left[(\mathsf{HW}[U] - \mathbb{E}\left[\mathsf{HW}[U]\right]) \times (\mathsf{HW}[U \oplus m] - \mathbb{E}\left[\mathsf{HW}[U \oplus m]\right])\right] = -\frac{\mathsf{HW}[m]}{2} + \frac{n}{4} \ . \tag{5.5}$$

As a consequence, it may be possible for an attacker to exploit the leakage depending on the two manipulations (Eq. (5.3) and (5.4)) of the current random index in the loop. Indeed, at each of the 2^n steps of the loop in the table recomputation, the leakage of the $\boldsymbol{\phi(\omega)}$ in Eq. (5.3) and (5.4) which plays the role of U in Eq. (5.5) will be combined (by a centered product) to recover a variable depending on the mask. Afterwards, these 2^n variables will be combined together (by a sum) in order to increase the SNR as much as possible. Finally, this sum is combined (again by a centered product) with a leakage depending on the key. This rough idea of

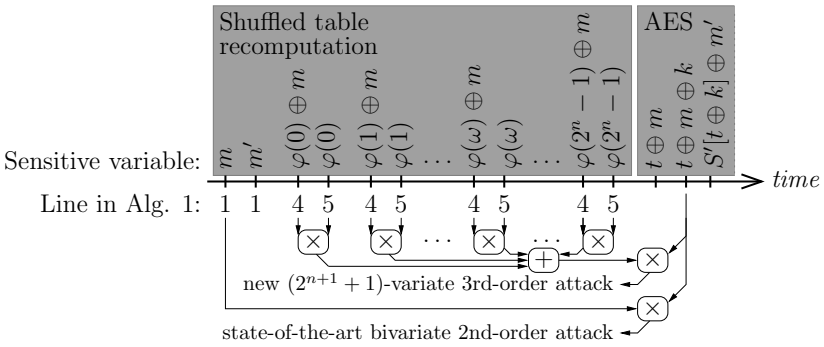


Figure 5.1: State-of-the-art attack and new attack investigated in this article

the attack is illustrated it on Fig. 5.1, which represents the "*trace*" corresponding to the *dynamic execution* of Alg. 2, followed by the masked AES AddRoundKey & SubBytes steps.

Remark 1 (Construction of the high-order attack) *The construction of the attack depicted in Fig. 5.1 leverages on two building blocks:*

1. *the centered product, represented as* $\otimes$*, which allows to get rid of a mask (recall Eq.* (5.5)*), albeit at the expense of a smaller SNR (it is squared, as shown in [117] – see Sec. 5.4.3)*
2. *the sum of variables with the same leakage model, represented as* $\boxplus$*, which increases the SNR linearly with the number of variables summed together.*

An attacker could want to perform the attack on the output of the S-Box. But depending on the implementation of the masking scheme the output masks can be different for each address of the S-Box (see for example the masking scheme of Coron [107]). To avoid loss of generality we focus our study on the S-Box input mask of the recomputation. Indeed by design of the table recomputation masking scheme, the input mask is the same for each address of the S-Box: the attacker can thus exploit it multiple times. Moreover an attacker can still take advantage of the confusion of the S-Box [118] to better discriminate the various key candidates. Indeed he can target the input the of SubBytes operation of the last round. Notice the use of capital M and capital Φ, which indicates that the leakage is modeled as a random variable.

5.4.2. Multivariate attacks against table recomputation

In the previous section, it has been shown that at each iteration of the loop of the table recomputation, it is possible to extract a value depending on the mask. As

a consequence it is possible to use all of these values to perform a multivariate attack. In this subsection we give the formal formula of this new attack. Let us define the leakages of the table recomputation. The leakage of the masked random index in the loop is given by: $\Phi(\omega) \oplus M + N_\omega^{(1)}$. The leakage of the random index is given by: $\Phi(\omega) + N_\omega^{(2)}$.

Depending on the knowledge about the model, the leakage could be centered by the "true" expectation or by the estimation of this expectation. We assume this expectation is a known value given by: $\mathbb{E}\ \mathsf{HW}\ \Phi\ \omega\ \oplus M\quad N_\omega^{(1)}\ =$ $\mathbb{E}\ \mathsf{HW}[\Phi(\omega)] + N_\omega^{(2)}\ = \frac{n}{2}$. Then let us denote the central leakages as:

$$X_\omega^{(1)} = \mathsf{HW}[\Phi(\omega) \oplus M] + N_\omega^{(1)} - \frac{n}{2}\ , \tag{5.6}$$

$$X_\omega^{(2)} = \mathsf{HW}[\Phi(\omega)] + N_\omega^{(2)} - \frac{n}{2}\ . \tag{5.7}$$

Besides, the leakage of the masked AddRoundKey is:

$$X^\star = T \oplus M \oplus k^\star + N - \frac{n}{2}\ . \tag{5.8}$$

In a view to use all the leakages of the table recomputation, an original combination function could be defined.

Definition 2 *The combination function exploiting the leakage of the table recomputation is given by:*

$$\begin{array}{rccc} \mathrm{C}_{TR}: & \mathbb{R}^{2^{n+1}} \times \mathbb{R} & \longrightarrow & \mathbb{R} \\ & \left(\left(X_\omega^{(1)}, X_\omega^{(2)}\right)_{0 \leqslant \omega \leqslant 2^n-1}, X^\star\right) & \longmapsto & \left(-2 \times \frac{1}{2^n} \sum_{\omega=0}^{2^n-1} X_\omega^{(1)} \times X_\omega^{(2)}\right) \times X^\star\ . \end{array}$$

Following the Fig. 5.1 it can be noticed that is in fact the combination of two sub-combination functions. Indeed, first of all, the leakages of the table recomputation are combined; the result of this combination is the following value:

$$X_{TR} = -2 \times \frac{1}{2^n} \sum_{\omega=0}^{2^n-1} X_\omega^{(1)} \times X_\omega^{(2)}\ . \tag{5.9}$$

Second, this value is multiplicatively combined with $X^\star$.

Remark 2 *It can be noticed that the random variable does not depend on* Φ. *Indeed in Eq.* (5.9) *the sum can be reordered by* Φ. *Moreover as this sum is computed over all the possible* $\Phi(\omega)$ *it implies that* $\frac{1}{2^n} \sum_{\omega=0}^{2^n-1} X_\omega^{(1)} \times X_\omega^{(2)}$ *is exactly the expectation over the*

$\Phi(\omega)$. *As a consequence is random only through the mask and the noise.*

Based on the combination function , a multivariate attack can be built.

Definition 3 *The MultiVariate Attack (MVA) exploiting the leakage of the table recomputation (TR) is given by the function:*

$$\begin{aligned}\mathrm{MVA}_{TR}: \quad & \mathbb{R}^{2^{n+1}} \times \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{F}_2^n \\ & \left(\left(X_\omega^{(1)}, X_\omega^{(2)}\right)_\omega, X^\star, Y\right) \longmapsto \underset{k\in\mathbb{F}_2^n}{\operatorname{argmax}} \rho\left[\mathrm{C}_{TR}\left(\left(X_\omega^{(1)}, X_\omega^{(2)}\right)_\omega, X^\star\right), Y\right],\end{aligned}$$

where $Y = \left(T \oplus M \oplus k - \frac{n}{2}\right) \cdot \left(M - \frac{n}{2}\right) | T$ *and* ρ *is the Pearson coefficient. According to Eq.* (5.5)*, the model* Y *is equal to an affine transformation of* $-T \oplus k$ *(note the negative sign for the correlation* ρ *extremal value when* $k \in_2^n$ *to be positive).*

is sound. By the law of large numbers, correlation coefficient involved in the expression of tends to $\rho(-T \oplus k^*, -T \oplus k)$ when the number of traces tends to infinity. This quantity is maximal when $k = k^*$, by the Cauchy-Schwarz theorem. Then for enough traces the noise will impact all the key guesses similarly and as a consequence the result of is maximal when $k = k^*$.

Remark 3 *The attack presented in Def. 3 is a* $(2^{n+1}+1)$*-multivariate third order attack.*

Let us denote the leakage of the mask (which occurs at line 1 of Alg. 2) by:

$$X^{(3)} = M + N^{(3)} - \frac{n}{2} \ . \tag{5.10}$$

Definition 4 *We denote by the using the centered product as combination function. Namely:*

$$\begin{aligned}\text{2O-CPA}: \quad & \mathbb{R} \times \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{F}_2^n \\ & \left(X^{(3)}, X^\star, Y\right) \longmapsto \underset{k\in\mathbb{F}_2^n}{\operatorname{argmax}} \rho\left[X^{(3)} \times X^\star, Y\right] \ .\end{aligned}$$

A careful look at Def. 2, Def. 3 and Eq. (5.9) reveals that the only difference between the and the is the use of instead of $X^{(3)}$. Thus will act as the leakage of the mask. Let us call the *second order leakage*. The informative part of the second order leakage is the same as the informative part of the leakage mask i.e.,

$$\mathbb{E}\left[X_{TR} | M = m\right] = \mathbb{E}\ \ X^{(3)} | M = m \quad .$$

It is a straightforward application of the results of [99]: Use Eq. (5.5) and notice the intentional -2 factor in Eq. (5.9). Both expectations are thus equal to m.

5.4.3. Leakage analysis

By using the formula of the theoretical success rate () we show that as the same operations are targeted by the and the . Consequently, it is equivalent to compare the or the of these attacks. Based on this fact we can theoretically establish the conditions in which the outperforms the . These conditions are given in Theorem 5.4.3.

Recently A.A Ding et al. [117, §3.4] give the following formula to establish the Success Rate () of second-order attacks:

$$\mathrm{SR} = \Phi_{N_k-1}\Big(\frac{\sqrt{b}\ \delta_0\delta_1}{4}K^{-1/2}\kappa\Big)\ . \tag{5.11}$$

In this formula:

- δ_0 denotes the of the first share and δ_1 denotes the of the second one;
- Φ_{N_k-1} denotes the cumulative distribution function of (N_k-1)-dimensional standard Gaussian distribution; as underlined by the authors in [117], if the noise distribution is not multi-variate Gaussian, then Φ_{N_k} is to be understood as its cumulative distribution function;
- N_k denotes the number of key candidates;
- K denotes the confusion matrix and κ the confusion coefficient;
- b denotes the number of traces.

Remark 4 *An updated version of this formula for* first order *has been presented in Eqn. (27) of [119] which solves the issue of the non invertible matrix.*

This formula allows to establish the link between the and of second order attacks against Boolean masking schemes.

Let us apply the A.A Ding et al. formula in the case of our two attacks:

$$\mathrm{SR}_{\text{2O-CPA}} = \Phi_{2^n-1}\Big(\sqrt{b}\,\frac{\mathrm{SNR}\left[X^{(3)},M\right]\mathrm{SNR}\left[X^{\star},(T,M)\right]}{4}K^{-1/2}\kappa\Big)\ ,$$
$$\mathrm{SR}_{\mathrm{MVA}_{TR}} = \Phi_{2^n-1}\Big(\sqrt{b}\,\frac{\mathrm{SNR}\left[X_{TR},M\right]\mathrm{SNR}\left[X^{\star},(T,M)\right]}{4}K^{-1/2}\kappa\Big)\ .$$

We target the same operation for the share that leaks the secret key ($X^{\star}$). Moreover by remark 5.4.2 the informative parts of the leakages depending on the mask (and $X^{(3)}$) is the same in the two leakages. As a consequence, K and κ are the same in the two attacks.

It can be noticed that the only difference in the success rate formula is the use of $,M$ instead of $X^{(3)},M$. Therefore, it is equivalent to compare these values and compare the of these attacks.

The of the "second-order leakage" is greater than the of the leakage of the mask if and only if

$$\sigma^2 \leqslant 2^{n-2} - \frac{n}{2} \ ,$$

where σ denotes the standard deviation of the Gaussian noise.

As a consequence will be better than in the interval $\sigma^2 \in [0, 2^{n-2} - n/2]$.

See Appendix .1. Interestingly, the same result is also a byproduct of the demonstration of Proposition 5.4.5 (see Appendix B).

Theorem 5.4.3 gives us the cases where exploiting the second-order leakage will give better results than exploiting the classical leakage of the mask. For example if $n = 8$ (the case of AES) the second-order leakage is better until $\sigma^2 \leqslant 60$.

Figure 5.2 shows when the of is greater than the of $X^{(3)}$. In order to have a better representation of this interval $1/$ is plotted.

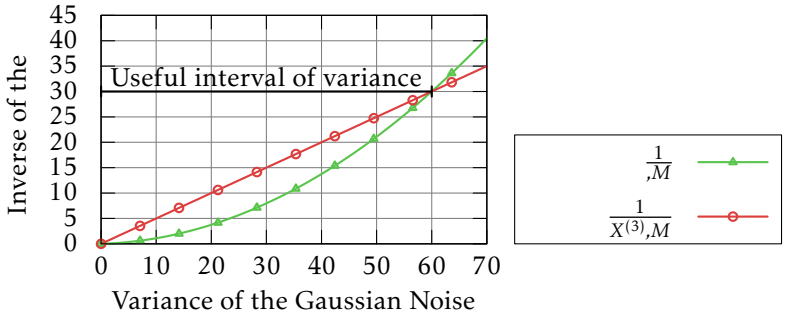


Figure 5.2: Comparison between the variance of the noise for the classical leakage and the second-order and the impact of these noises on the SNR

5.4.4. Simulation results

In order to validate empirically the results of Sect. 5.4, we test the method presented on simulated data. The target is a first order protected AES with table recomputation. To simulate the leakages we assume that each value leaks its Hamming weight with a Gaussian noise of standard deviation σ. The 512 leakages of the table recomputation are those given in Subsect. 5.4.2.

A total of 1000 attacks are realized to compute the success rate of each experiment. In this part, the comparisons are done on the number of traces needed to reach 80%

of success.

It can be seen in Fig. 5.3 and in Fig. 5.4 that the difference between the two attacks is null for $\sigma = 0$ and $\sigma = 8$ (that is, $\sigma^2 = 64 \approx 60$). It confirms the bound of the interval shown in Fig. 5.2. This also confirms that comparing the is equivalent to comparing the .

It can be seen in Fig. 5.7 that in presence of noise the outperforms the . The highest difference between the and is reached when $\sigma = 3$. In this case, the needs 2500 traces to mount the attack while the needs 7500 traces. This represents a relative gain[1] of $\approx 200\%$. As shown in Fig. 5.6, the relative gain decreases to 122% when $\sigma = 4$.

5.4.5. Theoretical analysis of the Success Rate

While the previous analysis of Subsect. 5.4.3 gives the bounds of effectiveness of the it does not allow a quantitative comparison of the respective behaviors of the and the between these bounds. In this subsection we propose an approach which allows a deeper analysis of the relevant parameters of their . We exploit the results of [120] which presents a closed form formula which links the to the for first order attacks. These results have recently been extended to high order attacks [121].

[[120 **([?, Corollary 1]).** *The* SR *of an* additive distinguisher *satisfies:*

$$1 - \mathrm{SR} \approx \exp\left(-\mathrm{SE} \times q\right) \ , \tag{5.12}$$

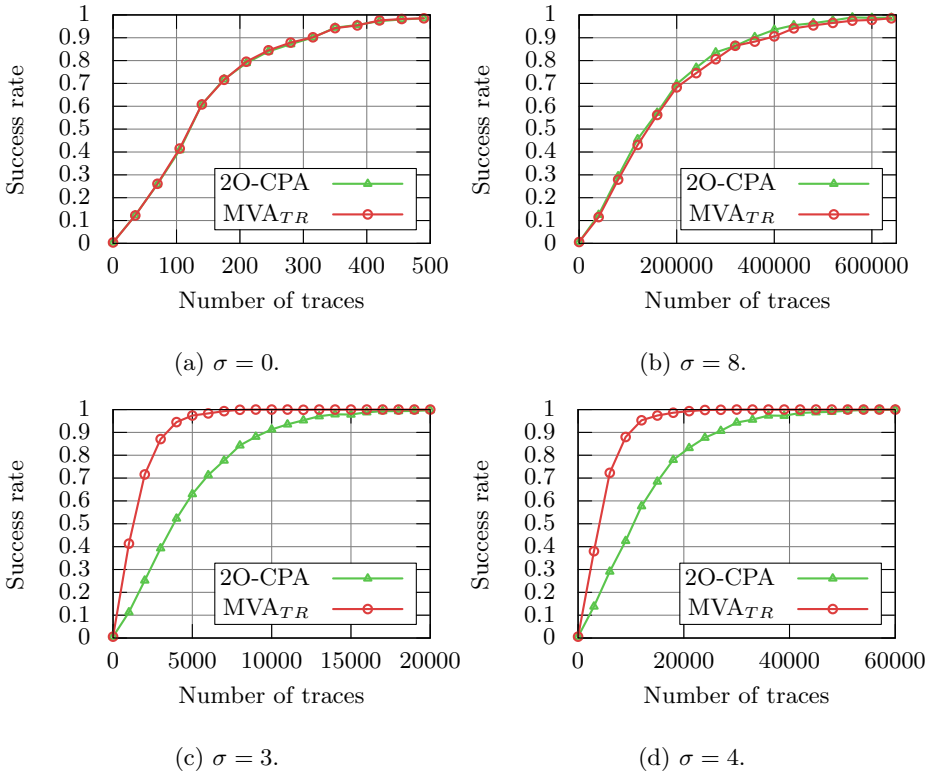
where is the success exponent and q the number of traces used for the attack.

The proof is given in [120].

Proposition 5. *The* SE *of the* 2O-CPA *is:*

$$\mathrm{SE}_{\text{2O-CPA}} = \min_{k \neq k^\star} \frac{\kappa\left(k^\star, k\right)}{2\left(\frac{\kappa'(k^\star,k)}{\kappa(k^\star,k)} - \kappa\left(k^\star, k\right)\right) + 2\left(\alpha_1^{-2}\sigma_1^2 + \alpha_2^{-2}\sigma_2^2 + \alpha_1^{-2}\sigma_1^2\alpha_2^{-2}\sigma_2^2\right)}, \tag{5.13}$$

[1] The formal definition of the relative gain is given in Def. 5.



(a) $\sigma = 0$.

(b) $\sigma = 8$.

(c) $\sigma = 3$.

(d) $\sigma = 4$.

Fig. 3: Comparison between 2O-CPA and MVA$_{TR}$

[5] The formal definition of the relative gain is given in Def. 5.

where in our case (which complies to Eqn. (5.2) of Definition 1):

$$\alpha_1^2 = \alpha_2^2 = Var\left[\mathbb{E}\left[X^{(3)}|M\right]\right] = Var[\mathbb{E}[X^\star|M,T]] = \sqrt{\frac{n}{4}} \ ,$$
$$\sigma_1^2 = \sigma_2^2 = \mathbb{E}\left[Var\left[X^{(3)}|M\right]\right] = \mathbb{E}[Var[X^\star|M,T]] = \sigma^2 \ ,$$

$\kappa\left(k^\star,k\right)$ and $\kappa'\left(k^\star,k\right)$ are general confusion coefficients defined in Definition 8 of [120]. Notice that $\kappa\left(k^\star,k\right)$ is a natural extension of the seminal coefficient introduced by Fei et al. in [118].

See Appendix A.

We note that α_i^2 and σ_i^2 respectively represent the power of the signal and of the noise.

5

As Def. 2, Def. 3 and Eq. (5.9) reveals that the only difference between the and the is the use of instead of $X^{(3)}$. Thus we can directly compute the success exponent of .

Proposition 6. *The* SE *of the* MVA_{TR} *is:*

$$\text{SE}_{\text{MVA}_{TR}} = \min_{k\neq k^\star} \frac{\kappa\left(k^\star,k\right)}{2\left(\frac{\kappa'(k^\star,k)}{\kappa(k^\star,k)} - \kappa\left(k^\star,k\right)\right) + 2\left(\alpha_1^{-2}\sigma_1^2 + \alpha_2^{-2}\sigma_2^2 + \alpha_1^{-2}\sigma_1^2\alpha_2^{-2}\sigma_2^2\right)} \ , \quad (5.14)$$

where in our case

$$\alpha_1^2 = \alpha_2^2 = Var[\mathbb{E}[X_{TR}|M]] = Var[\mathbb{E}[X^\star|M,T]] = \sqrt{\frac{n}{4}} \ ,$$
$$\sigma_1^2 = \mathbb{E}[Var[X_{TR}|M]] = 4 \times \left(\frac{\sigma^2}{2^n} \times \frac{n}{2} + \frac{\sigma^4}{2^n}\right) \ ,$$
$$\sigma_2^2 = \mathbb{E}[Var[X^\star|M,T]] = \sigma^2 \ .$$

The proof is similar as the proof of Prop. 5.4.5 using the values of noise computed in the Appendix .1.

Exploiting this values it is possible to extract the parameters which impact the respective behavior of the the two attacks and especially the ones reaching to a higher difference between the two attacks. Similarly to Subsect. 5.4.4 we will compare the two attacks using the relative gain.

Definition 5 (rel-gain$^{(\mathrm{SR})}$) *The relative gain between* 2O-CPA *and* MVA$_{TR}$ *is given by:*

$$\text{rel-gain}^{(\mathrm{SR})} = \frac{m^{(\mathrm{SR})}_{\text{2O-CPA}} - m^{(\mathrm{SR})}_{\text{MVA}_{TR}}}{m^{(\mathrm{SR})}_{\text{MVA}_{TR}}} \ ,$$

where $m^{(\mathrm{SR})}_{\text{2O-CPA}}$ *and* $m^{(\mathrm{SR})}_{\text{MVA}_{TR}}$ *are respectively the number of traces needed by* 2O-CPA *and* MVA$_{TR}$ *to reach success rate value* SR.

And we will also used the difference in number of traces needed to reach SR.

Definition 6 (gain$^{(\mathrm{SR})}$) *The difference in number of traces needed to reach* SR *of success is given by the gain:*

$$\text{gain}^{(\mathrm{SR})} = m^{(\mathrm{SR})}_{\text{2O-CPA}} - m^{(\mathrm{SR})}_{\text{MVA}_{TR}} \ ,$$

where $m^{(\mathrm{SR})}_{\text{2O-CPA}}$ *and* $m^{(\mathrm{SR})}_{\text{MVA}_{TR}}$ *are respectively the number of traces needed by* 2O-CPA *and* MVA$_{TR}$ *to reach* SR *of success rate.*

5

Notice that rel-gain$^{(\mathrm{SR})}$ and gain$^{(\mathrm{SR})}$ are tools to compare attacks after having computed their SR. They differ from *relative distinguishing margins* metrics [**?**] which analyses the value of the distinguisher (and not their SR).

Proposition 7. rel-gain$^{(\mathrm{SR})}$ *does not depend on the value of* SR.

See Appendix B.

This means that, in Fig. 5.7, the curves for and are the same, modulo a scaling in the X axis. For instance, in Fig. 5.7(a) and (b), the scaling factor is 1, i.e., the two curves superimpose perfectly. As a result, one can compare these two attacks in terms of traces number to extract the key, irrespective of the value chosen for the threshold.

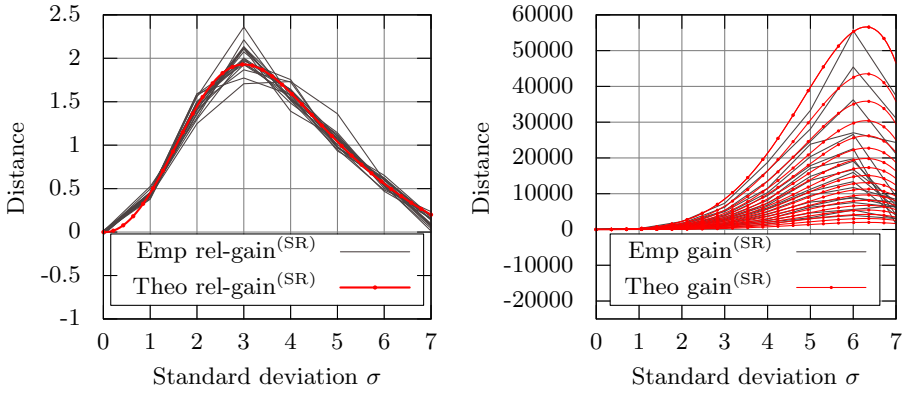
Proposition 8. gain$^{(\mathrm{SR})}$ *depends on the value of* SR, *but the value of the noise variance where* gain$^{(\mathrm{SR})}$ *is maximum not depends on* SR.

See Appendix C.

Remark 5 *While the bounds of Theorem 3 depend only on the* SNR *the maximum effectiveness (the maximum of* gain$^{(\mathrm{SR})}$ *or* rel-gain$^{(\mathrm{SR})}$*) of the* MVA$_{TR}$ *compare to the* 2O-CPA *also depends on the operation targets (e.g.* AddRoundKey *or* SubBytes*) by the confusion coefficients* κ *and* κ'.

Numerical Results.

In order to validate our theoretical analysis we build empirical validation based on simulations. We reuse the curves generated for Sect. 5.4.4. In Fig. 5.10 the empirical results based simulation are plotted in gray and the Theoretical ones in red pointed lines. The first observation is that the theoretical analysis match well the simulations which validates our model choices.



(a) rel-gain$^{(SR)}$ (b) gain$^{(SR)}$

Fig. 4: Comparison between the 2O-CPA and the MVA$_{TR}$

In Fig5.8 it can be noticed that for several SR (different gray lines) the empirical rel-gain$^{(SR)}$ are closed which confirmed the Prop. 7. Exploiting the formula of Def. 5 we can find the noise variance σ^2 where rel-gain$^{(SR)}$ is maximum. Indeed it occurs in a root of the derivative of rel-gain$^{(SR)}$. In our scenario it occurs for $\sigma^2 = 9.11$ (that is $\sigma \approx 3.02$). For this value of σ^2, the relative gain is about equal to 2, that is, **our** MVA$_{TR}$ **attacks requires three times less traces than the** 2O-CPA **to extract the key**.

The behavior of gain$^{(SR)}$ is different indeed the SR has an impact on it, the gray lines are not superimposed (see Fig.5.9). But similarly to rel-gain$^{(SR)}$ the SR does not impact the value of noise where the maximum gain$^{(SR)}$ is reached. This confirms the Prop. 8. In our scenario it is reached for $\sigma^2 = 39.67$ (that is $\sigma \approx 6.30$).

In order to compute this maximum we have computed the roots of the derivatives (of rel-gain$^{(SR)}$ and gain$^{(SR)}$ w.r.t. σ^2) using the MAXIMA software.

5.5. An example on a high-order countermeasure

The result of the previous section can be extended to any masking scheme based on table recomputation. In particular the can apply to High-Order masking schemes.

5.5.1. Coron masking scheme attack and countermeasure

The table recomputation countermeasure can be made secure against High-Order attacks. An approach has been proposed by Schramm and Paar [123]. However, it happened that this masking scheme can be defeated by a third order attack [124]. To avoid this vulnerability Coron recently presented [107] a new method based on table recomputation, which guarantees a truly high-order masking. The core idea of this method is to mask each output of the S-Box with a different mask and refresh the set of masks between each shift of the table (masking the inputs by one mask). HV attacks are still a threat against such schemes. Indeed an attacker will recover iteratively each input mask. Afterwards he will be able to perform a first order attack on the AddRoundKey to recover the key. To prevent attacks based on the exploitation of the leakages of the input masks an approach based on a random shuffling of the loop index is possible (see Alg. 3). Algorithm 3 is a $(d-1)$-th order countermeasure, meaning that attacks of order strictly less than d fail. In this algorithm, the x_i for $i < d$ can be seen indifferently as *shares* or as *masks*. The original masked S-Box algorithm from Coron [107] is the same as Alg. 3, with ϕ chosen as the identity. It can be noticed that the entropy needed to build the permutation could be low compared to the entropy needed for the masking scheme (especially because of the numerous costly RefreshMasks operations).

Algorithm 3: Masked and shuffled computation of $y = S(x)$

input : $x_1, \ldots, x_d$, such that $x = x_1 \oplus \ldots \oplus x_d$
output: $y_1, \ldots, y_d$, such that $y = y_1 \oplus \ldots \oplus y_d = S(x)$

$\phi \leftarrow_{\mathcal{R}} {}^{n}_{2} \rightarrow^{n}_{2}$ // Draw of random permutation of ${}^{n}_{2}$
for $\omega \in^{n}_{2}$ **do**
  $\mathcal{T}(\omega) \leftarrow (S(\omega), 0, \ldots, 0) \in \binom{n}{2}^{d}$ // $\oplus(\mathcal{T}(\omega)) = S(\omega)$
end
for $i = 1$ **to** $d - 1$ **do**
  for $\omega \in^{n}_{2}$ **do**
    for $j = 1$ **to** d **do**
      $\mathcal{T}'(\phi(\omega))[j] \leftarrow \mathcal{T}(\phi(\omega) \oplus x_i)[j]$ // $\mathcal{T}'(\phi(\omega)) \leftarrow \mathcal{T}(\phi(\omega) \oplus x_i)$
    end
  end
  for $\omega \in^{n}_{2}$ **do**
    $\mathcal{T}(\phi(\omega)) \leftarrow (\mathcal{T}'(\phi(\omega)))$ // See in Alg. 2 of [107]
  end
end
// Invariant: $\oplus(\mathcal{T}(\phi(\omega))) = S(\phi(\omega) \oplus x_1 \oplus \ldots \oplus x_{d-1}),\ \forall \omega \in^{n}_{2}$
$(y_1, \ldots, y_d) \leftarrow (\mathcal{T}(x_d))$ // $\oplus(\mathcal{T}(x_d)) = S(x)$
return $y_1, \ldots, y_d$

5.5.2. Attack on the countermeasure

We apply Alg. 2 on X which is equal to $T \oplus k^\star$, i.e., $\bigoplus_{i=1}^{d} X_i = T \oplus k^\star$. Similarly to the definitions in Subsect. 4.2, let us define the leakages of the table recomputation of the masking scheme of Coron where the order of the masking is $d-1$: $X^{(1)}_{(\omega,i,j)} = \mathsf{HW}[\Phi(\omega) \oplus X_i] + N^{(1)}_{(\omega,i,j)} - \frac{n}{2}$ and $X^{(2)}_{(\omega,i,j)} = \mathsf{HW}[\Phi(\omega)] + N^{(2)}_{(\omega,i,j)} - \frac{n}{2}$, where $i \in [\![1, d-1]\!]$ will index the $d-1$ masks. The d-th share is the masked sensitive value. Besides $j \in [\![1, d]\!]$ denotes the index of the loop from lines 7 to lines 9 of the Alg. 2. The leakage of the masks is given by $X^{(3)}_i = \mathsf{HW}[X_i] + N^{(3)}_i - \frac{n}{2}$. Finally, we denote by: $X^\star = \mathsf{HW}[\bigoplus_{i=1}^{d-1} X_i \oplus k^\star \oplus T] + N - \frac{n}{2}$ the leakage of the masked value.

Definition 7 *The combination function C^d_{CS} exploiting the leakage of the table recomputation (Coron Scheme, abridged CS) is given by:*

$$\mathrm{C}^d_{CS}: \mathbb{R}^{d\times(d-1)\times 2^{n+1}} \times \mathbb{R} \to \mathbb{R}$$
$$\left(\left(X^{(1)}_{(\omega,i,j)}, X^{(2)}_{(\omega,i,j)}\right)_{\substack{\omega\in\mathbb{F}_{2^n}\\ i\in[\![1,d-1]\!]\\ j\in[\![1,d]\!]}}, X^\star\right) \mapsto \prod_{i=1}^{d-1}\left(\frac{-2}{d2^n}\sum_{\substack{\omega\in\mathbb{F}_{2^n}\\ j\in[\![1,d]\!]}} X^{(1)}_{(\omega,i,j)} \times X^{(2)}_{(\omega,i,j)}\right) \times X^\star.$$

Similarly to Subsect. 4.3, we define for all $1 \leqslant i \leqslant d-1$:

$$X_{CS^d_i} = \frac{-2}{d2^n}\sum_{\substack{\omega\in\mathbb{F}_{2^n}\\ j\in[\![1,d]\!]}} X^{(1)}_{(\omega,i,j)} \times X^{(2)}_{(\omega,i,j)} \ .$$

This value is the combination of all the leaking values of the table recomputation depending of one share.

Remark 6 *The scaling by factor $-2/d$ allows to have, for all $i \in [\![1, d-1]\!]$:*

$$\mathbb{E}\left[X_{CS^d_i} | X_i = x_i\right] = \mathbb{E}\left[X^{(3)}_i | X_i = x_i\right] \ .$$

Additionally we define for, $i = d$, $X_{CS^d_i} = X^\star$. Based on the combination function C^d_{CS} a multivariate attack can be built.

Definition 8 *The MultiVariate Attack exploiting the leakage of the table recomputation of the $d-1$ order Coron masking Scheme is given by:*

$$\mathrm{MVA}_{CS}^{d}: \quad \mathbb{R}^{d\times(d-1)\times 2^{n+1}} \times \mathbb{R} \times \mathbb{R} \to \mathbb{F}_2^n$$
$$\left(\left(X^{(1)}_{(\omega,i,j)}, X^{(2)}_{(\omega,i,j)}\right)_{\substack{\omega\in\mathbb{F}_{2^n}\\ i\in[\![1,d-1]\!]\\ j\in[\![1,d]\!]}}, X^\star, Y\right) \mapsto \underset{k\in\mathbb{F}_2^n}{\operatorname{argmax}}\, \rho\left[\prod_{i=1}^{d}\left(X_{CS_i^d}\right), Y\right] ,$$

where $Y = (-1)^{d-1} \times \left(\mathsf{HW}[T\oplus k] - \frac{n}{2}\right)$.

is sound. The demonstration follows the same lines as that of Proposition 5.4.2. In the case of Proposition 5.5.2, the expectation of $\prod_{i=1}^{d}(d)$ knowing the plaintext $T=t$ is proportional to $t\oplus k$. Indeed by [125] $\prod_{i=1}^{d}(d)|T=t = \left(\frac{-1}{2}\right)^{d-1} \times \left(t\oplus k - \frac{n}{2}\right)$

Remark 7 *The attack presented in Def. 8 is a $(d\times(d-1)\times 2^{n+1}+1)$-variate $(2\times(d-1)+1)$-order attack.*

Definition 9 *The "classical"* dO-CPA *is the HOCPA build by combining the d shares using the centered product combination function.*

$$\text{dO-CPA}: \quad \mathbb{R}^{d-1} \times \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{F}_2^n$$
$$\left(\left(X_i^{(3)}\right)_{i\in[\![1,d-1]\!]}, X^\star, Y\right) \longmapsto \underset{k\in\mathbb{F}_2^n}{\operatorname{argmax}}\, \rho\left[\prod_{i=1}^{d-1} X_i^{(3)} \times X^\star, Y\right] .$$

5.5.3. Leakage analysis

The difference between the two attacks is the use of d instead of $X_i^{(3)}$ as the leakage of the $d-1$ shares which do not leak the secret key. A.A Ding et al. also provides a formula to compute the of HOCPA [117, §3.4].

Similarly to Sect. 4, the only differences in the formula are the SNR of the shares which do not leak the key. Then by comparing the SNR $\left[X_{CS_i^d}, X_i\right]$ and SNR $\left[X_i^{(3)}, X_i\right]$ we compare the success rate of the attacks. It can be noticed that in our model the SNR does not depend on i.

Theorem 10. *The* SNR *of the "second-order leakage" is greater than the* SNR *of the leakage of the mask if and only if*

$$\sigma^2 \leqslant d\times 2^{n-2} - \frac{n}{2} , \tag{5.15}$$

where σ denotes the standard deviation of the Gaussian noise.

As a consequence MVA_{CS}^{d} *will be better than* dO-CPA *when the noise variance lays in the interval* $[0, d\times 2^{n-2} - n/2]$. *We can immediately deduce that the size of the Useful Interval of Variance increases linearly with the order of the masking scheme.*

Proof. See Appendix .3.

Figure 5 shows the impact of the attack order d on the interval of noise where the MVA_{CS}^{d} outperforms dO-CPA (let us called this interval the Useful Interval of Variance denoted by UIoV). We can see that the size of these intervals increases with the order. For example for $d = 3$ the useful interval of variance is $[0, 188]$. In practice, it is very difficult to perform a third order attack with a noise variance of 188. Indeed, recall that the number of traces to succeed an attack with probability 80% is proportional to the inverse of the SNR [120].

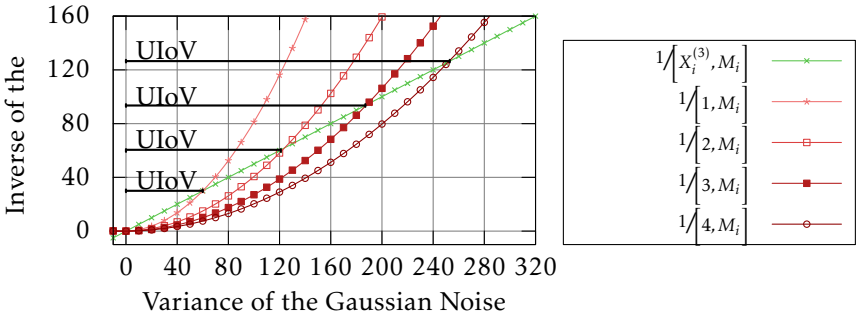


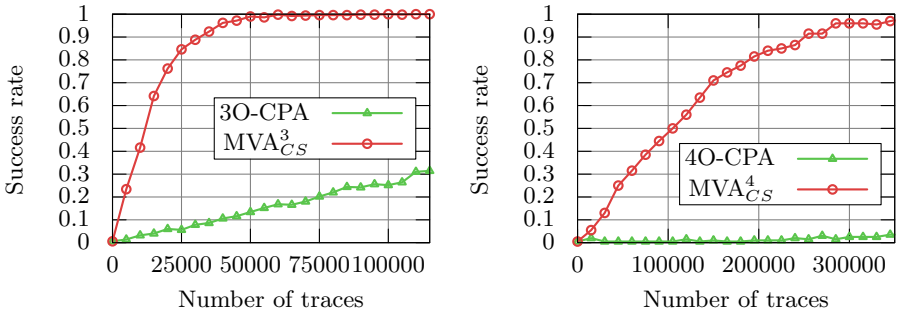
Figure 5.11: Comparison between the signal to noise ratio of $X_i^{(3)}$ and signal to noise ratio of d (where d is the attack order)

5.5.4. Simulation results on Coron masking Scheme

In order to validate the theoretical results of Subsect. 5.5.3, the has been tested on simulated data and compared to . The simulations have been done with the Hamming weight model and Gaussian noise such as the leakages defined in Subsect. 5.5.2. We test these attacks against a second and a third order masking scheme.

To compute the success rate, attacks are redone 500 times for the second order masking and 100 times for the third order masking (because this attack requires an intensive computational power).

In Fig. **??** it can be seen that $\mathrm{MVA}_{CS}^{(3)}$ reaches 80% of success rate for less than 20000 traces while the 3O-CPA does not reach 30% for 100000. In Fig. **??** it can be seen that $\mathrm{MVA}_{CS}^{(4)}$ reaches 80% of success rate for less than 200000 traces while the 4O-CPA does not reach 5%.



(a) $d = 3, \sigma = 3$. (b) $d = 4, \sigma = 3$.

Figure 5.12: Comparison between dO-CPA and MVA^d_{CS}

5.6. A note on affine model

In Sect. 5.4 and 5.5, the leakage function was expected to be the Hamming weight. Let us now study the impact of the leakage function on the MVA_{TR} attack. We suppose that the leakage function is affine.

5.6.1. Properties of the affine model

Definition 10 (Affine leakage function) *Let V the leaking value, α the weight of the leakage of each bit, and $\cdot$ the inner product in ^{n}, that is $\alpha \cdot V = \sum_{i=1}^n \alpha_i V_i$. A leakage function Ψ_α is said affine if this function is a weighted sum of the bits of the leaking value, i.e., $\Psi_\alpha(V) = \alpha \cdot V$.*

In the sequel, we assume sensitive variables are balanced and have each bit independent of the other, as is customary in cryptographic applications.

Proposition 11. *Let* $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}_2^n$.

$$\mathbb{E}[\Psi_\alpha(V)] = \frac{1}{2}(\alpha \cdot \mathbf{1}) \qquad \textit{and} \qquad \textit{Var}[\Psi_\alpha(V)] = \frac{1}{4}\|\alpha\|_2^2 \ .$$

Proof. We have $\mathbb{E}[\Psi_\alpha(V)] = \alpha \cdot \mathbb{E}[V] = \alpha \cdot \left(\frac{1}{2}\mathbf{1}\right)$ and $\text{Var}[\Psi_\alpha(V)] = \alpha^{\mathsf{t}}\mathsf{Cov}[V]\,\alpha = \frac{1}{4}\|\alpha\|_2^2$. □

Then it is possible to compute the results of the centered product.

Lemma 12. *Let U be a random variable following a uniform law over $\mathbb{F}_2^n$, and $z \in \mathbb{F}_2^n$. We have:*

$$\mathbb{E}\left[\left(\Psi_\alpha(U) - \mathbb{E}\left[\Psi_\alpha(U)\right]\right) \times \left(\Psi_\beta(U \oplus z) - \mathbb{E}\left[\Psi_\beta(U \oplus z)\right]\right)\right] = -\frac{1}{2}(\alpha \odot \beta)\cdot z + \frac{1}{4}\alpha\cdot\beta \ ,$$

where $\odot$ denotes the element-wise multiplication, that is $(\alpha \odot \beta)_i = \alpha_i\beta_i$.

See in Appendix A.

Assumption 1 *In order to compare the results in case of an affine model and the Hamming weight model ($\mathsf{HW} = \Psi_{\mathbf{1}}$) let us assume that the model variance is the same in the two cases i.e., $Var[\Psi_\alpha(V)] = Var[\mathsf{HW}[V]]$; this is equivalent to $\|\alpha\|_2^2 = n$.*

Let us also assume that all the values manipulated during the algorithm leak in the same way i.e., the weight vector α of the sum is the same for all the variables V of the algorithm. This is realistic because it is likely that sensitive variables transit through a given resource, e.g., the accumulator register.

In the rest of this section, we will denote by α the vector of weight of the leakage model.

Let us redefine the leakage of the table recomputation the (centered) leakage of the random index: $X_\omega^{(1)} = \alpha \cdot (\Phi(\omega) \oplus M) + N_\omega^{(1)} - \frac{1}{2}(\alpha \cdot \mathbf{1})$, the (centered) leakage of the mask random index: $X_\omega^{(2)} = \alpha \cdot (\Phi(\omega)) + N_\omega^{(2)} - \frac{1}{2}(\alpha \cdot \mathbf{1})$, the (centered) leakage of the mask: $X^{(3)} = \alpha \cdot M - \frac{1}{2}(\alpha \cdot \mathbf{1})$, Besides, let $X^\star$ be the leakage of a sensitive value depending on the key. We have either:

- $X^\star = \alpha \cdot (T \oplus k^\star \oplus M) + N - \frac{1}{2}(\alpha \cdot \mathbf{1})$, which is similar to Eq. (5.8), or
- $X^\star = \alpha \cdot (S(T \oplus k^\star) \oplus M) + N - \frac{1}{2}(\alpha \cdot \mathbf{1})$, if there is an S-Box S.

In a view to unite both expressions, we denote by Z the sensitive variable, that is either $Z = T \oplus k^\star$, or $Z = S(T \oplus k^\star)$. Consequently, we have $X^\star = \alpha \cdot (Z \oplus M) + N - \frac{1}{2}(\alpha \cdot \mathbf{1})$.

Lemma 13. *In case of affine leakage model the second order leakage X_{TR} is given by:*

$$\mathbb{E}\left[X_{TR} | M = m\right] = \mathbb{E}\left[\frac{-2}{2^n}\sum_{\omega=0}^{2^n-1} X_\omega^{(1)} \times X_\omega^{(2)} \mid M = m\right] = \left(\alpha^2\right) \cdot m - \frac{1}{2}\|\alpha\|_2^2 \ ,$$

where $\alpha^2 = \alpha \odot \alpha$.

Direct application of Lemma 5.6.1.

In case of affine model, the leakages of the (recall Def. 2) and the 2O-CPA are different. Indeed, let us denote $\alpha^n = \underbrace{\alpha \odot \alpha \odot \cdots \odot \alpha}_{n \text{ times}}$. We have:

We have:

$$\mathbb{E}\left[\mathrm{C}_{TR}\left(\left(X_\omega^{(1)}, X_\omega^{(2)}\right)_\omega, X^\star\right) \mid T\right] = -\frac{1}{2}\alpha^3 \cdot z + \frac{1}{4}\sum_{i=1}^{n}\alpha_i^3 \ ,$$

and

$$\mathbb{E}\left[X^{(3)} \times X^\star \mid T\right] = -\frac{1}{2}\alpha^2 \cdot z + \frac{1}{4}\|\alpha\|_2^2 \ .$$

Direct application of Lemma 5.6.1 and Lemma 5.6.1.

5.6.2. Impact of the model on the confusion coefficient

As the models in the two different attacks are different, the parameters K and κ (recall Eq. (5.11)) also differ. In order to compare the two attacks we first establish the impact of the model on the value of the minimum confusion coefficient $\min_{k \neq 0} \kappa_k$. Then we show that the impact is not important in case of the targeted sensitive value is proceed in a nonlinear part of the algorithm (an S-Box).

In practice the confusion coefficients are very close. We study the impact of the disparity of α using several distributions (see Fig. 5.13):

- $\alpha_i = \sqrt{1+\epsilon}$ for i even and $\alpha_i = \sqrt{1-\epsilon}$ otherwise (abridged $\alpha = \sqrt{1 \pm \epsilon}$),
- and the other sign convention (abridged $\alpha = \sqrt{1 \mp \epsilon}$).

We also randomly generate 1000 α. All those distributions satisfy the assumption 1, namely $\sum_{i=1}^{n} \alpha_i^2 = n$.

The confusion coefficient for α^2 and α^3 are very close (see Fig. 5.13).

Moreover we find that the maximum difference in all the simulations with random weight is $\max(\min_{k\neq 0} \alpha^2 \kappa_k - \min_{k\neq 0} \alpha^3 \kappa_k) = 0.019$. In terms of number of traces needed to reach 80% of success this represents a small difference of 5%.

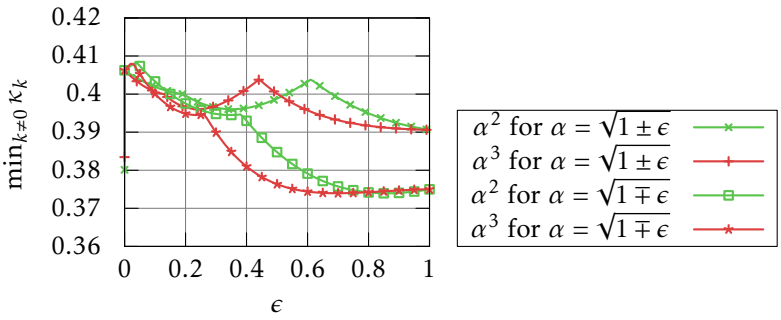


Figure 5.13: Comparison of $\min_{k\neq 0} \kappa_k$ for the and the

5.6.3. Theoretical analysis

Similarly to the Subsect. 5.4.3 let us study the impact of the affine model on the success of the compared to the .

As motivated in Sect. 5.4.1, we can modify the in order to target the last round S-Box input: $X^\star = \alpha \cdot \Big(T \oplus k^\star \oplus M\Big) + N - \frac{1}{2}(\alpha \cdot \mathbf{1})$.

The of the "second-order leakage" is greater than the of the leakage of the mask if and only if

$$\sigma^2 \leqslant 4\alpha^4 \times \frac{2^{n-2}}{n} - \frac{n}{2} \ ,$$

where $p\alpha = (\sum_{i=1}^{n} |\alpha_i|^p)^{1/p}$ is the p-norm ($p \geqslant 1$) of vector α, and where σ denotes the standard deviation of the Gaussian noise.

As a consequence is better than when the noise variance is in the interval $[0, 4\alpha^4 2^{n-2}/n - n/2]$. See Appendix B.

The minimal value of $4\alpha^4$ subject to $2\alpha^2 = n$ is reached when all the component of α are equal. This means that the worst case for the compared to the is when the leakage is in Hamming Weight.

See Appendix C.

5.6.4. Simulation results

Some simulations have been done in order to validate the results of the theoretical study of the previous sections. The results, presented in this section, confirm that:

- attacks are not impacted by the small differences of the confusion coefficient (κ, recall Sec. 5.6.2).
- attacks depend on the as predicted by Theorem 5.6.3.

For the purpose of the simulations, the target considered is the input of the S-Box of the last round; as a consequence we consider

$$X^\star = \alpha \cdot \left(T \oplus k^\star \oplus M\right) + N - \frac{1}{2}(\alpha \cdot \mathbf{1}) \ .$$

The mask M and the plain text T are randomly drawn from $\frac{8}{2}$. The noises are drawn from a Gaussian distribution with different variances σ^2. The results of the attacks are expressed using the success rate. To compute the success rates the experiments have been redone 1000 times. For each experiment the secret key $k^\star$ are randomly drawn over $\frac{8}{2}$. To compare the efficiency of the two attacks we compare the number of traces needed to reach 80% of success.

For the first experiment we choose $\alpha = \sqrt{1 \pm \epsilon}$ (i.e., $\forall i$, $\alpha_i = \sqrt{1 + (-1)^i \epsilon}$).

Case $\epsilon = 0.9$

In this case $4\alpha^4 = 14.480$ and according to Theorem 5.6.3, the should outperform the classical success rate in the interval $[0, 111]$. It can be seen in Fig. 5.14 and 5.15 that in such case when $\sigma^2 = 0$ or when $\sigma^2 = 111$ the and the need the same number of traces to reach 80% of success. First of all, this confirms the soundness of our model. Second, it validates that, in case of affine model when the target is proceeded in a non linear part of the cryptographic algorithm, the main factor which makes attacks different is the . When $\sigma = 3$ the needs around 3800 traces to reach 80% of success whereas the needs around 1000 traces (see Fig. 5.16). This represents a relative gain of 280%. Compared to the relative gain observed in case of the Hamming weight model (recall Fig. 5.5), this confirms that the performs better compare to the in case of an affine model. It can be seen in Fig. 5.17, when

the $\sigma = 4$, the number of traces needed to reach 80% of success is around 2500 for the and around 10000 for the ; this represents a relative gain of 300%.

Case $\epsilon = 0.5$

When $\epsilon = 0.5$, $4\alpha^4 = 10$; consequently, Theorem 5.6.3 predicts that the should outperform in the interval $[0,76]$. It can be seen in Figure 5.19 and 5.20 that in such case when $\sigma^2 = 0$ or when $\sigma^2 = 76$ the and the need the same number of traces to reach 80% of is success. This confirms the results of Theorem 5.6.3.

It can be seen in Fig. 5.21 that when $\sigma = 3$ the needs around 1000 traces to reach 80% of success whereas the needs 3500 traces. The relative gain of use the is 250%. When $\sigma = 4$ then the number traces needed by the to reach 80% of success is around 3000. The number of traces needed by the is around 9000. The relative gain of the with respect to the is 200%.

5

For one bit attacks

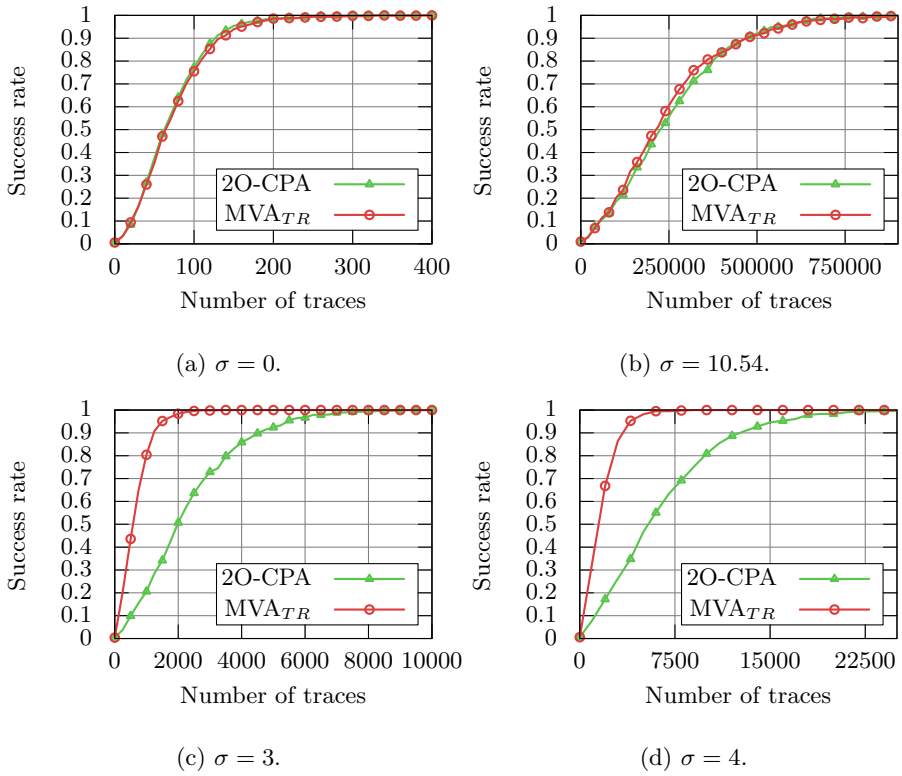
The best case for compared to the is when all the bits are zero except one (see Appendix C). Let us compare the two attacks in a such case. We assume that all the coordinates of α are equal to zero except the most significant bit. As $4\alpha^4 = 64$ the Useful Interval of Variance is $[0,508]$. It can bee see in Fig.5.24 that when the noise is null both attacks perform in the same way. It confirms that also in this case the difference resides in the . When $\sigma = 8$ the reach 80% of success with 25000 traces whereas the needs 175000; this represents a relative gain of 600% (see Fig. 5.25).

5.7. Practical validation

This section presents the results of the multivariate attack exploiting the table recomputation stage on true traces.

5.7.1. Experimental Setup

The traces are electromagnetic leakages of the execution of an AES-128 assembly implementation with table recomputation. Our implementation has been loaded on ATMEL ATMega163 8-bit to be analyzed. This smartcard is known to be leaky. It contains 16Kb of in-system programmable flash, 512 bytes of EEPROM, 1Kb of internal SRAM and 32 general purpose working registers. The smartcard is controlled by a computer through the Xilinx Spartan-VI FPGA embedded in a SASEBO-W platform. The ATMega is powered at 2.5 V and clocked at 3.57 MHz.



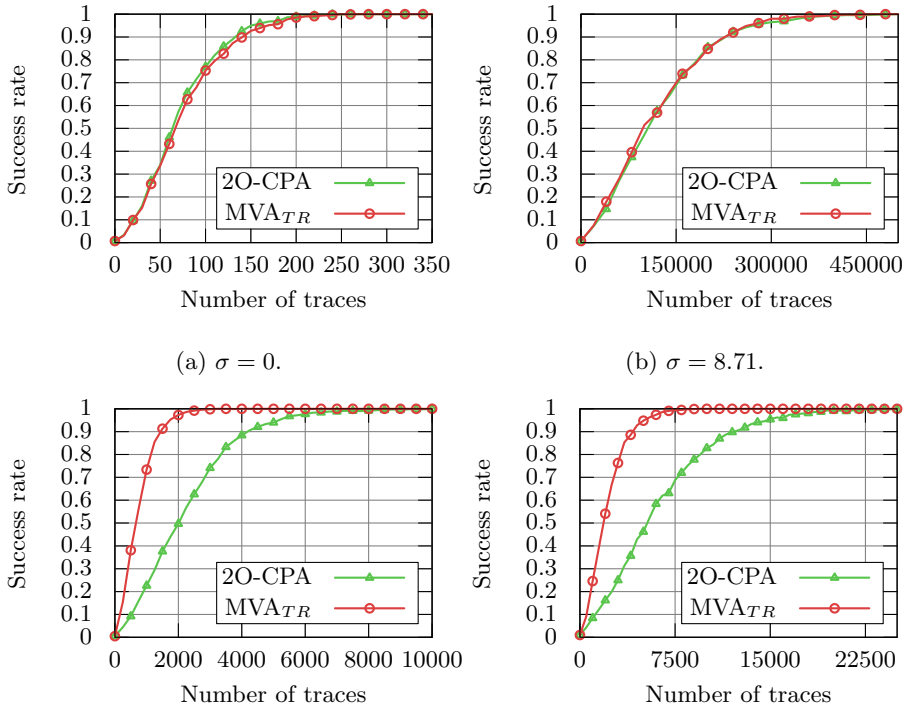
(a) $\sigma = 0$.

(b) $\sigma = 10.54$.

(c) $\sigma = 3$.

(d) $\sigma = 4$.

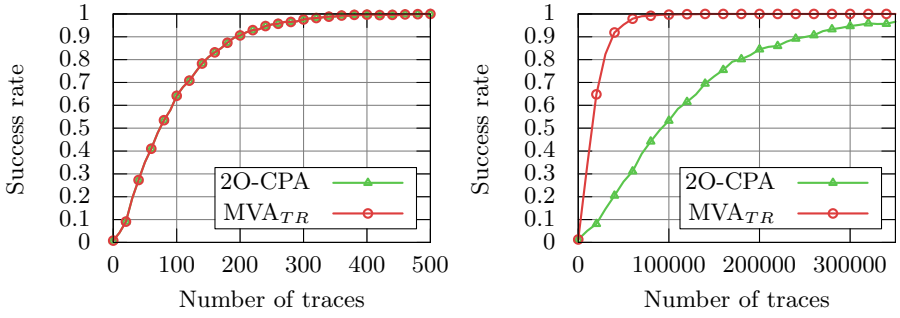
Fig. 8: Comparison between 2O-CPA and MVA$_{TR}$ for $\varepsilon = 0.9$



(a) $\sigma = 0$. (b) $\sigma = 8.71$.

(c) $\sigma = 3$. (d) $\sigma = 4$.

Fig. 9: Comparison between 2O-CPA and MVA$_{TR}$ for $\varepsilon = 0.5$



(a) $\sigma = 0$. (b) $\sigma = 8$.

Fig. 10: Comparison between the 2O-CPA and the MVA$_{TR}$ in case of one bit model in presence of high Gaussian noise

The measurements were taken using a LeCroy wave-runner 6100A oscilloscope by means of a Langer EMV 0–3 GHz EM probe and PA-303 30 dB Langer amplifier. The acquisitions have been acquired with full bandwidth and with a sampling rate of F_S = 500 MS/s.

To build our experiments 13000 traces have been acquired. Each trace contains 12 million leakages samples in order to simplify our analysis we only acquired the table recomputation step and the first round of the AES.

5.7.2. Experimental results

Let us first study the results of the attack in terms of success rate. The leakage function as been recovered using a linear regression. For example the normalized vector of weight for the leakage of the first share is

$$\alpha = (0.95, 1.22, 0.98, 1.13, 0.59, 1.01, 1.04, 0.95) .$$

Both the and the target $T \oplus k^\star \oplus M$ as in our implementation the input and output masks are the same.

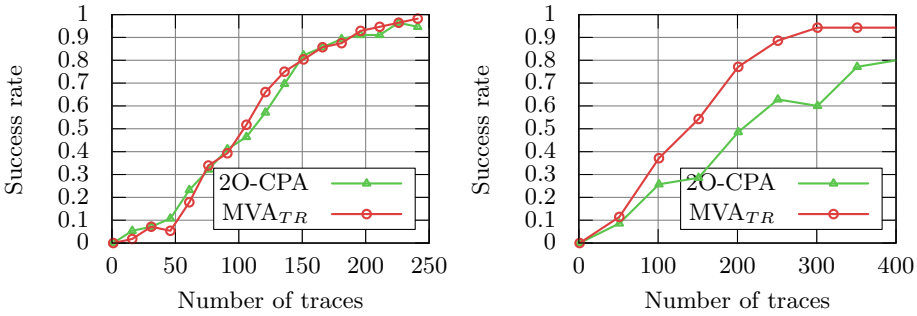
It can be seen in Fig. 5.27 that the results of the two attacks are similar. Both attacks perform similarly because the curves are not noisy.

Indeed the average values of the of the 256 leakages of the masked random index $(\Phi(\omega) \oplus M)$ and the of the 256 leakages of the random index $(\Phi(\omega))$ is 5.

If we assume that the variance of the signal is equal to two (such as HW on 8-bit CPUs) then the variance of the noise is less than 0.5. The mask (M) and the key-dependent share ($T \oplus k^\star \oplus M$) leak with a of 14 which corresponds to a noise variance of 0.1, which is very low (compared to the upper bound of the useful interval of variance given in Theorem 5.4.3, namely 60).

This two results are specific to the implementation and a clear disadvantage for the . But even in this case the works as well as the , this shows that there is (generally) a gain to use the .

In order to confirm these results let us verify that when the noise increases the outperforms the . Let us add an artificial Gaussian noise with a standard deviation of 0.0040. This models the addition of a countermeasure on top of the table recomputation. Then it can be seen in Fig. 5.28 that in this case the outperforms the . This confirms the practicality of our attack, and also that the gain is in the .



(a) Comparison on raw traces (b) Comparison with noise addition

Fig. 11: Comparison of the SR of the MVA$_{TR}$ and the 2O-CPA

5.8. Countermeasure

The represents a threat against block ciphers with table recomputation step. In order to mitigate this new vulnerability we present in this section a countermeasure, depicted in Alg. 4. This countermeasure will ensure the security against the new proposed attack. We present it in the context of a first order masking scheme but this countermeasure is generic and as a consequence can be applied in a higher order masking scheme such as the masking scheme of Coron.

Remark 8 *The proposed countermeasure tackles the input masks vulnerability. The protection of the output mask is easier as all the output masks can be different for all the table entries.*

5.8.1. Countermeasure Principle

The core idea of this countermeasure is to randomly draw permutations not all over the possible permutations but only over a particular kind of permutations: the ones which are commutative with S (the SubBytes function).

Definition 11 *A permutation $f : {}_2^n \to {}_2^n$ is said to be* commutative *with respect to the function $g : {}_2^n \to {}_2^n$ and the composition law if and only if $f(g(x)) = g(f(x)), \forall x \in {}_2^n$.*

Exploiting this kind of function the countermeasure principle is as follow: as random permutation, a commutative permutation with respect to S is drawn. Let us call the permutation γ. Exploiting the commutative property of the random permutation, $\gamma(S[\omega])$ is computed instead of $S[\gamma(\omega)]$ (line 5 of Alg. 4). Contrast this line with line 5 of Alg. 2. As a consequence if an attacker combines the leakages of the random mask index (line 4) and the random index (line 5) the obtained value depends very little in the masks m and m' (see in-depth analysis in Sec. 5.8.3).

Algorithm 4: Shuffled masked table recomputation, with our additional countermeasure

input : Genuine SubBytes $S : {}_2^n \to {}_2^n$ bijection
output: Masked SubBytes $S' : {}_2^n \to {}_2^n$ bijection

1 $m \leftarrow_{\mathcal{R}} {}_2^n$, $m' \leftarrow_{\mathcal{R}} {}_2^n$ // Draw of random input and output masks
2 $\phi \leftarrow_{\mathcal{R}} {}_2^n \to {}_2^n$ // Draw of random permutation of ${}_2^n$, permuting with S
3 **for** $\omega \in \{0, 1, \ldots, 2^n - 1\}$ **do** // S-Box recomputation loop
4 $z \leftarrow \phi(\omega) \oplus m$ // Masked input
5 $z' \leftarrow \phi(S[\omega]) \oplus m'$ // Masked output
6 $S'[z] = z'$ // Creating the masked S-Box entry
7 **end**
8 **return** S'

5.8.2. Implementations

The major issue of the countermeasure in an implementation perspective is to randomly generate a commutative permutation.

A first approach could be to generate *off-line* a large enough set of permutations and store them into the device. At each execution using a random number, a permutation will be selected. Of course such approach can be prohibitive in terms of memory need and as a consequence is not applicable.

A probably better approach is to generate *on-the-fly* a commutative permutation. In this subsection we give an example of a such algorithm. The idea is to randomly generate a power (with respect to the combination law) of the SubBytes : S bijection.

Definition 12 *The power $p \in \mathbb{N}$ of the function S is given by:*

$$\begin{array}{rccc} S^p : & \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ & x & \longmapsto & \underbrace{S \circ S \circ \ldots \circ S}_{p}(x), \end{array}$$

where $\circ$ denotes the composition law.

The bijections $S^p : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ and $S : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ are commutative $\forall p \in \mathbb{N}$.

In order to generate a random power of S it is possible to directly compute S^r by applying r times the permutation S where r is a random number. Notice that r can be larger than the number of possible power S by the group law property of the combination. But this approach can be time consuming.

In a view to accelerate this operation, the use of the cycle decomposition of S may be an interesting approach. Let us recall this well known theorem: [Theorem 5.19 [126]] Let S_n be the symmetric group of n elements then each element of S_n can be expressed as a product of disjoint cycles.

The maximum number of exponentiations needed to compute S^p could be reduced from p to $p \pmod{l_1} + p \pmod{l_2} + \ldots + p \pmod{l_m}$ where the l_i denote the respective length of the cycles in the cycles decomposition of S. Notice that $l_1 + l_2 + \ldots + l_m = 2^n$.

We can express S as $S = c_1 \circ c_2 \circ \ldots \circ c_m$ by Prop. 5.8.2. As the order of a cycle is equal to its length l we have that:

$$S^p = c_1^{p \pmod{l_1}} \circ c_2^{p \pmod{l_2}} \ldots \circ c_m^{p \pmod{l_m}}.$$

Let us take as example of S the SubBytes function of AES. This permutation can be decomposed on five disjoint cycles of respectively length $l_1 = 59, l_2 = 81, l_3 = 87, l_4 = 27, l_5 = 2$. The order of S in this case is $\mathrm{lcm}(59, 81, 87, 27, 2) = 277182$. As a consequence the computation of S^{277182} requires a maximum of 256 table evaluations.

5.8.3. Security Analysis

The security provided by this countermeasure results from different working factors. Of course the first one is to ensure that the is still unfeasible or at least less effective than the which would remain feasible. We validated this security using simulation with the same setup as in Subsect. 5.4.4. Namely we assume that each value leaks its Hamming weight with a Gaussian noise of standard deviation σ. A total of 1000 attacks has been realized to compute the success rate of each experiment.

The attacker can combine *multiplicatively* $\gamma(S[\omega])$ with $\gamma(\omega)$. The results of the attack resulting from this combination can be found in Fig. 5.32 for two different noise standard deviations. We can immediately see that in this case the does not allow to recover the key.

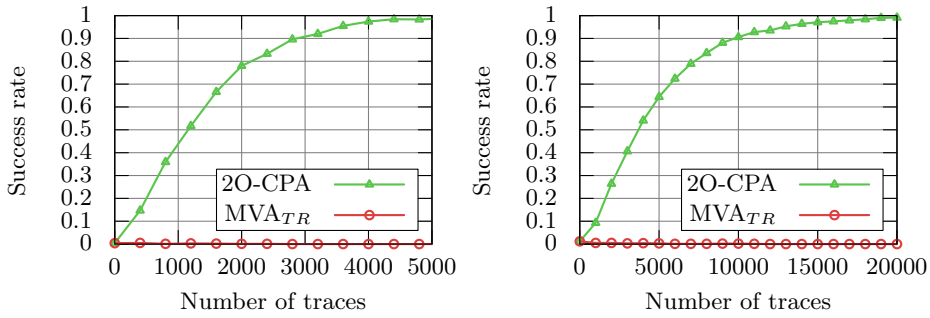
The second working factor for the countermeasure of Alg. 4 is the number of possible commutative permutations. Indeed if this number is too low an attacker can test all the permutations and build attacks such as in [110]. For example using the possible powers of S in AES, we reach a total count of 277182 bijections commutating with S, which is hard to exhaustively test but remains possible.

Of course another aspect of the countermeasure is the security of the permutation generation itself against possible Side Channel Analysis. If an attacker is able for example to recover: $p \pmod{l_1}, p \pmod{l_2}, \ldots, p \pmod{l_m}$, he will be able to recover the random permutation. This means that at least the exponentiation of S should be executed in constant time.

5.8.4. Implementation analysis

The countermeasure presented previously may have an impact both on the time and on the entropy needed for the table recomputation step. Interestingly the entropy i.e., the number of random bytes needed, is smaller in our new countermeasure. Indeed in the case where the non-linear operation is built using the S-box of AES our new countermeasure needs less than 5 bytes of entropy whereas in the case of shuffle implementation 256 bytes are needed[2].

[2] Of course this reduction may have an impact on the security especially in the case where an attacker performs an exhaustive search overall the permutations



(a) $\sigma = 2$. (b) $\sigma = 3$.

Fig. 12: MVA$_{TR}$ with commutative bijection as countermeasure (Alg. 3)

Implementation	Time
	0.810 μs
	0.861 μs
	23,700 μs (i.e., 23.7 ms)
	3.03 μs

Table 5.1: Time needed for the table recomputation

The other important implementation parameter is the execution time of the table recomputation. In order to evaluate it, we implement in C a classical table recomputation without any countermeasure denoted by , a shuffled version denoted by where the permutation is drawn over all the possible permutations. We also implement our countermeasure in a naïve approach denoted by and finally our countermeasure exploiting the cycle decomposition denoted by . The summary of the different times of table recomputation execution can be found in Table 5.1.

The profiling has been done using GPROF on an i5-6198DU CPU running at 2.30 GHz.

It can be first noticed that the naïve approach leads to a prohibitive overhead while the implementation using the cycle decomposition is computed in a reasonable supplementary amount of time. As a consequence we can deduce that this countermeasure can be an interesting alternative to avoid the attacks presented in this article. Finally the time needed to generate the random permutation is small. Indeed both the implementation with and without shuffle have almost the same execution time. Nevertheless these results may be slightly different on embedded systems where the random generation could be costly.

5.9. Conclusions and Perspectives

The table recomputation is a known weakness of masking schemes. We have recalled that practical countermeasures (e.g., shuffling with a high entropy) could be built to protect the table recomputation. In this article, we have presented a new multivariate attack exploiting the leakage of the protected table that outperformed classical HODPA even if a large amount of entropy is used to generate the countermeasure. This multivariate attack gives an example of a HODPA of non-minimal order which is more efficient than the corresponding minimal order HODPA. We have theoretically expressed the bound of noise in which this attack outperforms HOCPA using the . Then we have empirically validated this bound. Interestingly, we show that if the leakage model consists in a linear combination of bits, then our

attack becomes all the better as the model gets further away from uniform weights (so-called Hamming weight model). Moreover, we have shown that the relative gain to use the multivariate attack grows linearly with the order of the masking schemes. This result highlights the fact that the study of masking scheme should take into account as second parameter the number of variables exploitable by these attacks. Indeed we have shown in this article that when the number of variables used to perform the attacks increases, the *order* does not alone provide a criterion to evaluate the security of the countermeasure, and that the *SNR* is a better security metric to consider.

In future works we will investigate how to protect table recomputation against such attacks and investigate the cost of such countermeasures, evaluate the threat of such attacks on high-order masking schemes implemented on real components. We will also investigate how multivariate attacks could be applied on other masking schemes and protection techniques. And then, we will quantify the impact of these attacks.

Acknowledgments

The authors would like to thank Annelie Heuser for fruitful comments and interesting discussions. Besides, we acknowledge Prof. Olivier Rioul for guidance about the relationship between success rate and signal-to-noise ratio.

6

Feature Selection Methods for Non-Profiled Side-Channel Attacks on ECC

Elliptic curve cryptography (ECC) is a public key cryptosystem which is widely used for different real world applications. With the introduction of side-channel attacks, there is a growing concern regarding the security of such implementations. Indeed, side-channel attacks have been reported to break even the theoretically secure ciphers due to the exploit in the physical leakage. The non-profiled side-channel attacks, especially are considered more serious than the profiled counterpart, as the former can work in almost black box setting. Several attacks have been proposed, however, one of the main issue normally encountered is regarding the selection of relevant features from the side-channel signal. For ECC implementation, normally the side-channel measurements will contain lots of irrelevant points which could hinder the effectiveness of the attack. For profiling scenario, these features can be determined, since the attacker has full knowledge, however, for black box non-profiled setting, this might pose an issue. In this work, we investigate different feature selection approaches to improve the accuracy for non-profiled attacks on ECC. We demonstrate the effectiveness of proposed methods on real measurements from FPGA and microcontroller targets, achieving accuracy comparable to profiled case (88.6% and 98.4% respectively).

6.1. Introduction

Elliptic Curve Cryptography is based on the difficulty of computing the discrete logarithm problem over elliptic curves (ECDLP). The classical cryptanalysis mainly focuses on efficiency improvements to compute the ECDLP, but so far, there is no efficient classical algorithm to compute the ECDLP for large enough key sizes. On the other hand, in practice, other attacks are more relevant, for instances side-channel analysis (SCA) [127].

SCA normally exploits the physical properties of the implementation. When conducting SCA, the attacker observes one or few specific physical traits of the target implementation such as the time, power consumption or electromagnetic emanations (EM) and derives the key from these observations using statistical methods. For ECC, there are many proposed attacks, which exploit different properties of the algorithms [128].

For most of the attacks, the attacker needs to know the (public) input data, or has to perform a profiling step with known secret inputs to successfully derive a secret key. A simple approach to protect against many of such attacks is randomization based countermeasures. In contrast, unsupervised attacks have the potential to render most of these countermeasures ineffective, because they do not require any knowledge about the input data, neither the public or the private part. The attacks are applied on randomized scalars with minimum assumptions. The original scalar can be deterministically computed once randomized scalar is recovered.

As the measurement signal be performed in several milliseconds and contain millions of sample points, direct application of the attack technique is not optimal. The main challenge is to identify few relevant sample points or features in the measured signal which would lead to a successful attack. In the present paper, we explore different approaches for feature selection and compare these to a profiled attack. While success rate of profiled attack represents the best case, bit-wise processing of scalar brings random guess success to 50% (worst case). In this paper, we evaluate a pool of feature selection methods commonly used in side-channel testing, in context of unsupervised attacks.

The rest of the paper can then be organized as follow: In Section 6.2, we provide a brief description of the related work. Section 6.3, we provide the description of the proposed approach. Practical experiments are discussed in Section 6.4 and finally, in Section 6.5, we conclude the paper.

6.2. Related Work

The application of unsupervised clustering or feature selection algorithms is a relatively new topic. One of the first publications used the k-means clustering algorithm [129]. Instead of using a two step approach, feature selection first and then clustering the traces with a reduced set of features, they directly apply the k-means clustering on the set of traces. To improve the success rate, the authors use several low-noise localized EM measurements. An improved version of this attack applied PCA for dimensional reduction to improve the success rate further [130]. The main drawback of the general approach is, that it is not applicable in a high-noise environment.

The more recent work [131] proposes a two step approach. They basically follow the framework from [132] and adapt it to ECC. In principle, the procedure is iterative in nature. A preliminary leakage assessment step is used to identify a first approximate of the leaking samples in the trace. In a second step, the identified preliminary features are used to cluster the traces. Then, based on this clustering a refinement step is performed, which leads to better set of features. The final attack on the actual trace to attack is carried out using the refined set of features. This attack framework works very well on both ECC and RSA. The two papers report very high success rates in many cases for both ECC and RSA. The main target in [131] is only on key dependent processes. However, in general, there might be some noise attributed to other data dependent leakage which might not be interesting to the attacker. Hence, in this work, we consider an alternative approach which might help to mitigate this problem.

6.3. Methodology

In this section, we will briefly describe different statistics we propose to use for feature selection, as well as the classification method. Based on these methods, we then choose sufficient number of relevant features and conduct the attack.

6.3.1. Trace Characterization

Computations in an Elliptic Curve Scalar Multiplication (ECSM) operation are segmented and performed in a loop, where each segment is corresponding to one bit of the secret scalar. Since the implementations of the ECSM operation are recommended to be fairly regular across all key bits to avoid timing side channels and SPA attacks, it allows us to analyze trace segments corresponding to only one bit of the scalar (which we henceforth refer to as *traces* for brevity) across multiple scalars for feature selection. The resulting selected features thus apply to the

traces corresponding to other key bits as well. We acquire a set of traces $\mathcal{T}$ with n traces t^i for $i \in [0, n-1]$ and we denote a sample of length L with index j as t^i_j for $j \in \{0, L-1\}$. Each of the traces correspond to one of the two labels (0 or 1) equal to the corresponding secret scalar bit processed.

6.3.2. Feature selection

In a supervised attack scenario where the attacker knows the labels corresponding to the different traces in the trace set $\mathcal{T}$, the attacker can partition the traces into two sets $\mathcal{T}_0$ and $\mathcal{T}_1$ and apply the commonly used univariate Welch's t-test (also known as TVLA [133]) and select those samples as features that correspond to a high TVLA value. But, in an unsupervised scenario, where the attacker does not know the labels, the attacker can follow two approaches. The first approach is a label-dependent approach, wherein first he uses certain clustering algorithms such as the univariate k-means [134] over all the samples of the trace. Further, a cluster of two classes $c_{0,j}$ and $c_{1,j}$ for each sample j is obtained. The clusters are then evaluted using the following approaches.

- Difference-of-Mean (DoM) [94]: The difference of the means of the two clusters are calculated as $\text{DoM}_j = m_{0,j} - m_{1,j}$ and samples with high DoM are selected as features.
- Welch's t-test [135]: The technique is the same as that of the profiled case, except that the t-test is applied over the clusters with predicted labels ($P_{0,j}$ & $P_{1,j}$) instead of actual labels.
- Normalized Inter-Class Variance (NICV) [136]: NICV is another metric used for leakage detection similar to the TVLA (t-test) metric and a sample with high value of NICV can be classified as a feature.

While these methods require preliminary estimation of the labels for each sample, the attacker can also attempt to select features solely based on the distribution of the data. For this, we consider the following approaches:

- Variance: One can compute the variance of the samples for a fixed index j as Var_j and repeat the same for all samples and select those samples with high variance as features. This is based on the argument made by Clavier et al. [137] that samples corresponding to high activity due to manipulated data have a large variance compared to samples with low or constant activity independent of the algorithmic input.
- Range: Computing the range for each sample j as $\text{Range}_j = \max\limits_{1 \le i \le n} t^i_j - \min\limits_{1 \le i \le n} t^i_j$ across traces follows a similar rational as variance, but has the possibility of giving rise to many outliers.

The features that we are interested in are those that help distinguishing traces based on the value of the corresponding processed key bit, which can be alternatively termed as *key-dependent* features. But, it is important to know that there are also samples that depend on the value of the intermediate data processed, which can be termed as *data-dependent* features. While combining key-dependent features will increase the success probability of key-recovery, data-dependent features are simply random and add noise due to randomness of the intermediate data, which is independent of the processed key bit. Since all the above mentioned approaches look at each sample individually in a univariate manner, it is definitely possible that some of the irrelevant and random data-dependent samples are also selected as features.

On the assumption that, there are multiple key dependent leakage points in a single trace, we claim that all the key-dependent features observable across all traces in the trace-set are very highly correlated. For example, consider a set of P key-dependent features of a trace t processing bit k as $(f_1, f_2, \ldots, f_{P-1})$, the time samples of trace t corresponding to these features $(t_{f_1}, t_{f_2}, \ldots, t_{f_{P-1}})$ all will belong to the same class k of their corresponding univariate clusters $(c_{k,f_1}, c_{k,f_2}, \ldots, c_{k,f_{P-1}})$. But, the same cannot be said for a data dependent feature as it is very unlikely to find two highly correlated data-dependent leakage points on the same trace due to the randomness of the intermediate data. Thus, we propose to adopt a multi-variate approach wherein we use the covariance metric to identify key-dependent features at two different indices (positions) on the trace.

For all pairs of possible indices u, v with $u, v \in \{0, L-1\}$, we calculate the covariance of vectors t_u^i and t_v^i with i running from $\{0, n-1\}$, which we denote as $\mathrm{Cov}_{u,v}$. We thus build a corresponding covariance matrix for the trace set $\mathcal{T}$ of size $(n \times L)$. We identify those entries in the matrix which have a very high magnitude for covariance and consider the corresponding pairs to be highly correlated. A high positive value at entry (u, v) indicates that the position of the class k is the same in the corresponding univariate clusters c_u and c_v and thus $\mathrm{Sign}(\mathrm{DoM}_u) = \mathrm{Sign}(\mathrm{DoM}_v)$. A high negative value otherwise indicates that the position of class k is switched in the respective univariate clusters leading to opposing signs for their DoMs. But, it is a well known fact in side channel analysis that any point in the trace is always very highly correlated with itself and its neighboring points. Thus, barring the neighboring pairs of points, which are the entries near the diagonal of the covariance matrix, an entry with a high magnitude of covariance conveniently far from the diagonal can be considered to be key-dependent features with very high confidence. Unlike the previous works on unsupervised attacks on ECC [129–131], we are able to distinguish key dependent features from data dependent features which will subsequently increase the attack success rates.

6.3.3. Classification Phase

Once the key-dependent features have been identified, most of the previous works [129–131] propose to utilize the multi-dimensional k-means classification algorithm over the identified features for classification. It might also be possible that the number of selected features might be too high for the k-means algorithm, thus bumping into a clear case of the *curse of dimensionality*. We here propose a much simpler technique for classification which only requires calculation of the mean of a subset of the selected features. From the selected set of $\tilde{P}$ relevant features $\mathcal{F} = (f_0, f_1, f_2, f_{\tilde{P}-1})$ from the feature selection phase, we select only that subset of features which pairwise correlate to a very high positive value, through which we get a clear partition of the feature set resulting in disjoint subsets $\mathcal{F}_0$ and $\mathcal{F}_1$. The attacker can choose either of the partitions and calculate the mean of the samples at the respective features of the selected partition ($\mathcal{F}_0$ or $\mathcal{F}_1$), thereby assigning a score of s_i for the trace t_i for all $i \in \{0, 1\}$. Assuming we have approximately equal number of traces corresponding to both the key bits (which is easily maintained by the randomness of the key), we calculate the mean of all the assigned scores s_i to be the global threshold for classification. This leaves us with only two possible values of the key depending on position of the class in the resulting cluster.

6.4. Experiments

In this section, we describe the implementation details of the targetted ECSM operation, the devices used, the evaluation setup and the corresponding experimental results. We target two different ECSM implementations, one on a hardware and the other on a software target to show the applicability of our attack.

6.4.1. Hardware Implementation and Evaluation Setup

It is a custom straightforward hardware design of an ECSM operation on a twisted Edwards curve Ed25519 [138] implemented using the Joye's double and add ladder [139] and extended projective coordinates [140]. The design is implemented on Virtex-5 FPGA on the standard side-channel evaluation board SASEBO-GII. The access pattern of the registers directly depends on the key bit used and thus, the difference in address of the registers used manifests as key-dependent leakage. The implementation is constant-time and devoid of conditional branches to protect against timing side-channel attacks. The implementation run at 24 MHz and measurements were performed using a Lecroy 610Zi oscilloscope synchronised with a suitable trigger from the design. A set of 4000 traces (2000 for each bit) with a sampling rate of 500 MSam/s were collected using high-sensitivity EM probe.

6.4.2. Software Implementation and Evaluation Setup

The protected version of the popular μNacl library, which implements the ECSM operation on Curve25519 using the Montgomery ladder and randomized projective coordinates, is tested on ATmega2560 on an ARDUINO MEGA board running at 16 MHz. The design uses *cswap* (conditional swap) operation leading to key bit leakage (refer to [141]). The measurement setup is the same as before.

6.4.3. Experimental Results

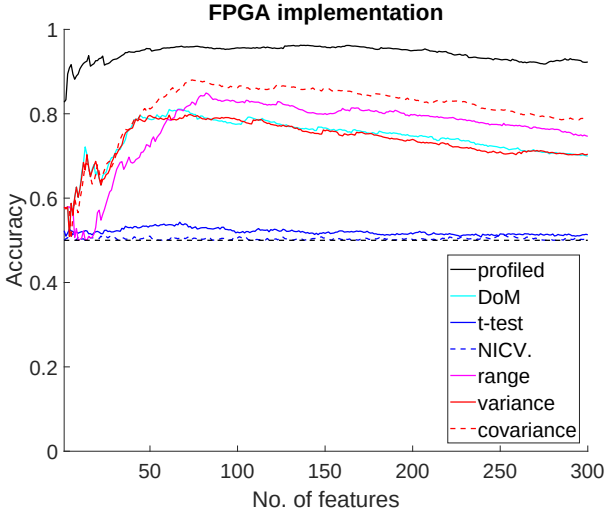
We performed the feature selection methods as described in Section 6.3.2, and we compared the results with the profiled scenario. *Accuracy* is used as comparison metric to evaluate the performance of different feature selection methods. We obtained 300 features based on each method. For profiled case, the features are obtained using t-test method (TVLA) on the labeled dataset. We also include the covariance based method to consider the bivariate feature selection scenario. The experiments are then conducted on 2 sets of data, collected from FPGA and ATmega respectively. The measurements from ATmega were misaligned when acquired leading to failure of attack. Thus, traces were realigned for further analysis. For profiling based attack, from the set of traces, the traces are divided for training and testing using 80:20 ratio (3200 traces for training and 800 for testing), whereas for non-profiled scenario, all the traces are used for the attack.

The accuracy results are then presented in Figure 6.1. As observed in the figure, the features obtained from the profiled TVLA approach yields the highest accuracy (96% for FPGA implementation and 99.9% for realigned microcontroller implementation). Most of the applied techniques surpass random guess. Among different methods, covariance based approach performs consistently better than the other approach. For the FPGA implementation, the achieved accuracy is 88.6%, whereas for aligned microcontroller implementation, the achieved accuracy is 98.43%. For other statistics, however, the results are less consistent. For example, in FPGA case, interval range gives the second best accuracy (84.75%), followed by variance (81%) and DoM (80%). However, for microntroller case, their accuracies are in the range of 55 - 65%, whereas, taking the t-test gives accuracy of 93.6%

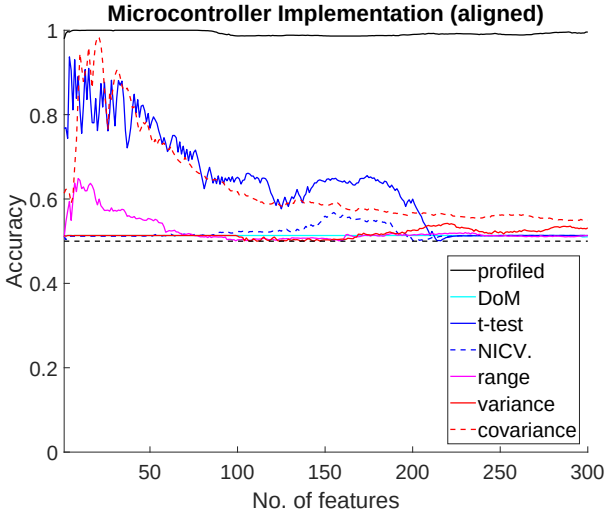
Hence, in these examples, we have shown that the proposed covariance approach can result in better accuracy, and has consistent performance across different target implementations.

6.4.4. Discussions

One of the issue that might arise for unsupervised feature selection is to determine the number of features used for the attack. Unlike profiling attack, in unsupervised



(a)



(b)

Figure 6.1: Accuracy of different feature selection methods, on different target implementations

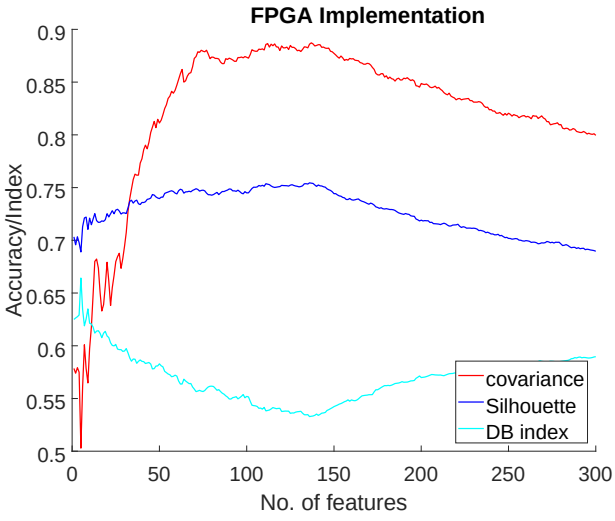
scenario, there is no label available to evaluate the accuracy directly. One idea for evaluating the features selected is to use the internal metric evaluation commonly used in clustering. We then choose Silhouette index [142] and DB index [143]. Normally, these metrics are used to evaluate number of clusters, however, in [131], they apply these metric to evaluate the quality of the cluster (since the initialization of the cluster is randomized) to determine the best resulting cluster. Similarly, we use these metrics to determine the optimal number of features. We conduct preliminary experiments on FPGA traces for the same and the optimal number of features returned by both the metrics (maximum value for Silhouette and minimum value for DB index) agrees with the best accuracy results. The results are presented in Figure 6.2. However, for the same experiments on the AVR microcontroller, the optimal number of features returned did not exactly correlate with the best accuracy. This could be due to misalignment in the traces. A good method for selecting number of features is still an open question.

k

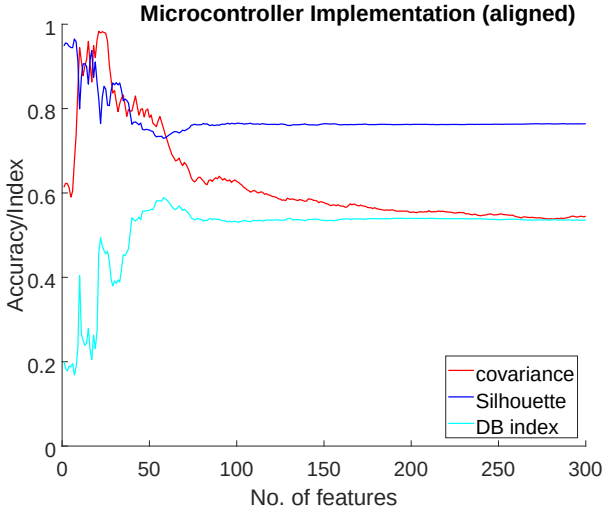
We could alternatively use deep learning to deal with misalignment. As shown in [144], in the presence of jitter countermeasure, Convolutional Neural Network (CNN) [145] can help to identify the targeted leakage parts in the traces. In our case, we targeted the misaligned traces (similar effect to jitter countermeasure). In profiling scenario, we could achieve the accuracy of 97 - 100% (as compared to 51.9% for profiling case with misaligned traces) without the need for realignment. Hence, a potential future work is to optimize the CNN algorithm for the misalignment case and extend it to the case of unsupervised scenario.

6.5. Conclusion

In this work, we have investigated different approaches for improved feature selection from ECC traces. It has been shown that choosing relevant features could help to improve the performance of the classification process. It has been shown that considering individual feature during the unsupervised phase might lead to the selection of non-optimal features unrelated to the targeted secret. Here, we proposed using covariance method (considering two points or bivariate) and showed that the proposed approach could minimize the selection of irrelevant feature and could help improving the accuracy of the classification. We also discussed some potential approaches which could further improve feature selection which we consider for future work.



(a)



(b)

Figure 6.2: Comparison between accuracy and returned estimation from Silhouette and DB index

IV

Active Side-Channel Attacks and Countermeasures

In accordance with the structure outlined in Figure 1.1, this section is focused on the third constraint, which is the resistance of cryptographic algorithms to fault injection attacks. Fault injection attacks are a type of attack that involves deliberately introducing faults or errors into a cryptographic system in order to exploit vulnerabilities and extract sensitive information.

In recent years, fault injection attacks have become increasingly prevalent, particularly in the context of embedded systems and IoT devices that are often subject to resource constraints. Therefore, the ability of a cryptographic system to resist fault injection attacks is a crucial aspect of its overall security.

This section will explore the various methods and techniques that can be employed to enhance the fault resistance of cryptographic algorithms, including the use of redundancy, randomization techniques and physical detection. Additionally, we will investigate the trade-offs that come with implementing these countermeasures, including increased computational overhead and resource utilization. Through this exploration, we aim to provide a comprehensive understanding of the strategies that can be employed to bolster the fault resistance of cryptographic systems, and their associated benefits and limitations.

7

SoK : On DFA Vulnerabilities of Substitution-Permutation Networks

Recently, the NIST launched a competition for lightweight cryptography and a large number of ciphers are expected to be studied and analyzed under this competition. Apart from the classical security, the candidates are desired to be analyzed against physical attacks. Differential Fault Analysis (DFA) is an invasive physical attack method for recovering key information from cipher implementations. Up to date, almost all the block ciphers have been shown to be vulnerable against DFA, while following similar attack patterns. However, so far researchers mostly focused on particular ciphers rather than cipher families, resulting in works that reuse the same idea for different ciphers.

In this article, we aim at bridging this gap, by providing a generic DFA attack method targeting Substitution-Permutation Network (SPN) based families of symmetric block ciphers. We provide the overview of the state-of-the-art of the fault attacks on SPNs, followed by generalized conditions that hold on all the ciphers of this design family. Furthermore, we propose a novel approach to find good fault masks that can leak the key with a small number of instances. We then developed a tool, called *Joint Difference Distribution Table (JDDT)* for pre-computing the solutions for the fault equations, which allows us to recover the last round key with a very small number of pairs of faulty and non-faulty ciphertexts.

7.1. Introduction

Substitution-Permutation Network (SPN) is a fundamental design strategy for block ciphers, with many primitives using SPN either for a part of their design or as the main design concept. An SPN consists of one or more (usually many) iterations of the following three operations:

1. Substitution (confusion): The state of the network is divided into words and a non-linear substitution is applied to each of them. The substitution function can be the same for all of them, or different functions can be used for different words.
2. Permutation (diffusion): A state-wise permutation is applied. This step is responsible for propagating the information between the internal state words as fast as possible.
3. Key mixing: A secret key is mixed with the state, usually using an XOR operation.

The maximum security level possible for a block cipher is measured by its resistance against brute force attacks, hence by the bit-size of the master key used to generate the round keys. However, there are many cryptanalytic techniques that try to push this boundary, by studying the specific properties of the building blocks of the cipher. Most of these techniques fall into one of two categories:

1. Linear Cryptanalysis: the attacker tries to approximate the cipher as a group of linear equations between the input, output and key bits, with high probability.
2. Differential Cryptanalysis: the attacker tries to leak information about the key bits by observing the differences between different input/output pairs.

Over the years, many design techniques and studies have been established in order to build ciphers that are secure against these two types of attacks. The idea in case of SPN is that the substitution layer provides highly non-linear relations between bits of internal words, with low maximum difference probability, while the permutation layer mixes these relations together, increasing the complexity. By repeating these two operations many times (rounds), mixing with (random) key bits in every iteration, the plaintext/ciphertext pair should be practically indistinguishable from two uniformly random vectors. However, if the number of iterations is not enough, the previous statement cannot be true. Thus, usually cryptanalysts start by analyzing reduced-round versions of the SPNs in question, while the cipher designers try to increase the number of rounds beyond the maximum number of rounds with non-ideal properties.

Surprisingly, the reduced-round properties of SPNs have been useful beyond the theoretical analysis and/or defining the minimum required number of rounds for a cipher. With the emergence of fault attacks as a rising domain in the field of hardware security, the attacker can change some of the internal bits of the cipher in the last few rounds and use the properties of only these rounds to leak information about the key. For example, there are practical fault attacks against AES that use the differential properties of 1, 2, 3 or 4 rounds, while classical attacks require the properties of 10 full rounds. However, in case of fault analysis such as Differential Fault Analysis (DFA), the attacks generally either rely on heuristic analysis, empirical data or on general ideas that do not take the specifics of the cipher into consideration. Similarly, the countermeasures for these attacks are generally either at the implementation level [146–148] or at the the protocol/encryption mode level [149–152].

The systemization of DFA on SPNs started by the work of Piret and Quisquater [153], where they proposed a somewhat general DFA analysis of AES-like ciphers. Although their attack has been enhanced in several subsequent works, it was still exclusive to AES-like ciphers. Besides, the complexity of the attack is high and the complexity analysis is approximate (Section 7.2). Moreover, the attack did not discuss how to identify the optimal faults thoroughly. On the other hand, in [154], the authors provided a discussion on the optimality of different DFA attacks, which we revisit in our paper and identify the advantages and shortcomings of their approach. Hence, we identified a need for the systemization of DFA attacks on SPNs, in order to have a general methodology for analyzing SPNs, as opposed to analyzing each cipher independently. This helps not only to study and compare the available SPNs in literature, but to analyze future SPNs, as well.

The goal of this paper is to understand why different SPNs behave differently against DFA, and what properties make an SPN stronger or weaker in this context, achieving a better understanding of how the two layers of an SPN interact with each other in the context of DFA. We study some of the available DFA attacks against SPNs, identifying the weak points of the SPN design strategy against DFA. We also find general vulnerabilities of SPNs against these attacks, enabling us to find new attacks against modern SPN ciphers. The main idea is to find a new approach to identify a good location for fault injection and quantify the corresponding expected amount of information leakage. For most SPNs, the attack of choice is a single-word fault injected in round $r-2$ for an SPN with r rounds, using a 2-round distinguisher. We identify what are the weaknesses that are common for all SPN ciphers and what are the differences between them. In the process, we also propose a method for efficiently performing the attacks using a Time-Memory trade-off, which allows to pre-compute a big part of the analysis, with the ability to reuse it for any attack instance. We believe the proposed methodology will serve as

a useful tool in analyzing a plethora of ciphers expected to be submitted for NIST lightweight cryptography competition.

Our Contributions

1. We revisit the information theoretic approach from [154], providing new insights on how the differential properties of the function attacked and the fault distribution affect the information leakage, showing that, for any *deterministic non-linear* bijective function of the form $S(x) \oplus K$, where x and K are unknown, the entropy of K can be reduced by calculating $S(x \oplus \Delta x) \oplus K$, as long as Δx is non-uniform. Hence, *we show that it is not possible to design an SPN-based cipher that is inherently secure against DFA* (Sections 7.3 and 7.4).
2. We *formalize* the complexity of retrieving the master key by injecting a fault into the last round of an SPN. Next, we propose an approach to accelerate such attack by reducing the number of faults and maximizing the information leakage per fault (Section 7.4).
3. Once a good fault location is identified, we propose a new tool for precomputing the solutions of the fault equations, called the *Joint Difference Distribution Table (JDDT)*, based on the fault model, and the properties of the substitution and permutation layers (Section 7.4.2).
4. We describe a class of SPNs that share a similar 2-round distinguisher and provide a new method to analyze the differential properties of related attack and use it to find key candidates for the last round key by observing a single pair of faulty and non-faulty ciphertexts (Section 7.5). We validate this analysis on modern ciphers, such as PRESENT-80, PRESENT-128, GIFT-64, GIFT-128, AES-128, SKINNY-64-64, SKINNY-128-128, PRINCE, LED-64 and LED-128 (Section 7.6). We report three different kinds of results:
   - M *Match optimal results* of well-studied ciphers, like AES
   - O *Find optimal attacks* for less studied ciphers, superseding previous known best attacks like PRESENT, LED, PRIDE
   - N *Find new attacks* on recently proposed ciphers with no (or little) public fault analysis like SKINNY, GIFT
5. We discuss some general intuitions or good practices which can help cipher designers to improve security of their cipher against DFA.

Table 7.1 shows the ciphers analyzed in this paper using our technique, which is also illustrated in Figure 7.1. It shows that some of the lightweight ciphers are

Table 7.1: Comparison between the different ciphers analyzed by our analysis technique. The location of the fault is considered to be known. N denotes new results, O denotes optimal results compared to previous results.

			Remaining Brute-force Complexity					
Cipher	Fault Model	Implementation	1 pair	2 pairs	3 pairs	4 pairs	16 pairs	Attack Type
AES-128	Random Byte	Any	$2^{8.06}$	1	1	1	1	M[155]
AES-128	1-Bit Flip	Any	$2^{0.15}$	1	1	1	1	O
LED-64	Random Nibble	Any	$2^{10.4}$	1	1	1	1	M[156]
LED-64	1-Bit Flip	Any	$2^{9.5}$	1	1	1	1	O
LED-128	Random Nibble	Any	$2^{74.4}$	$2^{20.8}$	$2^{10.4}$	1	1	M[156]
LED-128	1-Bit Flip	Any	$2^{73.5}$	2^{19}	$2^{9.5}$	1	1	O
PRESENT-80	4-Bit Flip	Any	2^{41}	2^{2}	1	1	1	O
PRESENT-80	Random Nibble	Bit-Sliced	2^{41}	2^{2}	1	1	1	O
PRESENT-128	4-Bit Flip	Any	2^{95}	2^{67}	2^{34}	2^{6}	1	O
GIFT-64	4-Bit Flip	Any	$2^{111.175}$	$2^{100.2}$	$2^{86.3}$	$2^{72.4}$	1	N
GIFT-128	4-Bit Flip	Any	$2^{111.175}$	$2^{100.2}$	$2^{86.3}$	$2^{72.4}$	1	N
PRIDE	4-Bit Flip	Any	$2^{104.6}$	$2^{87.2}$	$2^{72.4}$	2^{64}	1	O
SKINNY-64-64	Random Nibble	Any	2^{51}	$2^{40.4}$	$2^{34.8}$	$2^{5.6}$	1	M[157]
SKINNY-128-128	Random Byte	Any	$2^{102.4}$	$2^{81.28}$	$2^{69.76}$	$2^{11.52}$	1	M [157]
PRINCE	Random Nibble	Any	$2^{93.2}$	2^{48}	$2^{13.2}$	1	1	M[158]
PRINCE	4-Bit Flip	Any	$2^{84.16}$	2^{48}	$2^{4.16}$	1	1	O
Other Differential Fault Analyses in Literature								
PRESENT-80 [159]	16-bit Flip	Any	2^{40}	2^{16}	2^{6}	1	1	
PRESENT-80 [160]	1-bit Flip + Side Channel	Any	2^{80}	2^{76}	2^{66}	2^{64}	1	
PRIDE [161]	16-bit Flip	Any	$2^{86.4}$	2^{64}	$2^{22.4}$	1	1	

harder to break. However, none of these ciphers can be considered secure, since the key can be uniquely identified using at most 12 faulty and non-faulty ciphertext pairs.

7.2. Background

Differential Fault Analysis (DFA) is the oldest and most popular fault analysis method targeting symmetric cryptography. Since its inception in 1997 [162], almost all the symmetric block ciphers have been shown vulnerable against it.

The working principle of DFA is as follows. The attacker first runs an encryption procedure on plaintext P with a secret key K without disturbing the computation. Then, she repeats the encryption with the same inputs, but injects a fault, normally during the last few rounds of the cipher. She compares the faulty ciphertext with the correct one and gets information about one of the round keys. Depending on attacker model and the cipher structure, she repeats the fault injection several times until the guessing complexity of the key is low enough to get K.

The trend in analyzing block ciphers with DFA usually follows the same pattern for different encryption algorithms – first, an intuitive approach appears, requiring

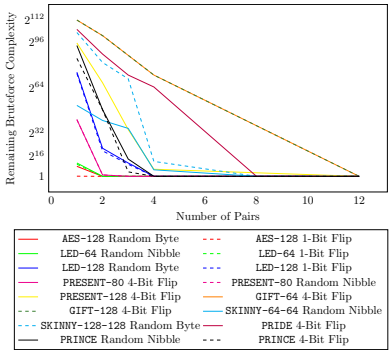


Figure 7.1: The number of remaining key candidates vs. the number of faulty and non-faulty ciphertext pairs for the ciphers in Table 7.1

more faults but lower brute-force complexity. Later, researchers tend to develop more sophisticated techniques that can reveal the secret key with either single or very low number of faults, while increasing the complexity of the analysis step. As an example, one can take DFA on AES, that improved from the early approaches requiring 35-250 faulty encryptions [163, 164], to more recent one that needs just a single fault [155]. Similarly, first DFA of PRESENT required 65 faults [165], later decreased to 2 [166].

Related work about DFA on SPNs. The work presented in our paper is closely related and inspired by the work of Piret and Quisquater [153], which was later extended and optimized by Tunstall et al. [155]. In this paper, we extend this line of work in three directions:

1. Instead of performing an approximate analysis of the attack complexity, assuming ideal primitives (Sbox and diffusion layers), we incorporate the details of the cipher in question into the analysis. While this leads to similar results in case of AES, since the Sbox of AES is well designed (almost ideal) and the diffusion layer uses an MDS matrix, the results concerning lightweight ciphers are different, due to the non-ideal primitives used. This analysis enables us to compare ciphers with respect to DFA security, beyond the simple bit security, showing that SPN ciphers with the same block and key sizes are not necessarily the same when it comes to security against DFA.

2. The analysis from [153] was targeting AES-like SPNs. We extend our analysis to a wider class of SPNs to include also bit-permutation based ciphers, such as PRESENT and GIFT, and SPNs with a diffusion layer that depends on an almost-MDS matrix, such as SKINNY.

3. We provide a framework for efficiently implementing the computational part of the attack, showing that a huge part of the attack can be pre-

computed only once per cipher and reused to attack as many instances as required, as opposed to the random search approach used in previous works.

Multiple-Fault Attacks on SPN. While the definition of whether a fault is considered a single fault or multiple fault is vague and generally implementation-dependent, it is usually the case that a single fault is either a single-bit fault or a fault that is limited to b bits, which is the input size of 1 Sbox. In our analysis, we consider any fault distribution over more than b bits to be a multiple fault attack and out of the scope of our paper. However, recently a similar analysis to ours has been presented by Lac et al. [167] which discusses a multiple-fault model. Their analysis was described for the LS family of block ciphers, then extended to AES-like cipher, e.g. AES and LED. The idea of their analysis is to accelerate the attack in Section 7.4 by injecting a multiple-word fault to the input of the diffusion layer in the second last round, such that all the Sboxes in the last round are active, with known input differences. However, the multiple-word known fault model is less practical compared to our work and is not suitable for some implementations. Moreover, in [167] only a special class of Sboxes was considered. Hence, in Section 7.5, we propose a single fault DFA for a wide class of SPNs.

Multiple-fault attacks were also used to attack PRESENT [159] and PRIDE [161], with each of these attacks requiring to flip 16 bits simultaneously.

Definitions.

Throughout this paper, we use some definitions for fault models that are listed below:

- Single fault: a fault whose maximum width is $\leq b$, where b is the number of input bits to the Sbox used in the cipher in question.
- Uniform/Random fault: a fault that can have any value between 1 and $2^b - 1$, where all the values are equiprobable.
- Constant/Known fault: a fault that has a specific value defined by the attacker.

For all the faults used in this paper, the location and timing of the faults are assumed to be known to the attacker. In practice, if the attacker is uncertain about the timing or location of the fault, he needs to repeat the analysis for every possibility.

Practical Fault Models. Three fault models are used throughout this paper:

1. Random fault model: A random byte/nibble is added to an internal byte/nibble of the cipher. This is the most practical fault model used

in this paper and it has been used in several practical attacks [155, 156, 158].

2. Single bit flip: A specific internal bit of the cipher is flipped. In practice, it is more complex than the random fault, but it has been shown to be practical in several papers/attacks [168–171].

3. Four-bit flip: 4 adjacent internal bits are flipped together. While this requirement can be challenging, it was shown in [159] that such fault is practically possible. Moreover, there are a few tricks the attacker can use to get around this requirement. In Section A, we show some tricks that work around the requirements of this model, showing that for some implementations, the attacker can achieve the required fault with only a random fault injection. The idea is that with some knowledge on the nature of the implementation, the attacker can use a random fault that can only lead to the required fault value.

7.3. Information Theoretic DFA Model: Towards a theoretical security metric for DFA

In an effort to find a theoretical metric for studying the security of ciphers against DFA, the authors of [154] introduced an information theoretic approach for evaluating whether a DFA is optimal or not. Given a fault model at the input of a function $S(x)+k$, the authors provided an information theoretic equation that can be used to calculate the maximum amount of information leakage under that fault model. First, we present the equation and then we present some results based on that equation, which contradicts with one of the inferential conclusions the authors made, due to the ambiguity of the mathematical definition of the fault model in the original paper.

Notation $X_i, Y_i, Z_i, K, \Delta X, \Delta Y$ and ΔZ are random variables defined in Table 7.2.

$$H(K|Z_1Z_2) = H(\Delta X|\Delta Y) + H(X_1|\Delta X\Delta Y) \tag{7.1}$$

Equation 7.1 can be used to calculate the entropy of the Secret Key, knowing a single pair of faulty and correct ciphertexts [154] (or, generally, any pair of ciphertexts). The first thing to notice is that $\Delta Y = \Delta Z = Z_1 \oplus Z_2$, which is public. However, the equation does not show how to calculate X_1 and ΔX. These two variables implicitly hold the information about the function $S(x)$ and the assumption about the fault model. Assuming a uniform fault model and no knowledge about $S(x)$, it is straightforward to deduce that no information about the key is leaked and

Table 7.2: Notation used in the rest of the paper.

X_i	input to the function $S(x)$ at a certain invocation
Y_i	output of the function $S(x)$ at a certain invocation
K	secret key
Z_i	$Y_i \oplus K$
ΔX	$X_1 \oplus X_2$
ΔY	$Y_1 \oplus Y_2$
ΔZ	$Z_1 \oplus Z_2$

hence $H(K|Z_1Z_2) = n$. In [154], the authors also calculate the entropy of the key $H_{\mathsf{AES}}(K|Z_1Z_2)$ assuming a uniform fault model and analyze the AES Sbox. Since $H_{\mathsf{AES}}(K|Z_1Z_2) = n$, the authors infer that AES Sbox is a good cryptographic function. While we are not challenging this conclusion, we can show that this analysis is not conclusive, as this result is achieved due to the uniformity of the fault model and not because of the properties of the AES Sbox. *In addition, we show that it is not possible to design any SPN that is inherently secure against DFA without randomization, where the target security level is considered to be at least the brute force complexity of searching for x, i.e.* $\mathcal{O}(2^{|x|})$.

First, we define $s_{\Delta x,\Delta y}$ as the number of solutions of Equation 7.2. Hence, $H(X_1|\Delta X = \Delta x, \Delta Y = \Delta y) = log(s_{\Delta x,\Delta y})$. Additionally, since $S(x)$ is a bijective function, $Pr(\Delta X = \Delta x, \Delta Y = \Delta y) = \frac{s_{\Delta x,\Delta y}}{2^n}$. Finally, $Pr(\Delta X = \Delta x|\Delta Y = \Delta y) = Pr(\Delta X = \Delta x, \Delta Y = \Delta y)$, since ΔY is public and $Pr(\Delta Y) = 1$.

$$S(x) \oplus S(x \oplus \Delta x) = \Delta y \tag{7.2}$$

Theorem .7 is used to find the amount of information leakage when the input difference follows a known distribution. The proofs of required for this section are available in Appendix .7.

If ΔX is sampled from $\mathcal{S}$, such that $|\mathcal{S}| = z$ and $P(\Delta X) = p_x$.then the expected number of leaked bits of K, when ΔY is observed is

$$n - \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x,\Delta y}}{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}} log\left(\frac{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}}{p_x}\right) \tag{7.3}$$

Given a pair of faulty and correct ciphertexts Z_1 and Z_2, if $\Delta X \in \{0,1\}^n$ is a uniform random variable, then $H(K|Z_1Z_2) = n$, regardless of the properties of the function $S(x)$. From Corollary .7, we can see that if the fault model is unbiased and unrestricted, the key space is not reduced, regardless of the cryptographic properties of $S(x)$. Hence, any fault model used in DFA must be non-uniform, with respect

to $S(x)$. If $\Delta X = \Delta x$ (constant), then using one pair (Z_1, Z_2), the key space can be reduced from 2^n to $s_{\Delta x,\Delta y}$. Only linear (affine) Boolean functions achieves the theoretical security bound $H(K|Z_1Z_2) = n \forall \Delta x$, regardless of the distribution of ΔX.

Despite that linear/affine functions achieve the required bound in terms of differential cryptanalysis, they are not helpful as they can be analyzed using linear cryptanalysis. For example, a function that looks like $z = L(x) + k$, where $L(x)$ is affine and z is known, can be analyzed as $L^{-1}(z) = x + L^{-1}(k)$ and the cipher is then attacked neglecting this function, with the target to find $L^{-1}(k)$ instead of k, since we can easily derive one from the other. Theorem .7 can be used to compare the quality of different fault values and to design attacks that maximize the information leakage. If $\Delta X = \Delta x$ (constant), then the expected number of leaked bits of K is $n - \sum_{\Delta y \in \{0,1\}^n} log(s_{\Delta x,\Delta y}) P(\Delta Y = \Delta y | \Delta X = \Delta x)$.

7.4. DFA against the last round of SPN

Theorem .7 and Corollary .7 can be used to provide a generic attack against any Substitution-Permutation-Network (SPN) that follows the description in Section 9.1. An important observation is that any linear function at the end of the last round can be effectively neglected. In other words, if the last step of the SPN is of the form $C = L(x) \oplus K_r$, where $L(x)$ is a linear function and K_r is the last round key, it can be converted into $L^{-1}(C) = x \oplus L^{-1}(K_r)$ and attack the cipher for the effective key $K_{eff} = L^{-1}(K_r)$. Then, the real key is calculated as $K_r = L(K_{eff})$. Hence, in the generic attack we consider this a standard step of no additional cost.

Using this structure, the space of last round key K_r of any SPN can be reduced using the following procedure. We assume the cipher has w words per state and the state size is n. Each word has size $b = \frac{n}{w}$.

1. For each substitution function in the last round, the Difference Distribution Table (DDT) is calculated and the minimum entry value is located. An input difference with such value in the corresponding row is selected for each function.

2. If the cipher includes a linear function $L(x)$ before the addition of K_r, we concatenate the function L^{-1} to the cipher, such that the output is $L^{-1}(C)$.

3. Iteratively, we inject a fault by flipping the bits of one word according to the corresponding input difference. By observing the output difference and applying Theorem .7, the space of the last round key K_{eff} bits XORed with this word is reduced to $s_{\Delta x,\Delta y}$, which is the number of solutions of the DDT for the input/output difference pair $(\Delta x, \Delta y)$.

4. By repeating this for every word, the overall space of K_{eff} is reduced from 2^n to $\prod_{i=0}^{w-1} s_{\Delta x,\Delta y}^{w_i}$, where $s_{\Delta x,\Delta y}^{w_i}$ is the number of solutions for word w_i of the last round, based on the corresponding input/output difference pair. The number of faults is equal to w.

For widely used ciphers, such as AES, LED, PRESENT and GIFT, the value of $s_{\Delta x,\Delta y} = 2$ for most of the possible input/output difference pairs, and $w = 16$. Hence, they require at most 16 faults and the resulting space for the last round key is 2^{16}. This is the simplest DFA attack against SPN. However, depending on the underlying cipher, optimizations can be found. For example, for the 4 mentioned ciphers, we can double the number of faults by repeating the fault injection operation for every word. While every DFA gives 2 candidates for a key word, the overlap between these candidates is the right key word, i.e. we get the actual key value. Another optimization is to use more sophisticated/smart fault injection mechanism to trigger many words simultaneously [159, 172]. However, we think that these optimizations are cipher-dependent and still follow the outline of the generic attack. Another aspect that is cipher dependent is the relation between the last round key K_r and the master key. If the master key cannot be uniquely determined from the last round key, the attacker can brute force the undetermined bits, which means that the number of candidates of the master key is the number of candidates for the last round key multiplied by the exponent of difference between the sizes of the two keys, $2^{|K_m|-|K_r|}$, where, K_m is the master key and $|X|$ is the bit length of X. Another solution, is to use the last round key candidates to decrypt the last round and use the same DFA attack to get candidates for the second-to-last round keys and try to compute the master key using the two sets of candidates, this can be repeated until enough round keys are obtained.

Since this is a generic attack, we can characterize the cipher by $2^{|K_m|-|K_r|} \cdot min(\prod_{i=0}^{n-1} s_{xy}^{w_i}) = 2^{|K_m|-|K_r|} \cdot \prod_{i=0}^{w-1} min_{(\Delta x,\Delta y)}(s_{xy}^{w_i})$. This shows the complexity of retrieving the master key using an instantiation of the attack.

7.4.1. Reduction of the number of faults

In this section, we describe a framework for accelerating the key recovery using lesser number of faults than what is described in Section 7.4. The goal of the attacker is to recover the key with the minimum number of faults possible, while keeping the differential analysis simple. For example, injecting the difference in the input block will maximize the number of active Sboxes, but the analysis of the faulty ciphertexts requires the full differential cryptanalysis of the cipher to be feasible, which contradicts the security of the cipher. In order to achieve that, he has to trigger more than one Sbox at once, while trying to maintain that the difference propagation remains simple. While it is hard to find the exact minimum number

of faults required, we describe a heuristic approach to approximate this minimum value. First, we define two properties of the Sboxes involved in the SPN.

Definition 13 *Interacting Sboxes two Sboxes are interacting if the outputs of these two Sboxes are combined together in the Linear Diffusion Layer.*

Definition 14 *Active Sbox is an Sbox for which the input difference is not equal to* 0 *when a certain fault is injected in the SPN.*

The goal of the attacker is to find the fault value/location that maximizes the number of active Sboxes while keeping the number of interacting Sboxes minimal. For example, for an attacker who can inject a known-byte fault at the input/output of any of the Sboxes of AES, his goal to maximize the number of active Sboxes in the final few rounds, while the number of interacting Sboxes has to be 0.

Example 1: Application to LS SPNs

The LS family of SPNs was proposed by Grosso et al. in 2014 [173]. The target of this family is to be able to have very efficient bit-sliced implementations. One round of the cipher is described as follows: a block of $n \times m$ bits is organized into an array:

$$\begin{bmatrix} b_0^0 & b_1^0 & \cdots & b_{m-1}^0 \\ b_0^1 & b_1^1 & \cdots & b_{m-1}^1 \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ b_0^{n-1} & b_1^{n-1} & \cdots & b_{m-1}^{n-1} \end{bmatrix} \tag{7.4}$$

Then, a non-linear substitution operation is performed on each column using an n-bit Sbox, followed by a linear diffusion operation applied on each row, using an m-bit linear function. Finally, the round key is XORed. The previous steps are repeated for r rounds. In [167], the authors proposed a DFA on this family, where a fault is injected in one of the rows before the diffusion layer in round $r-1$, such that the input to r has exactly one row where every bit is flipped. This activates every Sbox in round r with the same input difference Δx and output difference Δy_i. Consequently, the space of K_r is reduced to $\prod_{i=0}^{n} s_{\Delta x, \Delta y_i}$. However, since the DFA from [167] usually requires a fault over a large word of width 8 bits or more, depending on the row length, it is considered to be a multiple fault attack, if $m > n$. An interesting question would be if it is possible to find a set of low Hamming

weight differences, such that the bits whose value is 1 are located near each other, i.e. the set bits are spread over $\leq n$ bits, such that when these differences are applied to the input of the linear diffusion layer, the output difference has a high Hamming weight value. In other words, we trigger as much Sboxes as possible with a low Hamming weight fault. For example, we describe a toy-cipher that is constructed in an LS structure, using the AES Sbox and MixColumn operations. The state consists of an 8×32 bit array, as follows:

$$\begin{bmatrix} b_0^0 & b_1^0 & \cdots & b_{31}^0 \\ b_0^1 & b_1^1 & \cdots & b_{311}^1 \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ b_0^7 & b_1^7 & \cdots & b_{31}^7 \end{bmatrix} \tag{7.5}$$

Each round of the cipher consists of 32 column-wise Sbox operations, followed by 8 row-wise MixColumn operations. If the attacker can flip any 8 adjacent bits, there are 130 32-bit fault values that are spread over only 8 bits and trigger at least 24 Sboxes. However, the highest number of Sboxes that can be triggered with such model is 29 out of 32 Sboxes, with only a single value achieving this bound. Hence, for our toy cipher, the maximum number of active Sboxes, while maintaining the number of interacting Sboxes at 0, is 29. It is also noticeable that it is impossible for this toy cipher to have exploitable faults in any round except the last round where the number of active Sboxes is larger and the number of interacting Sboxes is 0.

In case of SCREAM [174], the state is a 16×16 bit array. Hence, a single fault according to our definition can be up to 16 adjacent bits. However, since 16-bit Sboxes are not common in practice, and since the bit-sliced implementation provided by the designers targets 8-bit microcontrollers, we search for fault masks that are spread over at most 8-bits. We consider two cases, the first case is when the 8 bits are any 8 adjacent bits at the input of the linear layer, while the second is when the 8 bits are exactly either the top half or the bottom half of the input. In the first case, the maximum number of active Sboxes is 14, while in the second case, it is 13. This means that in order to attack the last round of the cipher and activate more than 14 Sboxes, the fault has to be spread over 9 or more bits.

Example 2: Application to SKINNY

SKINNY is an AES-like block cipher presented in CRYPTO 2016 by Beierle et al. [175]. It is targeted for lightweight applications and uses the Tweakey framework to provide the possibility of using it as a tweakable block cipher [176]. It has 6 versions, 3 of which use 64-bit blocks and 4-bit nibbles/Sboxes, while the other 3 use 128-bit blocks and 8-bit nibbles/Sboxes. The master key sizes are 64, 128, 192, 128, 256, 384, for each of them, respectively. Generally, a version tagged `SKINNY-n-m` uses n-bit blocks and m-bit tweakeys. Being an AES-like cipher, it follows our description nicely, except that the mixing layer uses a non-MDS MixColumn matrix given below:

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{7.6}$$

Since 2 columns have 3 non-zero elements and 2 columns have 1 non-zero element, injecting a fault in round $r-2$ activates at most 3 Sboxes only in round $r-1$, and if the fault is injected in the proper nibble/byte, it can activate at most 7 Sboxes in round r, which is less than half the number of Sboxes, distributed as follows: 2 columns with 3 Sboxes, 1 column with 1 Sbox and 1 column with 0 Sboxes. If a nibble/byte fault is injected in round $r-3$, the number of interacting Sboxes cannot be equal to 0.

7.4.2. Joint Difference Distribution Table (JDDT)

We can define a single round of an SPN as a group of non-linear functions $S(x)$, where $S(x)$ consists of a linear part (diffusion layer) and a non-linear part (Sbox). The main idea of the JDDT is that when a single word difference Δ is injected into the input of the diffusion layer in round j, the inputs to n corresponding Sboxes in round $j+1$ are not independent, but are $\{l_1(\Delta), l_2(\Delta), ...l_n(\Delta)\}$, where $l_i(x)$ is a linear function from 1 word to 1 word, which corresponds to the difference propagation through the diffusion layer. Hence, instead of analyzing each of the Sboxes in round $j+1$ independently, we develop a Joint Difference Distribution Table (JDDT) for the n Sboxes, which is actually a portion of the DDT of the function $[A_1, A_2, ...A_n] \leftarrow S(\{a_1, a_2, ...a_n\})$. The JDDT of n Sboxes consists of exactly $2^{-(n-1)b}$ rows of the overall DDT of $S(x)$. The purpose of the JDDT is to provide candidates for the output value of $S(x)$, given Δ and the output difference.

We compute the JDDT as follows: we consider all the 2^{4b} possible output differences and divide them into four b-bit differences. We use each of these values to access the corresponding DDT and find all the possible input differences corresponding to this value. Typically, for good Sboxes, these would be four lists of around 2^{b-1} values each, which means $2^{4(b-1)}$ possible values for the difference at the output of the diffusion layer. However, only a subset of these values satisfies the relation $\{l_1(\Delta), l_2(\Delta), l_3(\Delta), l_4(\Delta)\}$. Hence, they are tested and only the solutions corresponding to valid differences are stored into the JDDT.

7.5. Three Round DFA Attack on SPNs

In this section, we describe a single fault DFA attack against a family of SPN-based block ciphers. This family includes majority of the widely used SPNs, such as AES, LED, PRESENT and GIFT. The advantage of the attack described in this section is that it uses a single-fault injection and a single pair of faulty and correct ciphertexts. It can be used as a security metric for an SPN against DFA attacks. The family of SPNs we consider has the following properties:

1. The state of the cipher consists of w words, each of b bits divided into g groups, where $w = 4g$ and $g \ mod \ 4 \equiv 0$.
2. Each round consists of a substitution layer that operates on every word individually, using a non-linear Sbox, followed by a diffusion layer that consists of two parts: shuffling and mixing.
3. The shuffling step generates new groups, such that every group consists of 4 words from 4 different groups in the previous round.
4. The mixing step performs a linear operation on the words of every group, such that every output word depends on the 4 input words. In other words, the mixing step operates on each group independently.
5. Every word at the output of the substitution layer of the current round affects exactly one group of the next round.
6. Every 4 groups before shuffling are mapped into 4 groups after shuffling.
7. There exists at east one word difference value δ_o, such that when exactly one word is active in round j, with difference δ_o at the input of the diffusion layer, 4 words are active at the beginning of round $j+1$ and 4 groups are active at the beginning of round $j+2$.

These steps are depicted in Figure 7.2.

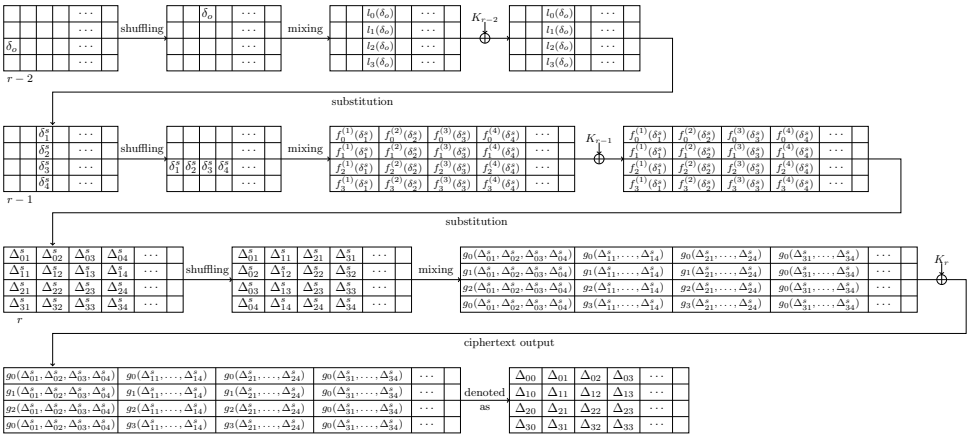


Figure 7.2: 3 Round DFA on SPNs

In case of AES, and LED, the last property is satisfied by all word differences, hence the attack can be launched with a random difference. On the other hand, in case of PRESENT or GIFT, this property is only satisfied when all the bits of the chosen word are inverted, i.e. $\delta = [1111]_2$. A random fault in the case of AES is not to be confused with the uniform fault model in Theorem .7, since $S(x)$ in this attack will be a 4 word to 4 word function consisting of the mixing layer followed by 4 Sboxes. Hence, the fault model used for AES is a restricted fault, picked from only 256 possible input values out of 2^{32} values.

The attack consists of an offline phase and an online phase. In the offline phase, first, we perform a slight modification to the SPN during the analysis. Instead of each round consisting of Substitution, Shuffling, Mixing and Addition of the round key K_r, we consider it to consist of Substitution, Shuffling, Addition of the effective round key $L^{-1}(K_r)$ and finally, Mixing. In this view, we merge Mixing and Substitution into one function $S(x)$. Then, we generate the extended DDT of the Sboxes in this function. Finally, we compute the JDDT for this function.

In the online phase of the attack, once the output difference of a group is obtained, the JDDT is accessed, and based on the assumptions on the input difference values (fault model, Sbox/Mixing differential properties, etc), a set of potential output values is required. These values are XORed with the ciphertext to get a set of keywords candidates of this group. The time complexity of the attack is $\mathcal{O}(2^{4b+2})$ $S(x)$ operations. Since a full encryption consists of around $g \times r$ $S(x)$ operations, the time complexity can be represented as $\mathcal{O}(\frac{2^{4b+2}}{g \times r})$ encryptions. The space complexity is the space required to store the JDDT, $\mathcal{O}(2^{5b})$. The number of key candidates is $|K_s| \times 2^{4b(g-4)+|K_m|-|K_r|}$, where $|K_s|$ is the size of the key space for the four attacked groups after the analysis and is a characteristic of the cipher.

The number of key candidates can be further reduced by generating all the candidates for the outputs of round $r-1$ and performing the online phase again for each, then keeping only the keys that satisfy a specific relation between K_r and K_{r-1}, such that both round keys represent a valid key schedule from the same master key. In round $r-1$ only one group is active, hence we can solve only for 4 key words. For every K_r candidate, we get $|K_s^{'}| \leq 2^{4b}$ candidates for these four words. Eventually, the key space size temporarily grows to $|K_s^{'}| \times |K_s| \times 2^{4b(g-4)+|K_m|-|K_r|}$. However, given the key scheduling algorithm of the cipher, which is used to generate round keys, we consider the relationship between the 4 words of round $r-1$ and the 16 words of round r. $P_{ks} \leq 1$ is the probability that such relation holds, assuming uniformly random bit assignment. Hence, the final key space size is $P_{ks} \times |K_s^{'}| \times |K_s| \times 2^{4b(g-4)+|K_m|-|K_r|}$ and the overall complexity of the online phase of the attack is bounded by $\mathcal{O}(\frac{|KS| \times 2^{4b(g-4)}}{r})$, which is the brute force cost of the second step of the attack. It is to be noted that the second step may not always lead to better results, as it depends on the key schedule of the cipher. For the attacker to gain from this step, the condition $P_{ks} \times |K_s^{'}| \ll 1$ should be satisfied.

7.6. Single Fault Attacks against real world SPNs

In this section, we first study the DFA on PRESENT-80, an SPN based on bit permutations. To the best of our knowledge, no single fault attack with a small number of pairs have been reported so far against it. We also discuss applying our technique to AES-128, showing it matches the best known attack against AES. Besides, we apply our technique to SKINNY, since is has some unique properties that can be exploited to achieve more efficient attacks. We provide more case studies in Appendix .6.

7.6.1. PRESENT-80/128: Finding Optimal Attack

PRESENT [177] is a lightweight block cipher proposed by Bogdanov et al. in CHES 2007. It targets applications such as RFID tags and sensor networks. It has been extensively studied over the years. [178] provides an analysis of the differential properties of PRESENT. In [159], the authors use a multiple fault model to attack it, flipping all the bits of 4 specific words at the same time. Since the fault model is specific and hard to achieve, they rely on a hardware Trajan to inject it. In [160], the authors presented a single fault attack on PRESENT. However, the attack requires side-channel assistance (namely power measurements) in order for the analysis to work, which is not required in this paper. At the end of this section we compare the attack from [160] to our attack.

Table 7.3: The 4-bit Sbox used in the PRESENT cipher

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

Table 7.4: The bit permutation used in PRESENT

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

PRESENT consists of 31 rounds, each round operates on a 64-bit block and contains three operations: addRoundKey, sBoxLayer and pLayer. addRoundKey is described as follows

$$s_j \leftarrow s_j \oplus k_j \tag{7.7}$$

where j is the round index and s_j, k_j are the cipher state and round key at round j, respectively. The next step is to divide the state s_j into 16 4-bit nibbles $b_j^{(i)}$, i.e. $s_j = b_j^{(0)} b_j^{(1)} \dots b_j^{(15)}$. Each nibble is replaced using the Sbox function $sb(x)$, defined using Table 7.3. The 16 parallel Sboxes represent the sBoxLayer.

Finally, the pLayer is described as a bit permutation $P(i)$ over 64 bits, where i refers to the bit index. The permutation is described in Table 7.4. For the purpose of the analysis in this paper, we describe the permutation using a different, yet equivalent, representation. The new representation consists of the shuffling and mixing operations described in Section 7.5. The shuffling operation starts after the sBoxLayer, by grouping the 16 $b_j^{(i)}$ into four groups, where each group is a column in the following matrix.

$$M = \begin{bmatrix} b_j^{(0)} & b_j^{(1)} & b_j^{(2)} & b_j^{(3)} \\ b_j^{(4)} & b_j^{(5)} & b_j^{(6)} & b_j^{(7)} \\ b_j^{(8)} & b_j^{(9)} & b_j^{(10)} & b_j^{(11)} \\ b_j^{(12)} & b_j^{(13)} & b_j^{(14)} & b_j^{(15)} \end{bmatrix} \tag{7.8}$$

Then, the shuffling operation is defined as the matrix transpose operation M^t.

Table 7.5: The mixing operation in PRESENT

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P'(i)$	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

Hence, the mixing operation can be described as 4 parallel instances of the permutation in Table 7.5, where a bit i is i-th bit of the nibble $g+\lfloor i/4 \rfloor \times 4$, and $g \in \{0,1,2,3\}$ is the group number. A somewhat similar representation of PRESENT was used for efficient software implementations in [179].

This representation makes some of the properties of the pLayer more clear:

1. The bits of nibbles $b_j^{(g)}, b_j^{(g+4)}, b_j^{(g+8)}, b_j^{(g+12)}$ (before shuffling) depend completely on the bits of nibbles $b_j^{(4g)}, b_j^{(4g+1)}, b_j^{(4g+2)}, b_j^{(4g+3)}$.
2. If only one nibble in group g is active at the input of the mixing operation with difference δ, the number of active nibbles after mixing is equal to the Hamming weight of δ.
3. If only one nibble $0 \leq i \leq 3$ in group g is active at the input of the mixing operation, then all the active nibbles after mixing have the same difference $\delta' = 2^i$.

Property 2 can be used to show that if one nibble is active at the input of the mixing operation, then only $\delta = [1111]_2$ leads to 4 active nibbles at the output. Property 3 can be used to find the value of the output differences from $\{1,2,4,8\}$. Hence, by injecting a fault δ in nibble i in the output of the sBoxLayer in round 29, a difference equal to $2^{i \ mod \ 4}$ is injected in the 4 nibbles of group $\lfloor i/4 \rfloor$. In round 30, the sBoxLayer changes these differences into $\{\delta_0, \delta_1, \delta_2, \delta_3\}$, then the shuffling operation distributes these four differences into 4 different groups. By using property 2, we can show that the number of active Sboxes in round 31 (the final round), is the Hamming weight of the vector $v = [\delta_0 \ \delta_1 \ \delta_2 \ \delta_3]$. In order to study the properties of the vector v, we need to study the differential properties of the Sbox. The analysis here refers to the differential cryptanalysis of PRESENT performed by Wang [178]. Depending on where the fault $[1111]_2$ is injected in round 29, four Sboxes in round 30 are triggered with difference $2^{i \ mod \ 4} \in \{1,2,4,8\}$. Table 7.6 represents the part of the DDT of the Sbox used in PRESENT corresponding to these values. The second-to-last column, μ_{HW}, is the average Hamming weight of the output difference, corresponding to the given input difference and a random value. It is clear that the maximum average number of active Sboxes in round 31, given a fault of $[1111]_2$ in round 29, is 12 and it is achieved when $i \equiv 3 \ mod \ 4$. In other words, the best locations, in terms of diffusion, to inject the fault are nibbles 3, 7, 11, or 15. Similarly, the value of the differences at the input of the active Sboxes in round 31 depends on the fault locations: $[0001]_2$ for $i = 3$, $[0010]_2$ for $i = 7$, $[0100]_2$ for $i = 11$, and

Table 7.6: Part of the DDT of the Sbox used in PRESENT

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	μ_{HW}	EXP(SOLs)
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0	2.5	2^2
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0	2.25	$2^{1.25}$
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0	2.25	$2^{1.25}$
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4	3	$2^{1.5}$

$[1000]_2$ for $i = 15$. The last column of Table 7.6 shows the expected number of solutions of the Sbox output value, assuming the given input difference and a random value, calculated using Theorem .7. It shows that the information leakage is maximized when $i \in \{7, 11\}$. Analyzing the last round using the attack described in Section 7.5, the number of candidates of the last round key $|K_s|$ is given by

$$|K_s| = 2^{1.25\times 12} \times 16^4 = 2^{31} \tag{7.9}$$

which shows a leakage of 34 bits by applying. In order to perform the second step of the attack, we need to look at the key scheduling algorithm of PRESENT and study the relation between the active bits of rounds 30 and 31.

PRESENT-80

The key scheduling algorithm for PRESENT-80 works as follows:

1. The master key K_m is stored in an 80-bit register as: $[\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_0]$.
2. The current round key K_i is given by $[\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_{16}]$.
3. $[\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_0] \leftarrow [\mathsf{K}_{18}\mathsf{K}_{17}...\mathsf{K}_0\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_{19}]$
4. $[\mathsf{K}_{79}\mathsf{K}_{78}\mathsf{K}_{77}\mathsf{K}_{76}] \leftarrow sb([\mathsf{K}_{79}\mathsf{K}_{78}\mathsf{K}_{77}\mathsf{K}_{76}])$.
5. $[\mathsf{K}_{19}\mathsf{K}_{18}\mathsf{K}_{17}\mathsf{K}_{16}\mathsf{K}_{15}] \leftarrow [\mathsf{K}_{19}\mathsf{K}_{18}\mathsf{K}_{17}\mathsf{K}_{16}\mathsf{K}_{15}] \oplus \texttt{round\_counter}$.

Assuming the contents of the register at round 31 are $[\kappa_{79}\kappa_{78}...\kappa_0]$, then the contents at round 30 are

$$[\kappa_{60}\kappa_{59}...\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\tilde{\kappa_{16}}\tilde{\kappa_{15}}...\kappa_0]sb^{-1}([\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76}])[\kappa_{75}\kappa_{74}...\kappa_{61}] \tag{7.10}$$

Besides, the round keys are given by

$$K_{31} = [\kappa_{79}\kappa_{78}...\kappa_{16}] \tag{7.11}$$

$$K_{30} = [\kappa_{60}\kappa_{59}...\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\tilde{\kappa_{16}}\tilde{\kappa_{15}}...\kappa_0]\texttt{truncate}(sb^{-1}([\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76}], 3)) \tag{7.12}$$

where $\texttt{truncate}(\mathsf{x}, 3)$ return the 3 most significant bits of x.

If the fault is injected in round 29 in nibble 7, then nibbles 1, 5, 9 and 12 are active in round 30, with an Sbox input difference equal $[1000]_2$. The correct solutions for

Table 7.7: The probability of having a key space of size k after 2 fault injections

k	0	1	2	3	4	5	6	7	8	$Pr(x \leq 8)$
$Pr(x = k)$	0.018	0.073	0.147	0.195	0.195	0.156	0.104	0.060	0.030	0.979

these nibbles of K_{30} must satisfy

$$[\kappa_{52}\kappa_{51}\kappa_{50}\kappa_{49}\kappa_{36}\kappa_{35}\kappa_{34}\kappa_{33}\kappa_{20}\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\kappa_4\kappa_3\kappa_2\kappa_1] \tag{7.13}$$

Since $\kappa_4, \kappa_3, \kappa_2$, and κ_1 are not used in K_{31}, the probability of satisfying the condition $P_{ks} = 2^{-12}$. Besides, since the 4 nibbles have an input difference of $[1000]_2$, and using Table 7.6, the number of solutions $|K_s^{-1}|$ for the 4 active nibbles of K_{30} is 2^6. Hence, combining steps one and two of the attack, the number of candidates of K_{31} becomes 2^{25}, and the number of master key candidates becomes 2^{41}, with complexity 2^{31}.

On the other hand, if the fault is injected in round 29 in nibble 11, then nibbles 1, 5, 9 and 12 are active in round 30, with an Sbox input difference equal $[1000]_2$. The correct solutions for these nibbles of K_{30} must satisfy

$$[\kappa_{56}\kappa_{55}\kappa_{54}\kappa_{53}\kappa_{40}\kappa_{39}\kappa_{38}\kappa_{37}\kappa_{24}\kappa_{23}\kappa_{22}\kappa_{21}\kappa_8\kappa_7\kappa_6\kappa_5] \tag{7.14}$$

Similar to the previous case, 12 conditions must be satisfied. Hence, the analysis is the same. If the attack is applied twice (2 fault injections), the second time leads to a new set of 2^{41} keys is calculated. However, the probability of one or more of the wrong key candidates overlapping the first set, i.e. the probability that the size of intersection between the two sets ≥ 1, can be computed using the Binomial distribution. The experiment is defined as selecting a uniform random key 80-bit vector, and it is successful if the selected key is one of the 2^{41} keys calculated during the first attack. Hence, $p = 2^{-39}$ and $Pr(X \leq k) = \sum_i^k Pr(X = i)$, which is also the probability of the key space size to k with 2 faults. Table 7.7 shows that the key space is reduced to at most 8 keys (77-bit leakage) after 2 fault injections, with probability 97.9%.

7.6.2. AES-128: Matching Best Known DFA Attack

AES [180] is considered as the standard block cipher for most applications. It was selected in 2001 by NIST through a public international competition. The design details can be found in [181]. It follows the description in Section 7.5 directly, with the ShiftRows operation representing shuffling and the MixColumns operation representing mixing. Moreover, since the MixColumns operation is designed to achieve the maximum branching number of 5, it follows immediately that if exactly a single byte at the input of MixColumns is active, all the four output bytes

must be active. Hence, a uniform random byte fault model will serve the purpose of the attack described in Section 7.5. However, as shown in the attack in PRESENT, sometimes it is easier to achieve a single-bit fault, e.g. bit-sliced implementation. We are going to study both cases in this section. We refer to [180] for a full description of AES.

Similar to PRESENT, the number of solutions of Equation 7.2 where $S(x)$ is the Sbox function, $s_{\Delta x,\Delta y}$, is either 0, 2 or 4 for any given pair $(\Delta x, \Delta y)$ except (0,0). However, as the AES Sbox is an 8-bit function, this means that the probability of any of these pairs where the number of solutions is not zero is much lower. Moreover, the DDT of the AES Sbox is more structured, such that for any given non-zero Δx, the number of values Δy that correspond to 0, 2 and 4 solutions $s_{\Delta x,\Delta y}$ are 129, 126 and 1, respectively. Hence, for any input fault difference and a random input value, the expected number of solutions for the output value of that Sbox is $2^{1.0156}$ solutions, according to Theorem .7. On the other hand, unlike the case of PRESENT, the diffusion in AES does not depend on any properties of the output value of the Sbox.

In order to asses the attack in Section 7.5, we need to study not just the properties of the DDT, but also the properties of the JDDT described in the attack. By injecting a single byte fault Δ in the input of the MixColumns operation, the four byte differences at the input of the Sboxes are $\{\delta, \delta, 2\cdot\delta, 3\cdot\delta\}$, or a rotation of this set, depending on which byte in the input is faulted. Hence, the number of solutions in the JDDT for an input difference δ and output difference $\{\Delta_1, \Delta_2, \Delta_3, \Delta_4\}$, is the number of solutions $\{y_1, y_2, y_3, y_4\}$, such that $\delta_1 = 2\cdot\delta, \delta_2 = \delta, \delta_3 = 2\delta$, and $\delta_4 = 3\cdot\delta$. This number is equal to $s_{2\cdot\delta,\Delta_1} \times s_{\delta,\Delta_2} \times s_{\delta,\Delta_3} \times s_{3\cdot\delta,\Delta_4}$. We can apply Theorem .7 on the 32-bit function constructed by performing the MixColumns operation followed by 4 parallel Sboxes for two cases:

1. $\delta \in \{0,1\}^8 - \{0\}^8$ is a uniformly random variable ($p_x = \frac{1}{255}$): This case is valid when the attack in Section 7.5 with a uniformly random fault. Given the output difference value, $\{\delta_1, \delta_2, \delta_3, \delta_4\}$ represent a 32-bit random vector, selected from 127^4 possible values. However, 24 conditions are imposed by the MixColumns equations. This limits the number of possibilities to $\frac{127^4}{2^{24}} = 2^{3.95}$. First, for simplicity, we assume that all the possibilities are equiprobable. Using Theorem .7, the expected number of solutions for any given 32-bit difference at the output of the mixing operation is $2^{4\times 1.0156} = 2^{4.0624}$ solutions, then the overall expected number of solutions is $2^{8.01}$. Since the AES state 4 columns (groups), step 1 of the attack in Section 7.5 reduce the last round key space from 2^{128} values to $|KS| = 2^{32.05}$. Applying step 2 of the attack, we get $|KS^{'}| = 2^{8.01}$. For AES-128, $|K_m| = |K_{10}|$ and a single round key is enough to deduce the master key and, hence, all the other round keys. Therefore, every can-

didate for K_{10} imposes 32 conditions on the 32 active bits of K_9, leading to $P_{ks} = 2^{-32}$. To sum up, after applying the two steps of the attack, the expected number of key candidates is $2^{8.06}$, with complexity $\mathcal{O}(2^{32.05})$. To assess the expected number of key candidates after applying the attack twice, we use a similar binomial distribution to the one used to assess PRESENT. Since, $Pr(X = 0) = 1 - 2^{-111.88} \approx 1$, therefore, it is expected that the attacker can uniquely identify the key using two faults with overwhelming probability. A similar result has been achieved by Tunstall et al. in [155], where they concluded that when a single fault is injected in round 8 at the input of the MixColumns operation, the key space can be reduced to 2^8 candidates. The analysis was specific to AES-128 and it used an approximation regarding the expected number of solution for each Sbox. Hence, the analysis in our paper is more generic and more conservative, showing that the number of key candidates after one fault is expected to be $\approx 4.2\%$ more than what they originally estimated.

2. δ is the output difference of an AES Sbox triggered by a constant input difference: This case is valid the attack in Section 7.5 with a constant fault. As discussed earlier, for some implementations, e.g. bit-sliced software implementations, it might be easier to inject a fault in a specific bit of the state. In such cases we use a stronger fault model, where $\delta = \delta_o$, such that δ_o is a constant known to the attacker and has a Hamming weight of 1. For this model, the first part of the analysis is similar. However, every difference δ_i at the input of the Sbox is restricted to 127 out of 255 values (as a known difference Δx for the AES Sbox can lead to 127 possible values for Δy). Hence, the number of key candidates after step 1 is reduced by a factor of $\frac{127}{255}^4$, leading to a total number of key candidates $|KS| = 2^{28.02}$. In step 2, $|KS^{'}| = 2^{4.0624}$, since we need to solve only for δ_o. Since P_{ks} depends on the conditions derived from the key scheduling algorithm and not the fault model, it remains the same. Consequently, the overall number of key candidates for this fault model is $2^{0.09}$.

In order to verify the previous results, we have implemented the pre-computation algorithm described in Section 7.5 for each case. We calculated the average number of leaked bits for every possible output difference (255^4 possibilities), according to Theorem .7. The computation was performed on an AMD Opteron 6378 quad-core processor. The average number of solutions for case 1 is $\approx 2^8$, and computing the JDDT (the pre-computation phase) takes around 14 CPU-Hours. For case 2, we need two JDDTs. The first one is for the first step of the attack, where δ can take 1 value with probability 2^{-6} and 126 values with probability 2^{-7}. The second

table is for the second step of the attack, considering a constant input difference. The first table is computed in 30 CPU-Hours on the same machine and leads to $2^{7.03}$ solutions per group on average. The reason of the time overhead compared to the random case is to calculate the conditional probability distribution used in Theorem .7. The second table is computed in 3 CPU-Minutes and leads to an average of $2^{4.034}$ solutions, for $\delta = 1$. The resulting key space has a size of $2^{0.154}$ or 1.1 candidates. Most of the time the key can be identified with a single fault.

7.6.3. SKINNY: Matching Best Known DFA Attack

A similar analysis to the analysis of SKINNY in this section was recently independtly reported by Vafaei et al. [157]. We briefly discussed the diffusion of SKINNY in section 7.4.1. While SKINNY does not exactly fall in the family of SPNs we are considering in this section, since the MixColumns matrix is not an MDS matrix, most of the analysis still applies to it. If the injected fault follows a uniform distribution, then according to Theorem .7 and the fact that there are no Mix-Column equations to reduce the space of this fault, the column with 1 Sbox cannot be used to reduce the key space, but can only be used when the input difference follows a non-uniform distribution. For example, the JDDT of the function constructed by 3 of the SKINNY 4 bit Sbox, such that all the input difference have to be equal shows an expected reduction from 2^{12} to $2^{4.36029}$ solutions for a uniform fault model. By analyzing the MixColumns matrix of SKINNY, we can find the best location for injecting the fault in round $r-2$ is any nibble/byte of the third row, i.e. b_8, b_9, b_{10} or b_{11} at the input of the MixColumns operations. The activity pattern of the last round Sboxes is as follows (up to a cyclic rotation of the columns):

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{7.15}$$

However, since the round key is added only to the first two rows, the key recovery method is slightly different from other AES-like cipher. The output value of the Sboxes in rows 2 and 3 in the last round is visible to the attacker. Hence, the attacker can use the invert those Sboxes and find the input differences to them. Since the input difference to all active Sboxes in the same column is the same, then the attacker knows the input difference to 3 of the Keyed Sboxes. Using this knowledge, he can reduce the key nibbles/bytes of these Sboxes to an expected value is $2^{1.4}$ key candidates for SKINNY-64-64 and $2^{2.88}$ for SKINNY-128-128, using

Table 7.8: The active Sboxes in the last round of SKINNY for every fault location

Fault location in round $r-2$	Vulnerable Sboxes in round r
b_8	b_0, b_2, b_4
b_9	b_1, b_3, b_5
b_{10}	b_2, b_0, b_6
b_{11}	b_3, b_1, b_7

Theorem .7, reducing the size of the key space of the last round key to $\approx 2^{24.2}$ for SKINNY-64-64 and $\approx 2^{48.64}$ for SKINNY-128-128, while the size of the master key space is reduced to $2^{56.2}$ and $2^{112.64}$ respectively. Since there are four potential good fault locations, in Table 7.8 we sum up the vulnerable Sboxes in the last round corresponding to each of these faults. We can observe that after 4 pairs of fault and non-fault ciphertexts, all the Sboxes in the last round have been activated, with b_0, b_1, b_2 and b_3 activated twice. This leads to reducing the number of candidates of the last round to between an average of $2^{5.6}$ and $2^{11.52}$ candidates for SKINNY-64-64 and SKINNY-128-128 respectively. Overall, The last round key can be uniquely identified with high probability using 4 more pairs. By repeating the attack for the second to last round, we can reduce master key space to an expected value of $2^{11.2}$ and 2^{23}, for SKINNY-64-64 and SKINNY-128-128 respectively, after 8 pairs and uniquely identify the key with high probability of after 16 pairs.

With this result, SKINNY seems to be the most immune cipher against the three round attack we describe in this paper. However, we show next that the reason it is immune against this attack is the same reason it is vulnerable to a more efficient 4-round attack, unlike other ciphers.

Four Round Attack on SKINNY

In this section we study the attack when the fault is injected in round $r-3$ instead of $r-2$. The reason this attack is possible is due to the fact that the diffusion in SKINNY is very slow. While it may not be possible to keep the number of interacting active Sboxes to zero after 4 rounds, we show that a value of only 2 interacting Sboxes can be achieved after 4 rounds. We consider the faults after 3 rounds (after applying ShiftRows and MixColumns respectively):

Table 7.9: The active Sboxes in the last round of SKINNY for every fault location

Fault location in round $r-3$	Vulnerable Sboxes in round r
b_8	b_0, b_2, b_4, b_6, b_7
b_9	b_1, b_3, b_5, b_7, b_4
b_{10}	b_2, b_0, b_6, b_4, b_5
b_{11}	b_3, b_1, b_7, b_5, b_6

$$\begin{bmatrix} a & 0 & c & d \\ 0 & e & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & p & 0 & m \end{bmatrix} \tag{7.16}$$

$$\begin{bmatrix} a \oplus i & p & c & d \oplus m \\ a & 0 & c & d \\ i & e & 0 & 0 \\ a \oplus i & 0 & c & d \end{bmatrix} \tag{7.17}$$

It is worth mentioning the same property of SKINNY was used by Liu et al. [182], but in a different context and with less details. Since the last round key is added only to the top two rows, our goal is to find the output values of the top 8 Sboxes, given the information about the bottom 8. Hence, the attacker already knows the values of a, i, e, c and d and can reduce the key space of the corresponding key nibbles/bytes. In Table 7.9, we show the active Sboxes in the last round corresponding to each of the four good fault locations. After one fault the number of candidates of the last round key is expected to be 2^{19} and $2^{38.4}$ for both versions of SKINNY, by applying Theorem .7. We can also observe that if we use 4 pairs, every Sbox in the last round is activated at least twice. While we can repeat the same attack one round earlier to get the full master key, we notice that the four pairs we have are already 4 valid pairs for the 3-round attack described in the previous section. Hence, we can use the same pairs to reduce the space of the round key at round $r-1$. Combining the two attacks together, the number of candidates for the master key after only 4 pairs is expected to be $2^{5.6}$ for SKINNY-64-64 and $2^{11.52}$ for SKINNY-128-128. With 8 pairs, the master key can be uniquely identified with high probability.

The DFA attacks against SKINNY apply directly to any SKINNY-n-m version, as long as only n bits represent the fixed master key and the rest can be controlled by the attacker.

7.7. Conclusion

In this paper, we presented a generic DFA on SPNs. More specifically, we formulated an information-theoretic model of fault attack in the last round of an SPN and showed that for any non-uniform fault, the key space can always be reduced. We proposed an attack method which injects a single fault in the second last round of a class of SPNs and by using a novel tool, called *Joint Difference Distribution Table (JDDT)*, the master key of the cipher can be recovered. Our method was evaluated on various block ciphers, including PRESENT-80, PRESENT-128, GIFT-64, GIFT-128, AES-128, LED-64, and LED-128.

For the future work, we would like to extend our method to other block cipher designs. Also, we note that our work can serve as a basis for designing fault resistant block ciphers. Besides, the JDDT method for analysis of SPNs can potentially be fully automated, which can be integrated with some cryptographic design and analysis software tools.

8

Fault Injection attack on Private Circuit II

Cryptographic implementations are subject to physical attacks. Private circuits II is a proven protection against a strong attacker, able to read and write on a finite number of chosen internal nodes. In practice, side-channel analyses and fault injections are less accurate: typically, classical injection techniques (clock and power glitches, electromagnetic pulses, etc.) can be reproducible, but they do not allow to choose the targeted nodes (the situation is different for software dual-rail with precharge logic, such as [183, 184], where $(0,1) \leftrightarrow (1,0)$ bitflips are easier to achieve, since the computation is fully sequentialized [185]). So, a priori, private circuits II should be a secure protection against such classical fault injection attacks.

In this paper, we provide the first implementation of private circuits II in FPGA, secure against read and/or reset of one internal wire chosen by the attacker. Our implementation is a manually coded netlist which instantiates LUT6_2 (with dual outputs, as required for private circuits II). Our design is a SIMON block cipher programmed in a Spartan 6 Xilinx FPGA. It features a throughput of 142 Mbit/s. We perform a security analysis, and notice that some exploitable ciphertexts can be outputted despite the countermeasure. Our analysis reveals that correlated faults exist because LUT6_2 outputs are produced almost simultaneously. In particular, the critical path actually consists in a dual-rail pair, which is consistently faulted together. If this pair is late with respect to the clock rising edge, then the previous value can be latched instead of the new one. Such fault behaves like a toggle ($(01)_2$ becomes $(10)_2$ or vice-versa) of licit values. They propagate to the ciphertext which

becomes by the same token susceptible to a differential fault attack. Nonetheless, we emphasize that such faults require a steady fault injection setup: otherwise, multiple critical paths are violated, resulting in non-exploitable (fully zeroized) ciphertexts.

8.1. Introduction

Cryptographic keys are safeguarded in secure hardware devices. Now, implementation-level attacks threaten the security of such sensitive devices. Those attacks can be classified in two categories: *passive* and *active*. Passive attacks are also known as side-channel attacks, and consist in collecting physical measurements leaked through the boundary of a device. Such leakage contains information about the internal variables handled by the device, which can be sufficient to extract secret keys. Active attacks rely on perturbations to force the device malfunction. Depending on the way the errors propagate within the device, the faulted output might reveal information about the keys.

Obviously, it is important to protect devices against such attacks. It is admittedly hard to completely prevent them, let alone because of the overhead incurred by countermeasures. Thus tradeoffs between security and cost must be devised. The ideal situation is when a security parameter allows to quantify the achieved security level.

Private circuits is the most acknowledged way to provably instantiate a countermeasure with clear and meaningful security parameter(s). Actually, private circuits appeared in the literature in two steps, namely private circuits I (PC-I, [186]) and private circuits II (PC-II, [187]). The aim of PC-I is to protect only against passive attacks. The attacker is assumed to be able to read k wires within a digital circuit, thereby collecting k bits of information per clock cycle. PC-I is a constructive method which shows how to design a circuit which resists the eavesdropping by such attacker. The rationale is to entangle random bits with the design so as to have any tuple of k wires be independent from the (unmasked) sensitive values. Now, attackers can also be active, and attempt to modify wires in the circuits. This is the motivation for PC-II: this time, a design method to resist against an attacker who is able to read k chosen wires and overwrite t chosen wires is proposed. The write operation might be of two types: only *reset*, or *set* to an arbitrary chosen value. The pair (k,t) is the security parameter of PC-II; whatever this pair, PC-II is able to generate an implementation which is provably secure. The PC-II style adds redundancy to the netlist in such a way that any change to $1,\ldots,t$ wires will cause an erasure of data after passing through a logic gate. For instance, when $t = 1$, the redundancy is the same as dual-rail with precharge logics [188, Chap. 7], where bit 0 (resp. 1) is encoded as $(01)_2$ (resp. $(10)_2$), whilst $(00)_2$ and $(11)_2$ are illicit values

used by PC-II as possible erased values.

Today, circuits are difficult to probe due to several reasons: first of all, the number of metal layers is huge (> 10) for the latest CMOS technologies. Accordingly, chip designers take proactive protection of sensitive signals by burying them, which makes them less accessible by a probing station. Besides, backside probing is also chancy because it is hard to know (at a resolution of a few tens of nanometers) what can be probed *blindly* through the silicon wafer. Additionally, backside probing requires costly equipments (called Focused Ion Beam stations), which are expensive and feature a non-negligible risk of permanently damaging the circuit [189]. Second, some technological protections (sensors of circuit lid opening, shields, etc.) attempt to detect probing attacks.

But there are different ways than probing to perform side-channel and perturbation attacks. Typically, state-of-the-art lab equipments allow to collect side-channel information without contacting the device. Power analyses only need an external monitoring of the amount of current flowing through the circuit, while electromagnetic analyses can be conducted even far away from the circuit. Fault injection attacks like overclocking or underfeeding do not allow to predict exactly their effect: the faults can be injected at multiple unpredictable locations inside the chip. Although slightly more local, electromagnetic pulse injection has also a coarse area of influence.

In this context, it is interesting to evaluate the suitability of PC-I and PC-II countermeasures against such *macroscopic* (by opposition to the *microscopic* scale of probing attacks) attacks. In theory, protecting against an attacker who can probe one wire is sufficient to protect against a first-order attacker, who exploits only the aggregated and noisy leakage from parts or totality of the chip. The same argument applies to active attacks: a protection which resists arbitrary access to carefully selected ($\leqslant t$) wires can, all the more, protect against an attacker who is less accurate in the injected faults of same multiplicity ($\leqslant t$).

But the study is still worthwhile. For instance, due to implementation constraints, some requirements of private circuits design style are hard to meet. This has already been demonstrated on PC-I implementations, where gates are assumed to evaluate in a precise order. However, it is known that unless every gate is made synchronous, glitches can occur which break the correct evaluation order requirement [190].

Contributions In this paper, we specifically focus on the practical evaluation of PC-II. Our contributions are three-fold. First of all, we implement for the first time a block cipher using PC-II, using security parameters $k = t = 1$. This choice for the security parameters implies the use of random bits in each gate (since $k > 0$ for

PC-I), and an encoding of each bit of the PC-I netlist as a dual-rail (since $t = 1$, value 0 / 1 in PC-I becomes $(01)_2$ / $(10)_2$ in PC-II). Second, we identify a weakness in our security assumption, which creates an exploitable vulnerability. Shortly, as PC-II is built on top of PC-I, PC-II inherits implementation constraints from PC-I; in particular, to avoid glitches, PC-II must be mapped such that dual-rail signals are balanced. We leverage on the dual outputs of LUT6_2 in FPGA to meet this constraint. However, such implementation opens the door to *correlated faults*, whereby both outputs of a LUT6_2 are faulted together. This negates the attacker model ($t = 1 < 2$). However, our a priori security analysis convinced us that it was apparently difficult to obtain correlated faults of multiplicity two in the practical setting of overclocking, underfeeding, or strong electromagnetic (EM) fields injection. Third, we demonstrate an analysis and an attack platform where we can assess experimentally the likelihood of correlated faults to happen and to propagate successfully to the output of the cipher, using overclocking, underfeeding, and EM injection, and with the assistance of an internal FPGA debugger (ChipScope Pro feature of Xilinx). In practice, for 50% of the plaintexts, we manage to generate an exploitable faulted ciphertext with carefully tuned fault injection parameters.

Outline of the paper The rest of the paper is structured as follows. Section 8.2 tackles the private circuits II principle and implementation results in FPGA. Security analysis of PC-II is conducted in Sec. 8.3. Fault injection results are given and discussed in Section 8.4. Finally, Section 8.5 concludes the paper and opens some perspectives.

8.2. Private Circuits I & II in FPGA

8.2.1. PC-I in FPGA, for $k = 1$

A private circuit I with security parameter k guarantees that any tuple of k wires does not convey any information about a sensitive value. There are several possible protections in the literature, for instance private circuits [186] or stateful private circuits [191]. In the sequel, we focus on the later. Private circuit I implements a notion of Boolean sharing: every bit is represented as a tuple of $(2k+1)$ wires, such that the bit value can be recovered by XORing together the $(2k+1)$ wires values. In the seminal paper [186], a proof of concept is shown based on a netlist which instantiates only few primitives:

- memory elements (typically a DFF), and combinational functions, namely:
- an "inverter" (INV) gate,
- an "exclusive-or" (XOR) gate and

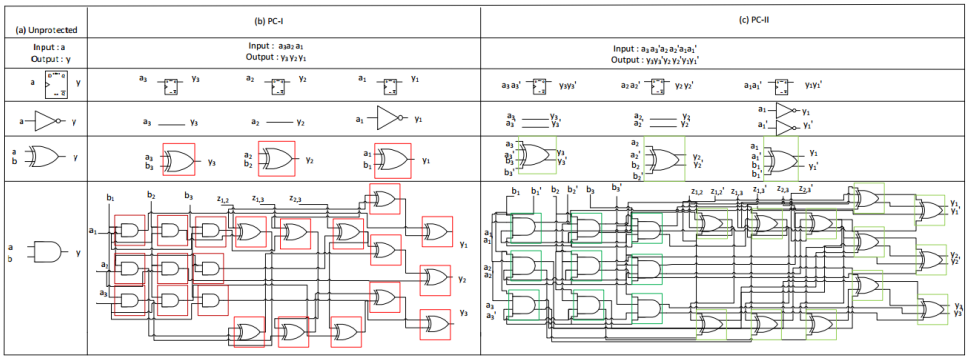


Figure 8.1: Gates protection in PC-I (with security parameter $k = 1$) and PC-II (with security parameters $k = t = 1$)

– an "and" (AND) gate.

The protection consists in replacing those instances by masked gates. Therefore, PC-I can be seen as a transformation from netlist to netlist. The mapping between the unprotected gates (DFF, INV, XOR and AND) and the PC-I version for $k = 1$ is given in Fig. 8.1 (a) and (b). Every bit a is thus transformed into a triple (a_1, a_2, a_3), and we notice that the secure evaluation of the PC-I AND gate requires 3 random bits, denoted as $z_{1,2}$, $z_{1,3}$ and $z_{2,3}$. The netlist transformation is straightforward because the PC-I transformation is compositional.

It can be seen in Fig. 8.1 (b) that we devote one full LUT6 (represented as red box – see Fig. 8.2(a)) to each gate, despite they have only one or two inputs. The reason is that we want to avoid synthesis optimizations which would (statically) reorder the gates. Obviously, this method is costly in terms of area, but it is guaranteed to be secure. Moreover, this netlist allows us to quickly implement PC-II (as discussed in next section 8.2.2). The configuration of the LUT6 for XOR and AND can be found in Tab. 8.1.

We notice that optimization of PC-I (which is out of the scope of our work) has been carried out in the literature in two directions. First of all, Park and Tyagi have improved the mapping of PC-I in FPGA by a better clusterization, without compromising the security [192, 193]. Glitch-free implementations have also been demonstrated recently [190, 194]. A convergence between PC-I and threshold implementation (a glitch-tolerant netlist style [195]) has been noticed recently [196]. Second, Rivain and Prouff [102] have adapted PC-I from the hardware case, which involves Boolean gates (they call PC-I the "ISW" scheme), to the software case, which involves machine words (like bytes). Their work has given rise to many applications, such as masked evaluation of substitution boxes [102, 197], and conversion algorithms between Boolean and arithmetic masking [198].

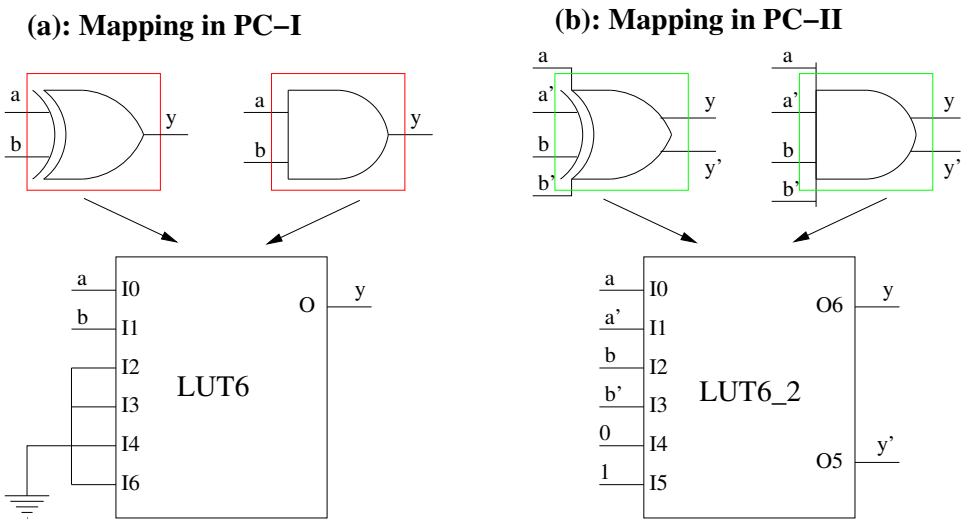
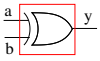
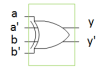
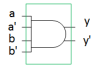


Figure 8.2: Mapping of Boolean functions, (a) in LUT6 as for PC-I, and (b) in LUT6_2 as for PC-II

Table 8.1: Configuration for PC-I and PC-II XOR and AND gates

		Schematic	INIT value
PC-I	XOR		INIT = $(6)_{16}$
	AND		INIT = $(8)_{16}$
PC-II	XOR		INIT = $(24000000420)_{16}$
	AND		INIT = $(40000000260)_{16}$

8.2.2. PC-II in FPGA, for $k = t = 1$

As already mentioned in the introduction, PC-II is a further refinement which enhances PC-I with fault resistance capability. The constructive method presented in the original paper [187] assumes several fault models: a predetermined number t of wires can be selected and either reset or set to chosen value, and so at each clock period.

We restrict to the case where the attacker is able to change the value of $t = 1$ wire. In this case, the PC-II protection consists in turning the PC-I circuit into a dual-rail equivalent. This means that:

- each wire x becomes a pair (x, x'), where x' is the opposite of x (that is, $x' = \neg x$), and that
- each gate is turned into a dual-rail instance (called a gadget in [187]), as illustrated in Fig. 8.1(c), where green boxes are LUT6_2 (cf. Fig. 8.2(b)).

The resistance against one fault arises from this argument: if one bit of a value (x, x') is corrupted, then the new pair becomes either $(0, 0)$ or $(1, 1)$. Now, in both cases, the final faulted value can be obtained irrespective x is equal to 0 or 1.

The PC-II gates are designed to be *infective*: should one input be invalid (that is, either $(0, 0)$ or $(1, 1)$), the gates propagate an invalid $(0, 0)$ value. This behavior of PC-II gates ensures an avalanche of zeroization, thereby preventing the attacker from collecting relevant faulted values at the end of the computation[1]

In Xilinx terminology, a LUT6_2 computes simultaneously two outputs named O6 and O5. The configuration of the LUT6_2 is encoded as a 64-bit integer called INIT [200]. Outputs O6 (resp. O5) execute the function whose truth table is given by the 32 upper (resp. lower) bits of INIT. We assign O6 to the true bit x of the dual-rail pair (x, x') while O5 is its complementary x'. Despite LUT6_2 has 6 inputs, four are needed for the PC-II gates. Specifically, the inputs are I0, I1, I2 and I3, whereas inputs I4 and I5 are fixed to 0 and 1. The exact configuration of PC-II gates is given in Tab. 8.1. Their construction is detailed in Tab. 8.2 for PC-II AND gate. Apart from licit values, which correspond to $(a, a') \in \{(0, 1), (1, 0)\}$ and $(b, b') \in \{(0, 1), (1, 0)\}$, other values are mapped to $(y, y') = (0, 0)$. Such mapping is similar to that of WDDL_noEE [201].

Remarkably, our implementation of PC-I and PC-II consist in the same netlists, let apart the configuration of the LUT masks (INIT values reported in Tab. 8.1).

[1] In the original paper on PC-II [187], authors describe in Tab. 2 a *cascade gadget* that artificially spreads the $(0, 0)$ value over the whole datapath. For the purpose of fault resistance, this is useless, as $(0, 0)$ propagate naturally through computing gates. But in order to have a very datapath-wide check, a simple computation of Hamming weight would be enough [199], [185, §3.1].

Table 8.2: Example of derivation of the INIT value for the PC-II AND gate programmed in a LUT6_2

a	a'	b	b'	y	y'
I3	**I2**	**I1**	**I0**	**O6**	**O5**
0	1	0	1	0 (INIT[37])	1 (INIT[5])
0	1	1	0	0 (INIT[38])	1 (INIT[6])
1	0	0	1	0 (INIT[41])	1 (INIT[9])
1	0	1	0	1 (INIT[42])	0 (INIT[10])
x	x	x	x	0	0

Table 8.3: Synthesis results for SIMON 96/96

	Unprotected	**PC-I**	**PC-II**
Max. frequency (MHz)	141	77	77
overhead:	./.	−45%	−45%
Registers	96 (<1%)	288 (<1%)	576 (<1%)
overhead:	./.	+200%	+500%
LUTs	1063 (1%)	3786 (4%)	5227 (5%)
overhead:	./.	+256%	+392%
(LUT6, LUT6_2)	(391, 0)	(2690, 0)	(98, 2592)

8.2.3. SIMON 96/96 in Private Circuits II

Private circuits are many times larger than their unprotected equivalent. Thus, we considered a lightweight block cipher, namely SIMON [202]. Interestingly, few studies concern secure implementation of SIMON (we found only a regular masking in [203]). We initially intended to implement SIMON [202] in its 128-bit version. But the overhead of private circuit is huge Thus a version of size of 96-bit for both plaintext and key was implemented on Xilinx Spartan 6 on the SASEBO-W board for the experiment. We notice that SIMON is particularly appropriate for an implementation in private circuits, because it is made only of XOR and AND.

8.2.4. Synthesis results for PC-I and PC-II in Xilinx Spartan 6

The synthesis targets a Xilinx Spartan 6 LX150 FPGA constrained to run at clock frequency of 24 MHz (very conservative value). The synthesis results are given in Tab. 8.3. In this table, the red numbers represent the occupied ratio on the FPGA. It clearly appears that the required resources increase when the circuit is

implemented in PC-I, and further increase when upgraded to PC-II. Actually, the core resources to implement SIMON are the same for PC-I and PC-II. Nonetheless, extra logic is required for the interface of PC-I and PC-II to the environment. Now, PC-I requires a wrapper to turn every variable a into a triple (a_1, a_2, a_3). On top of this wrapper, PC-II needs a conversion between single to dual-rail. So, every bit a is now encoded as a tuple of 6 elements $((a_1, a'_1), (a_2, a'_2), (a_3, a'_3))$. The values of the mask are generated thanks to a linear feedback shift register (LFSR) which yields a vector of size sufficient to feed the necessarily random bits. The LFSR is chosen for a quick implementation and evaluation but we are aware that a good PRNG/TRNG must be used for a real application. These numbers are computed thanks to a polynomial defined by a parameter according to the version of the SIMON. The throughput of SIMON in PC-{I,II} is 96 bit/52 clk $\times 77 \cdot 10^6$ clk/s $\approx$ 142 Mbit/s.

8.3. Security analysis of PC-II with $k = t = 1$

Obviously, our implementation is secure within the PC-II model with security parameters $k = t = 1$, i.e.:

- probing any wire does not disclose any information, and
- modifying any wire can neither be exploited.

Now, we aim to evaluate the resistance of PC-II against fault injection attacks.

8.3.1. Setup time violations

Any synchronous circuit must meet timing constraints: the *combinational gates* must have finished their evaluation before *sequential gates* can sample the result they computed. One can thus define a maximal operating speed for synchronous circuits. Of course, their operation is nominal only within certain environmental conditions, typically in terms of voltage and temperature.

Fault injection attacks consist in displacing the environmental conditions outside of the comfort zone for the circuit. This can be done permanently or transiently. For instance, a strong EM field varying quickly in the vicinity of the FPGA under test can locally create a voltage drop, thereby slowing down combinational gates. As the clock frequency is fixed, the setup time of combinational gates is violated. Symmetrically, the attacker can tamper with the clock, so as to accelerate it. The effect is similar: incomplete computations are sampled in the sequential gates (the DFFs).

If the field of the EM pulse is decreased, then the delays in the combinational paths are reduced. The borderline case is when only one bit in the datapath is faulted.

Such situation can be modeled as a single bit-flip error.

8.3.2. Timing faults on PC-II with $t = 1$

In PC-II circuits, a single bit-flip is harmless: indeed, PC-II resists against $t = 1$ fault.

Now, we argued that EM injection or overclocking/underfeeding are inaccurate faulting methods. Thus, it can be expected that if the stress is slightly increased, the second fault (e.g., bit-flip) will occur on a wire unrelated to the critical path. Hopefully, PC-II is able to withstand such double independent attacks: indeed, two unrelated LUT6_2 gates will output $(0,0)$ (recall Tab. 8.2), thus zeroizing the rest of the computation (cryptographic computations have a fast diffusion).

Now, in practice, we notice that in dual-rail circuits, the second critical path is actual (very often) the very matching pair of the critical path. Indeed, in a pair (x, x') of PC-II wires, the timings are almost balanced.

As argued in Sec. 8.2, the two nets from a same pair pass through the same LUT6_2, hence have (approximate) balanced timing. More precisely, it is known that balanced routing is hard to achieve in FPGA, because lack of control over the tools and lack of information about the internal structure and delays.

A previous work showed that having the pair of wires pass through the same LUT (by exploiting their dual outputs) can significantly reduce the unbalance [204]. Indeed, the "graph" for both wires is the same. Now, unbalances remain as the intra-LUT and LUT-to-LUT delays can vary. Accurate balancing can be achieved with third party tools, such as RapidSmith [205].

Still, for our argumentation, it is sufficient to know that dual-rail pairs have similar delays. Hence they are very likely to be faulted simultaneously upon setup time violations[2].

Such intrinsic problem of dual-rail circuits is illustrated in Fig. 8.3. It shows on its top an excerpt from a circuit, where nets (x_0, x_0') are produced by a LUT6_2. The "combi" cloud is typically a series of gates such at those given in Fig. 8.1(c). The routing between these nets is assumed different: the value x_0 arrives faster than x_0' to the DFFs. This is represented in the simulation (a) of Fig. 8.3. We stress here that, because in PC-II logic, gates are self-synchronizing, the only timing discrepancy of signals (x_0, x_0') when reaching their sampling DFF (x_1, x_1') is caused by a timing unbalanced of the routing between the last LUT6_2 in the logic cone (i.e., the "combi" cloud in Fig. 8.3) and the corresponding DFF. If, due to stress (under-

[2] We stress that this noting is quite innovative, because former papers (see e.g., [206]) consider that faults are *uniformly distributed* over the circuit.

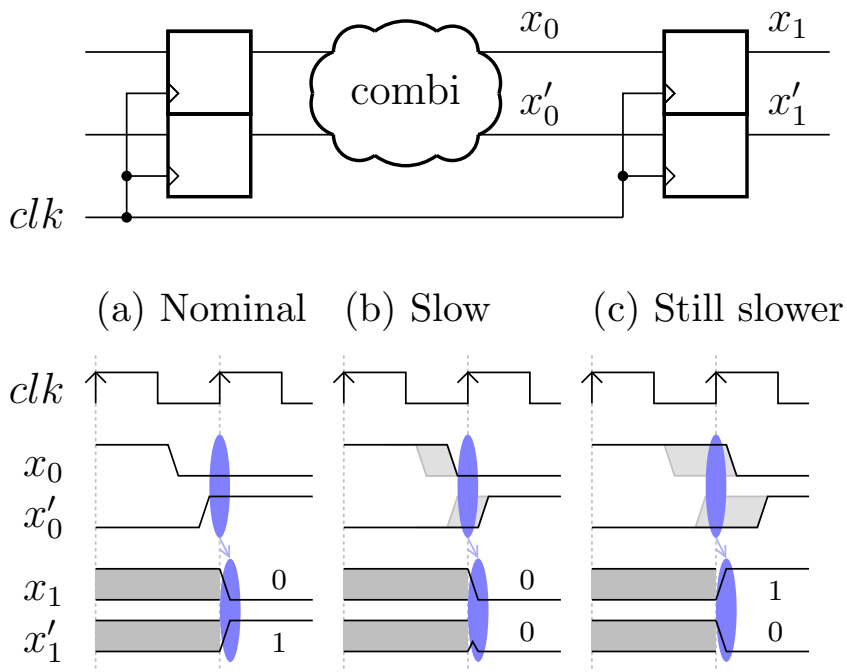


Figure 8.3: Illustration of setup time violation on a dual-rail circuit. (a): nominal case, the combinational logic evaluate before the rising edge of the clock; (b): slow case, the path x'_0 is violated; (c): still slower case, the two paths (x_0, x'_0) are violated, leading to a valid fault $(0,1) \rightarrow (1,0)$

powering, overclocking, or EM glitch), the combinational gates are made slower, then it can happen that the slowest net x'_0 does not reach its corresponding DFF timely. Hence the value of the DFF is $(x_1, x'_1) = (0,0)$ (see Fig. 8.3(b)), as intended in the PC-II countermeasure for one bit faults. Now, if the stress is further increased, both x_0 and x'_0 will be violated, which results in (x_1, x'_1) sampling a valid value, namely the previous value of (x_0, x'_0). Indeed, we recall that in a properly implemented PC-II netlist, there are no glitches, hence gates evaluate only when they have their final value (no intermediate values are computed). So, the situation represented in Fig. 8.3(c) causes a fault which overcomes PC-II countermeasure (with $t = k = 1$), at least if the previous value of (x_0, x'_0) is the complement of the new one, which happens in average with probability 1/2. We also mention that, contrary to some secure logic styles such as [207], there is no precharge between evaluations in PC-II. This is why the previous value of a combinational gate is always a licit value. Moreover, if the routing between x_0 and x_1 has the same duration as the routing between x'_0 and x'_1, then the situation depicted in Fig. 8.3(b) never happens, and only licit faults are produced, as in Fig. 8.3(c).

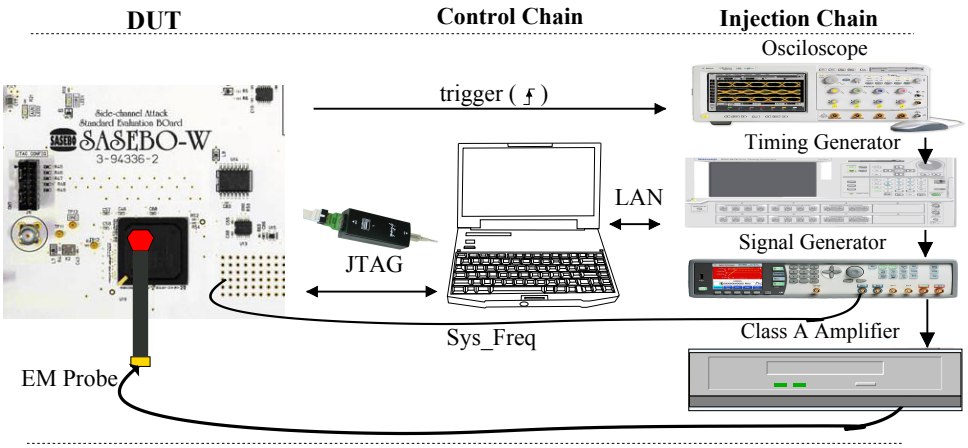


Figure 8.4: Picture of experimental setup for fault injection of SASEBO-W

8.4. Evaluation using faults

8.4.1. Experiment setup

We chose to evaluate PC-II against those faults: underfeeding, overclocking, and electromagnetic glitching. Regarding the underfeeding, only a simple power supply is necessary: it has a resolution of 500 μV. The fault is generated by gradually decreasing the supply voltage of the device under test (DUT) until faults occur. The same procedure is used for overclocking. We gradually increase the clock of the DUT until faults occur. Our setup (signal generator) allows to tune finely the frequency (resolution of 1 ps). Eventually, an amplifier allows us to injection EM faults. The full setup is presented in Fig. 10.7.

We notice that when faults are injected carelessly, the ciphertext is indeed fully zeroized. This validates practically our implementation and the principle of *infection by pairs of zeros* of PC-II. Now, in the rest of this section, our aim is to check the vulnerabilities identified in the previous section 8.3. This requires to fault only one path (namely, the critical path), hence we geared our fault injection experiments towards the most gentle stress as possible, so as to avoid faults with too high a multiplicity.

8.4.2. Internal and online debug of fault effects

For experiments, we found it useful to target the design to Spartan 6: indeed, Spartan 6 FPGA is packaged in frontside (FG). Therefore it is easier to perform EM injection on Spartan 6 FPGA than others FPGA which are packaged in backside (FF).

The traditional method to characterize faults is *indirect*: an exhaustive study of faults within a model is done (as in the differential fault analysis, or DFA, of Piret and Quisquater [208, 209]). Here, in order to extract *directly* the fault models, we implement the Integrated Logic Analyzer (ILA). We checked that the insertion of the analyzer does not impact the maximal working frequency of the DUT.

The implemented ILA has the following properties:

- ILA probes 96×4 nodes, i.e., the internal SIMON state, the dual-rail state[3], the key and the corresponding dual-rail key;
- ILA dump is triggered by the start signal of the SIMON encryption;
- ILA dumps 64 states after being triggered;
- ILA frequency is the same as the DUT frequency;
- ILA uses 17 of the 268 available BRAM blocks of the Spartan 6 FPGA and JTAG boundary scan chain to store dumped values;
- ILA can operate at a higher frequency than the DUT (checked).

It is non-intrusive, in that it is plugged on the design without interfering with it. We requested ILA to record and then dump the consecutive values of the state of SIMON. ILA is controlled by ChipScope Pro debugger interface. In the sequel, we use ChipScope to dump the execution traces, under VCD[4] format. Thanks to the tool `vcdtowlf` provided by Mentor Graphics, we turn this VCD file into a WLF[5] file. Then we use Mentor Graphics ModelSim to open and analyze the WLF traces (those will be shown in Fig. 8.5 and 8.6).

8.4.3. Results

Power supply fault

The first injection is to modify the power supply of the DUT (SIMON 96/96). The nominal value of power supply of FPGA is 1.00 V. By decreasing this value, we observe that under 0.68 V, the circuit produces incorrect results. However, around 0.68 – 0.67 V, the value of the ciphertext is different from zero. Then under 0.67 V, the ciphertext computed by SIMON is zero.

[3] To help the debug, we integrated the demasking $(a_1, a_2, a_3) \mapsto a = a_1 \oplus a_2 \oplus a_3$ in the design, for each bit a of the datapath. This is not secure from a side-channel point of view, but does not impact our experiments which are rather concerned by fault injection attacks.

[4] VCD is short for Values Change Dump; it is the IEEE Standard 1364-1995. One example of usage in security is provided in [210], where unbalances in a dual-rail netlist are analyzed.

[5] WLF is short for Waveform Log Format; it is the ModelSim default format for simulation results.

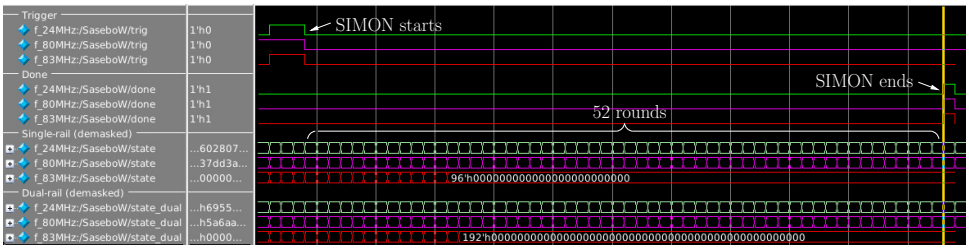


Figure 8.5: One experimental results of SIMON running at 24 MHz, and overclocked at 80 MHz and 83 MHz

Overclocking

Overclocking with fixed plaintext We increase the clock frequency of the SIMON block cipher from 24 MHz until the ciphertext gets erroneous. We choose a plaintext pt = 0x2072616c6c69702065687420 and a key k = 0x0d0c0b0a0908050403020100. The resulting WLF waveforms as dumped by ChipScope are joined, and are represented in Fig. 8.5. Three acquisitions are taken:

1. the first one for f = 24 MHz (in green),
2. the second one for f = 80 MHz (in pink), and
3. the last one for f = 83 MHz (in red).

The correct ciphertext is 0x602807a462b469063d8ff082. The three computations trigger at the same time. In Fig. 8.5, the scale is given round by round (it is not a time scale).

At 80 MHz, we observe in Fig. 8.5, that a fault is created in the circuit. We recall that the synthesis report (Tab. 8.3) announced a maximal frequency of 77 MHz, but the FPGA still functions normally until < 80 MHz. Nonetheless, at 80 MHz, the computed ciphertext is different from 0: it is 0x37dd3ac20989b9360ebef34a. By carrying on increasing the frequency value of frequency until 83 MHz, the ciphertext stays at 0x37dd3ac20989b9360ebef34a. Then, over 83 MHz, the ciphertext becomes 0 (as should happen in theory with PC-II netlists).

An explanation why at f = 80 MHz the ciphertext is not full-zero can be seen by doing a zoom on Fig. 8.5: we find that at the *third round*, the state starts to be incorrect.

For the reader's convenience, the state value is also given in Tab. 8.4. At the round three, the value of the valid state is 0xf64be72c5773b48da3938c84. For f = 80 MHz, the corrupted state is 0xf64be7285773b48da3938c84. The nibble c is changed in 8 at the eighth position of the state. In dual-rail, this corresponds to value a being changed to 9. Now, a represents $(1010)_2$, namely the two bits $(1,0)$ and $(1,0)$, whereas 9 means $(1001)_2$ in binary, which is a valid case in LUT6_2 truth

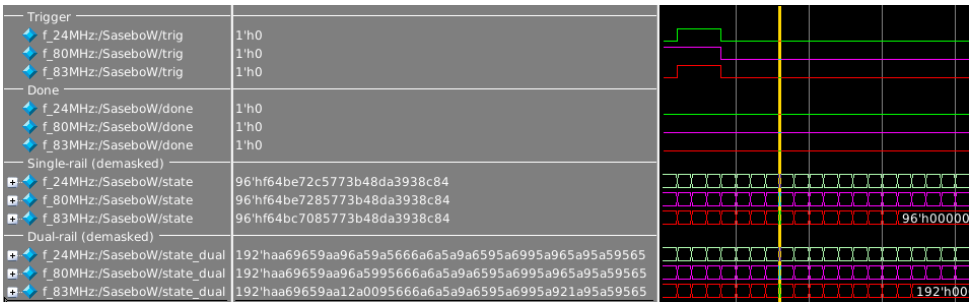


Figure 8.6: Zoom on Fig. 8.5, where we identify the first round (namely round #3) where a fault occurs

Table 8.4: State and dual-state differences at the third round.
Blue means "valid" error, such as $(0,1) \rightarrow (1,0)$, whereas
green means "single bit reset" error, such as $(0,1) \rightarrow (0,0)$.
(data extracted from Fig. 8.6)

f (MHz)	State (96-bit word)
24	0xf64be72c5773b48da3938c84
80	0xf64be7285773b48da3938c84
83	0xf64bc7085773b48da3938c84

f (MHz)	Dual-rail state (192-bit word)
24	0xaa69659aa96a59a5666a6a5a9a6595a6995a965a95a59565
80	0xaa69659aa96a5995666a6a5a9a6595a6995a965a95a59565
83	0xaa69659aa12a0095666a6a5a9a6595a6995a921a95a59565

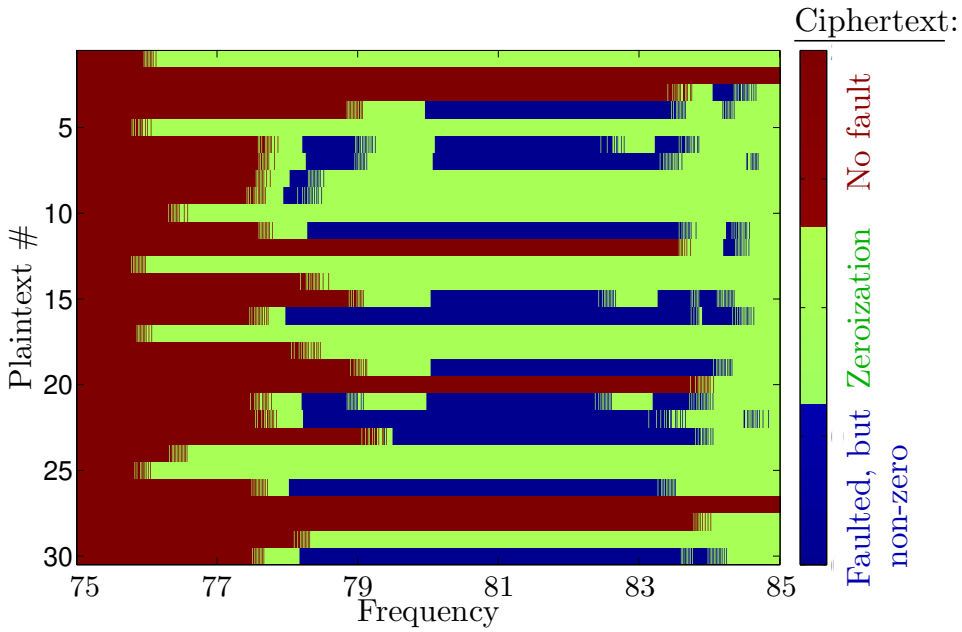


Figure 8.7: Effect of overclocking on 30 plaintexts for different frequencies

table (recall Tab. 8.2). Thus, this exemple illustrates the replacement of a licit value by another one, as in Fig. 8.3(c). So, on the critical path, there exists one LUT6_2 where a pair of dual-rail wires $(1,0)$ was changed in $(0,1)$, a valid value in dual-rail. Consequently the SIMON computes with a valid false state and it is possible to attack the circuit thanks to a DFA.

For $f = 83$ MHz, the first error in the state also occurs at third round. There, the faulted state takes the value 0xf64bc7085773b48da3938c84. The nibbles e, 2 and c at positions respectively 5, 7 and 8 become c, 0 and 8. In dual-rail, the numbers 1, 2, 0 are encoded in binary as $(0001)_2$, $(0010)_2$, $(0000)_2$. Those faults are single bit-flips. Thus, faulty states appear, and so a zeroization wave propagates. As can be seen in Fig. 8.5, the state reaches the value of zero at the round 11.

It is noteworthy that fault injections, if successful, can still be exploited despite the implementation is masked. Indeed, irrespective of the masking and the implementation details, a fault propagates until the output. The interested reader is referred to this paper [211] for more details.

Overclocking with varying plaintexts By changing the plaintext, the clock frequency when the circuit starts to be faulty changes, but remains around 80 MHz. Indeed, it is a well known fact that the critical path is data-dependent. In Fig. 8.7,

we present the effect of overclocking PC-II with 30 plaintexts. The experiment is performed using 1000 steps of 0.01 MHz. In total, the DUT frequency is changed from 75 MHz to 85 MHz. The red color in the Fig. 8.7 means that no fault occurred. The blue color means that the PC-II circuit is faulted with a non-zero ciphertext. Eventually, the green color means that the PC-II circuit is faulted with an output fully zeroized. We notice that, there is around 50% (16/30 plaintexts faulted with non-zeroization output) of probability to created an exploitable (not fully zero) fault, in a range of about 4 MHz (that is, between 79 and 83 MHz). In the other 50%, the $(0,0)$ values generated by the PC-II countermeasure absorb potential harmful $(0,1) \leftrightarrow (1,0)$ faults. Still, with roughly a probability of $1/2$, bypassing PC-II is possible provided the fault injection is controlled precisely enough.

We raised the clock frequency much beyond 85 MHz. Interestingly, the result remains zeroized for all the bits. One could have imagined that for a high enough frequency, only valid bits from the previous state are sampled. But apparently, there is consistently at least one gate which transitions, hence with output equal to $(0,0)$. In the extreme case, the clock would be so fast that the previous state would be sampled verbatim. But this would have no effect, since the round counter would also stall during this fast clock period. At least, this situation is harmless if the computation path and the control logic are both implemented in PC-II. If only the computation is implemented in PC-II and the logic in regular (faster) logic, then it would be possible to skip [212] or add [213] rounds.

Electromagnetic injections

The SIMON 96/96 in PC-II under EM injections behaves similarly as in the case of overclocking.

8.4.4. Discussion

The faults obtained by overclocking and EM injection are similar on our platform. We can deduce that we caused *delay faults*. According to the delays measured by ChipScope, we have:

- an average delay between true (y) and false (y') wires slightly non-zero: −166 ps, and
- with a fairly large standard deviation of 501 ps.

The distribution is represented in Fig. 8.8. It can clearly be seen that most pairs are balanced (the delay difference is peaked around zero), while some outliers exist, with a large delay difference, probably due to routing unbalances.

When the injected field is very strong, EM injection can also cause *sampling faults* [214],

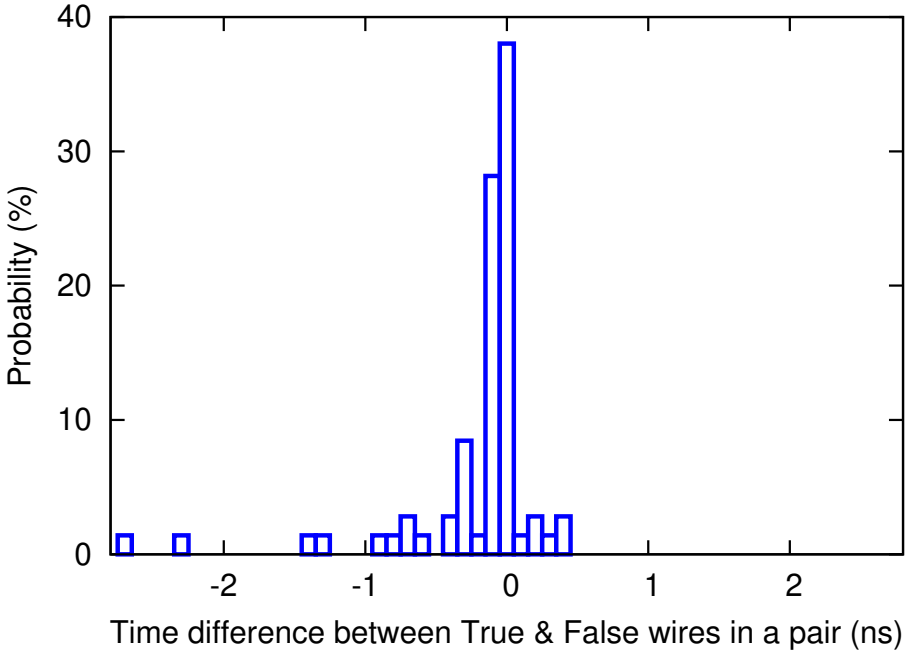


Figure 8.8: Statistics of delays unbalance, per step of 100 ps

where the DFFs are perturbed while they sample, precisely by having the EM pulse happen timely with the clock rising edge. Such fault model would not allow correlated faults, hence would not harm private circuits II. But the global timing faults are harmful: it is thus important, for the faults to be exploitable, to exercise only gentle stress: overclocking must have an effect, but not too strong, otherwise too many timing violations occur, amongst them zeroization can occur and wash the licit faulted values. EM injection must not cause DFF malfunctions, only increase the delay in gates, hopefully touching first the critical path.

8.5. Conclusion and perspectives

This paper has shown the possibility to collect erroneous outputs from cryptographic circuits protected by the private circuits II countermeasure, allowing different sorts of attacks, such as the differential fault analysis.

We notice that ChipScope (or equivalently SignalTap in Altera) is a nice tool to investigate the effects of faults on circuits. We leverage on this tool, apparently for the first time, to determine exactly the fault models for overclocking.

As a perspective, we intend to take advantage of the zeroization process to test

other attack paths, such as fault sensibility analysis [215] (the stress level at which a fault occurs is data dependent). Let us remark that safe errors are a priori not possible, since in hardware, even values which do not impact the computation value at the logic level are protected. For instance, a multiplexer with the non-selected input at the invalid zero value will propagate an invalid zero value all the same. Obviously, the fault attacks we present applies to other protected circuits, namely dual-rail circuits (WDDL, MDPL, etc.).

Besides, a recent paper [196] has shown similarities between PC-II and *threshold implementation*, a logic style which is theoretically designed to withstand glitches. Practical validation would definitely make sense.

Acknowledgments

The authors a grateful to Laurent Sauvage for setting up the fault injection platform and valuable pieces of advice. We also thank Thibault Porteboeuf for interesting feedback exchanges about ChipScope setup under ISE.

9

Using Modular Extension to Provably Protect Edwards Curves Against Fault Attacks

Fault injection attacks are a real-world threat to cryptosystems, in particular asymmetric cryptography. In this paper, we focus on countermeasures which guarantee the integrity of the computation result, hence covering most existing and future fault attacks. Namely, we study the *modular extension* protection scheme in previously existing and newly contributed variants of the countermeasure on elliptic curve scalar multiplication (ECSM) algorithms. We find that an existing countermeasure is incorrect and we propose new "test-free" variant of the modular extension scheme that fixes it. We then formally prove the correctness and security of modular extension: specifically, the fault non-detection probability is inversely proportional to the security parameter. Finally, we implement an ECSM protected with test-free modular extension during the elliptic curve operation to evaluate the efficient of this method on Edwards and twisted Edwards curves.

9.1. Introduction

Properly used cryptography is a key building block for secure information exchange. Thus, implementation-level hacks must be considered seriously in addition to the threat of cyber-attacks. In particular, fault injection attacks target physical implementations of secured devices in order to induce exploitable errors.

Asymmetric cryptography Asymmetric cryptography addresses different needs such as key exchange and digital signature. RSA, Diffie-Hellman, and ElGamal have been used for decades, and elliptic curve cryptography (ECC) algorithms such as ECDSA [216] are more and more deployed. ECC pairing-based cryptography has recently been accelerated in practice and is thus becoming practical [217]. For example, the construction of "pairing-friendly" elliptic curves is an active subject [218]. Homomorphic encryption schemes are getting more practical and are progressively considered viable solutions for some real-world applications requiring strong privacy. All these algorithms use large numbers and take place in mathematical structures such as finite rings and fields, which enables powerful mathematical properties but also facilitates attacks.

Fault Attacks As put forward in the reference book on fault analysis in cryptography [219, Chp. 9], there are three main categories of fault attacks.
1) *Safe-error attacks* consist in testing whether an intermediate variable is dummy (usually introduced against simple power analysis [94]) or not, by faulting it and looking whether there is an effect on the final result.
2) *Cryptosystem parameter alterations* aim at weakening the algorithm in order to ease key extraction. For example [220], invalid-curve fault attacks consist in moving an ECC computation to a weaker curve, enabling the attacker to use cryptanalysis attacks exploiting the faulty outputs.
3) Finally, the most serious attacks belong to the *differential fault analysis* (DFA) category. Often the attack path consists in comparing correct and faulted outputs, like in the well-known BellCoRe attack on CRT-RSA (RSA sped up using the Chinese Remainder Theorem), or the sign-change fault attack on ECC.

The *BellCoRe attack* [221] on CRT-RSA introduced the concept of fault injection attacks. It is very powerful: faulting the computation even in a very random way yields almost certainly an exploitable result allowing to recover the secret primes of the RSA modulus $N = pq$.

The *sign-change attack* [222] on ECC consists in changing the sign of an intermediate elliptic curve point in the midst of an elliptic curve scalar multiplication (ECSM). The resulting faulted point is still on the curve so the fault is not detected by traditional point validation countermeasures. Such a fault can be achieved by for instance changing the sign in the double operation of the ECSM algorithm (line 3 of Alg. 5). If the fault injection occurs during the last iteration of the loop, then the final result $\widehat{Q} = [-2\sum_{i=1}^{n-1} k_i 2^{i-1}]P + k_0 P = -Q + 2k_0 P$, i.e., either $\widehat{Q} = -Q$ or $\widehat{Q} = -Q + 2P$ depending on k_0, which reveals the value of k_0 to the attacker. This process can be iterated to find the other bits of the scalar, and optimizations exist that trade-off between the number of necessary faulted results and the required exhaustive search.

Algorithm 5: Double-and-add left-to-right scalar multiplication on elliptic curve $\mathcal{E}$.

Input : $P \in \mathcal{E}$, $k = \sum_{i=0}^{n-1} k_i 2^i$ (n is the scalar size in bits, where $k_i \in \{0,1\}$)
Output : $[k]P$

```
1 Q ← O                                   ▷ O is the point at infinity
2 for i ← n − 1 down to 0 do
3 |   Q ← 2Q                              ▷ ECDBL
4 |   if k_i = 1 then Q ← Q + P            ▷ ECADD
5 end
6 return Q
```

Figure 9.1: Sketch of the principle of *modular extension*.

Both RSA and ECC algorithms continue to be the target of **many** new fault injection attacks: see [223–227] just for some 2014 papers. Besides, this topic is emerging and other new fault attacks will appear sooner or later. Hence, the need for efficient and practical generic countermeasures against fault attacks is obvious. David Wagner from UC Berkeley concurs in [228]: *"It is a fascinating research problem to establish a principled foundation for security against fault attacks and to find schemes that can be proven secure within that framework."*

Countermeasures Verifications compatible with mathematical structures can be applied either at computational or at algorithmic level.

Algorithmic protections have been proposed by Giraud [229] (and many others [230–232]) for CRT-RSA, which naturally transpose to ECC, as shown in [233]. These protections are implementation specific (e.g., depend on the chosen exponentiation algorithm) and are thus difficult to automate, requiring specialized engineering skills.

Computational protections have been pioneered by Shamir in [234] using *modular extension*, initially to protect CRT-RSA. The idea is to carry out the same computation in two different algebraic structures allowing to check the computation before disclosing its result. For example protecting a computation in $\mathbb{F}_p$ consists in carrying out the same computation in $\mathbb{Z}_{pr}$ and $\mathbb{F}_r$ ($\mathbb{Z}_{pr}$ is the direct product of $\mathbb{F}_p$ and $\mathbb{F}_r$), where r is a small number ($r \ll p$); the computation in $\mathbb{Z}_{pr}$ must match that of $\mathbb{F}_r$ when reduced modulo r, if not an error is returned, otherwise the result in $\mathbb{Z}_{pr}$ is reduced modulo p and returned. The principle of modular extension is sketched in Fig. 9.1. This method operates at low level (integer arithmetic), thereby enabling countermeasures (and optimizations) to be added on top of it. They are thus easily maintained, which explains why this method is quite popular. Indeed, there is a wealth of variants for CRT-RSA stemming from this idea [235–240], as well as a few proofs-of-concept transposing it to ECC [222, 241, 242]. Despite the nonexistence

of literature, the same idea could apply to post-quantum code-based cryptography, pairing, and homomorphic computation for instance. Therefore, our paper focuses on computational countermeasures.

On the one hand, the variety of CRT-RSA countermeasures shows that fault attacks are a threat that is taken seriously by both the academic and the industrial communities. On the other hand, it bears witness to the artisanal way these countermeasures were put together. Indeed, the absence of formal security claims and of proofs added to the necessity of writing implementations by hand results in many weaknesses in existing countermeasures and thus in many attempts to create better ones.

Contributions The countermeasure described in [241] can be applied only on Weierstrass curve, and the overhead computation is 48% for curve with parameters on 256 bits. The main disadvantage of this countermeasure is the need for point testing during the addition and doubling operations. These tests can differ in $\mathbb{Z}_{pr}$ and $\mathbb{F}_r$, hence a *loose security proof*, because the computation in $\mathbb{F}_r$ can be trapped in the point at infinity.

In this paper, we take advantage of the speed-up record on ECDSA computation using the twisted Edwards curve `Ed25519` [138] coded with NaCl cryptolibrary [243]. We propose a new countermeasure against faults injection based on modulus extension with only one "test-free" addition operation using *complete* and *unified* formulas of addition point on Edwards and twisted Edwards curves. This allows for a *synchronized* computation in $\mathbb{F}_p$ and $\mathbb{F}_r$ while computing in $\mathbb{Z}_{pr}$, as opposed to the state-of-the-art countermeasures, such as [222, 241, 242]. Our countermeasure is new insofar as we give explicit conditions on the prime r: they happen to be easily met in the case of Edwards curves (see Sec. 9.6.1), whereas they restrict the number of possible r to a little number of values for the popular twisted Edwards curves (see Sec. 9.6.2). The overhead computation of this countermeasure is 28% for Edwards curve and 39% for twisted Edwards curve on 32-bit processors, such as an ARM Cortex-M4.

Outline The rest of the paper is organized as follows. Section 9.2 described the details of the existing countermeasure for ECC. Section 9.3 is the theoretical analysis of our new countermeasure using the modular extension with test-free. Section 9.4 described the mathematical background on the Edwards curves. The description of our countermeasure to make the modular extension without test in the elliptic curve operation is in Section 9.5. Section 9.6 explains the overhead of computation and some examples of our countermeasure. Section 13.7 concludes.

9.2. Existing Countermeasures for ECC

Countermeasures against fault injection attacks have been proposed for elliptic curve computations, but they are actually incorrect. For example, in [222], Blömer, Otto, and Seifert (BOS) propose a countermeasure based on the modular extension idea of Shamir for CRT-RSA [234]. The problem is that the modular extension scheme cannot actually be applied as is to Weierstrass elliptic curve, because the tests in the ECDBL and ECADD operations are not true at the same times for the computation in $\mathbb{Z}_{pr}$ and the one in $\mathbb{F}_r$, which breaks the scheme and will yield false negatives. This behavior can be a *serious security issue* as it reveals information about the inputs.

In 2010 Joye patented [242] essentially the same countermeasure except it uses $\mathbb{F}_{r^2}$ and $\mathbb{Z}_{pr^2}$ instead of $\mathbb{F}_r$ and $\mathbb{Z}_{pr}$, which does not address the raised issues.

In [241], Baek and Vasyltsov (BV) propose a countermeasure based on modular extension and point verification. The particularity of this countermeasure is that instead of computing a sibling ECSM on a smaller curve $\mathcal{E}(\mathbb{F}_r)$ to compare with its redundant counterpart over $\mathcal{E}(\mathbb{Z}_{pr})$, it only checks whether the point obtained by reducing the result $\mathcal{E}(\mathbb{Z}_{pr})$ modulo r is on the $\mathcal{E}(\mathbb{F}_r)$ curve (i.e., whether it satisfies the curve equations modulo r). Because of that, BV does not suffer from BOS problem. However, the correctness of BV comes with a drawback: indeed, faults may go undetected if they happen before $\mathcal{O}$ (the point at infinity) is reached in the computation modulo r as the intermediate point quickly tends to $(0:0:0)$ in projective coordinates and stays there until the end.

It is actually possible to get the best of both world: what is needed is BOS approach (i.e., pure modular extension scheme) but without the problematic tests. Luckily, Edwards curves allow to perform ECC without tests thanks to a complete addition law, as will be detailed in Sec. 9.4. But before, we will formally analyze the security of the modular extension scheme when the implementation is test-free.

9.3. Security Analysis of Modular Extension

Definition 15 (Fault model) *We consider an attacker who can fault data by randomizing or zeroing any intermediate variable, and fault code by skipping any number of consecutive instructions.*

Remark The three fault models have been described several times in the literature. For example, randomizing faults are discussed in [221], zeroing faults in [244], and instruction skip faults in [245].

Definition 16 (Attack order) *We call order of the attack the number of faults (in the sense of definition 15) injected during the target execution.*

In the rest of this section, we focus mainly on the resistance to first-order attacks on data.

Definition 17 (Secure algorithm) *An algorithm is said secure if it is correct and if it either returns the right result or an* error *constant when faults have been injected, with an overwhelming probability.*

[Security of test-free modular extension] Test-free algorithms protected using the modular extension technique, are secure as per definition 17. In particular, the probability of non-detection is inversely proportional to the security parameter r. *Faulted results are polynomials of faults.* The result of an asymmetric cryptography computation can be written as a function of a subset of the intermediate variables, plus some inputs if the intermediate variables do not suffice to finish the computation. We are interested in the expression of the result as a function of the intermediate variables which are the target of a transient or permanent fault injection. We give the formal name $\widehat{x}$ to any faulted variable x. For convenience, we denote them by $\widehat{x_i}$, $1 \leq i \leq n$, where $n \geq 1$ is the number of injected faults. The result consists in additions, subtractions, and multiplications of those formal variables (and inputs). Such expression is a multivariate polynomial. If the inputs are fixed, then the polynomial has only n formal variables. We call it $P(\widehat{x_1}, \ldots, \widehat{x_n})$. For now, let us assume that $n = 1$, i.e., that we face a single fault. Then P is a monovariate polynomial. Its degree d is the multiplicative depth of $\widehat{x_1}$ in the result.

A fault is not detected if and only if $P(\widehat{x_1}) = P(x_1) \mod r$, whereas $P(\widehat{x_1}) \neq P(x_1) \mod p$. Notice that the latter condition is superfluous insofar since if it is negated then the effect of the fault does not alter the result in $\mathbb{F}_p$.

Non-detection probability is inversely proportional to r. As the faulted variable $\widehat{x_1}$ can take any value in $\mathbb{Z}_{pr}$, the non-detection probability $\mathbb{P}_{\text{n.d.}}$ is given by:

$$\begin{aligned}\mathbb{P}_{\text{n.d.}} &= \frac{1}{pr-1} \cdot \sum_{\widehat{x_1} \in \mathbb{Z}_{pr} \setminus \{x_1\}} \mathbb{1}_{P(\widehat{x_1}) = P(x_1) \mod r} \\ &= \frac{1}{pr-1} \cdot \left(-1 + p \sum_{\widehat{x_1}=0}^{r-1} \mathbb{1}_{P(\widehat{x_1}) = P(x_1) \mod r} \right). \end{aligned} \tag{9.1}$$

Here, $\mathbb{1}_{\text{condition}}$ is an indicator function: it is equal to 1 (resp. 0) if the condition is true (resp. false).

Let $\widehat{x_1} \in \mathbb{Z}_r$, if $P(\widehat{x_1}) = P(x_1) \mod r$, then $\widehat{x_1}$ is a root of the polynomial $\Delta P(\widehat{x_1}) = P(\widehat{x_1}) - P(x_1)$ in $\mathbb{Z}_r$. We denote by $\#\text{roots}(\Delta P)$ the number of roots of ΔP over $\mathbb{Z}_r$. Thus (9.1) computes $(p \times \#\text{roots}(\Delta P) - 1)/(pr-1) \approx \#\text{roots}(\Delta P)/r$.

Study of the proportionality constant. A priori, bounds on this value are broad since $\#\text{roots}(\Delta P)$ can be as high as the degree d of ΔP in $\mathbb{Z}_r$, i.e., $\min(d, r-1)$. However,

in practice, ΔP looks like a random polynomial over the finite field $\mathbb{Z}_r$, for several reasons:

- inputs are random numbers in most cryptographic algorithms, such as probabilistic signature schemes,

- the coefficients of ΔP in $\mathbb{Z}_r$ are randomized due to the reduction modulo r. In such case, the number of roots is very small, despite the possibility of d being large. See for instance [246] for a proof that the number of roots tends to 1 as $r \to \infty$. Interestingly, random polynomials are still friable (i.e., they are clearly not irreducible) in average, but most factors of degree greater than one happen not to have roots in $\mathbb{Z}_r$. Thus, we have $\mathbb{P}_{\text{n.d.}} \gtrsim \frac{1}{r}$, meaning that $\mathbb{P}_{\text{n.d.}} \geq \frac{1}{r}$ but is close to $\frac{1}{r}$. A more detailed study of the theoretical upper bound on the number of roots is available in [247, Sec. A].

The same law applies to multiple faults. In the case of multiple faults ($n > 1$), then the probability of non-detection generalizes to:

$$\begin{aligned}
\mathbb{P}_{\text{n.d.}} &= \frac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_1},\ldots,\widehat{x_n} \in \mathbb{Z}_{pr}\setminus\{x_1\}\times\ldots\times\mathbb{Z}_{pr}\setminus\{x_n\}} \\
&\quad \cdot \mathbb{1}_{P(\widehat{x_1},\ldots,\widehat{x_n})=P(x_1,\ldots,x_n) \bmod r} \\
&= \frac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_2},\ldots,\widehat{x_n} \in \prod_{i=2}^{n} \mathbb{Z}_{pr}\setminus\{x_i\}} \\
&\quad \cdot \left[\sum_{\widehat{x_1} \in \mathbb{Z}_{pr}\setminus\{x_1\}} \mathbb{1}_{P(\widehat{x_1},\ldots,\widehat{x_n})=P(x_1,\ldots,x_n) \bmod r} \right] \\
&= \frac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_2},\ldots,\widehat{x_n} \in \prod_{i=2}^{n} \mathbb{Z}_{pr}\setminus\{x_i\}} \\
&\quad \cdot [p \times \#\text{roots}(\Delta P) - 1] \\
&= \frac{1}{(pr-1)^n} \cdot (pr-1)^{n-1} [p \times \#\text{roots}(\Delta P) - 1] \\
&= \frac{p \times \#\text{roots}(\Delta P) - 1}{pr-1}.
\end{aligned}$$

Therefore, the probability not to detect a fault when $n > 1$ is identical to that for $n = 1$. Thus, we also have $\mathbb{P}_{\text{n.d.}} \approx \frac{1}{r}$ in the case of multiple faults of the intermediate variables[1].

[1] Note that this study does not take correlated faults into account.

9.4. Edwards Curves over large-characteristic fields

In mathematics, the Edwards curves are a family of elliptic curves studied by Harold M. Edwards in 2007 [248]. Technically, an Edwards curve is not elliptic, because it has singularities; but resolving those singularities produces an elliptic curve. The concept of elliptic curves over finite fields is widely used in elliptic curve cryptography. Applications of Edwards curves to cryptography were developed by Bernstein and Lange: they pointed out several advantages of the Edwards form in comparison to the more well known Weierstrass form.

9.4.1. Edwards curves

Definition 18 (Edwards curves) *On the finite field $\mathbb{F}_p$ with p a prime number, an elliptic curve in Edwards form has parameters c, d in the finite field $\mathbb{F}_p$ and coordinates (x, y) satisfying the following equation:*

$$x^2 + y^2 = c^2(1 + dx^2y^2), \text{ with } cd(1 - c^4d) \neq 0. \tag{9.2}$$

The main advantage to use the Edwards curves is that addition formulas are *unified*: doubling and addition operations are the same. Affine unified addition formula is $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$$x_3 = \frac{x_1y_2 + y_1x_2}{c(1 + dx_1x_2y_1y_2)} \text{and } y_3 = \frac{y_1y_2 - x_1x_2}{c(1 - dx_1x_2y_1y_2)}. \tag{9.3}$$

The affine negation formula is as expected: $-(x_1, y_1) = (-x_1, y_1)$.

The neutral element of the curve is the point $(0, c)$. Contrary to Weierstrass curves, this point is not special (there is no abstract *"point at infinity"*), but verifies the curve equation. The point $(0, -c)$ has order 2. The points $(c, 0)$ and $(-c, 0)$ have order 4.

Bernstein and Lange [249] proved that if d is not a square in $\mathbb{F}_p$ then the unified addition law is *complete*. This means that the addition formula is valid for all points, with no exception. That is one of the advantages of Edwards curves over Weierstrass curves in which the addition law is *not complete*: a *complete* addition law provides some resistance to side-channel attacks.

9.4.2. Twisted Edwards curves

Twisted Edwards curves are a generalization of Edwards curves [250].

Definition 19 (Twisted Edwards curves) *Let p a prime number. On the finite field*

$\mathbb{F}_p$, *an elliptic curve in twisted Edwards form has parameters* a, d *in the finite field* $\mathbb{F}_p$ *and coordinates* (x, y) *satisfying the following equation:*

$$ax^2 + y^2 = 1 + dx^2y^2, \text{ with } ad(a-d) \neq 0. \tag{9.4}$$

Like Edwards curves, the addition formulas are unified. Affine unified addition formula is $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \text{and } y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}. \tag{9.5}$$

The neutral element is $(0, 1)$. Affine negation formula is natural: $-(x_1, y_1) = (-x_1, y_1)$.

Addition law is *complete* if a is a square and d is a non-square [249].

9.5. Practical Study

On Edwards curves and twisted Edwards curves, the addition law is *complete*: addition formulas work for all pairs of input points. In particular, there is no troublesome point at infinity. Another advantage of Edwards curve is the atomicity of the formula doubling and adding and the constant time to protect the classical Side-Channel Attack. The addition law is *unified*, meaning that there are no test to verify if the two input points are equal, opposite or different. To be more efficient, we use the unified projective coordinates to the addition law ECADD-complete-unified named "add-2007-bl-2" on [251] or on [249, Sec. 4, page 9].

The ECSM with modular extension protection using complete unified addition formulas is given in Alg. 6. The first phase can compute offline, because find r verifies the lemmas 9.5.1 and 9.5.2 is not trivial. The second phase is composed by two ECSM computation online. The first ECSM computation consists in multiplying the point P with the scalar k on the ring $\mathbb{Z}_{pr}$ using the parameters defined later in this section by the proprieties 9.5.1 or 9.5.2. The second ECSM computation is the multiplication of the point P' with the scalar k on the *small* curve over $\mathbb{F}_r$ using the parameters defined in the lemmas 9.5.1 or 9.5.2. It is worthwhile to note that these two ECSM share the same code (see. Alg. 7).

Given an elliptic curve over $\mathbb{F}_p$ and a point (x_G, y_G), we define by λ the multiple of p to add when the *point on curve* equation is plunged from $\mathbb{F}_p$ to $\mathbb{Z}$. Formally,

Definition 20 (Parameter λ for Edwards curves) *Given an Edwards elliptic curve of equation* (9.2)*, the parameter* λ *is the integer satisfying the relationship in* $\mathbb{Z}$*:* $x_G^2 + y_G^2 = c^2(1 + dx_G^2y_G^2) + \lambda p$.

Definition 21 (Param. λ for twisted Edwards curves) *Given a twisted Edwards el-*

Algorithm 6: ECSM with modular extension protection using complete unified addition formulas.

Input : $P \in \mathcal{E}(\mathbb{F}_p)$, $k \in \mathbb{Z}$
Output : $Q = [k]P \in \mathcal{E}(\mathbb{F}_p)$

Offline phase

Edwards Curves:

Compute $\lambda p = x_G^2 + y_G^2 - c^2(1 + cx^2y^2)$
repeat
  Choose a random prime $r < p$
  Compute $x'_G = X_G \mod r$
  Compute $y'_G = y_G \mod r$
  Compute $c' = c^2 + \lambda p \mod r$
  Compute $d' = \frac{dc^2}{c^2+\lambda p} \mod r$
until $x'_G \neq 0$ *and* $y'_G \neq 0$ *and* $c'd'(1 - c'^4d') \neq 0$ *and* c' *a square and* d' *a no-square*
▷ r verifies the lemma 9.5.1

Twisted Edwards Curves:

Compute $\lambda = (1 + dx_G^2y_G^2 - ax_G^2 + y_G^2) \div p$
Find all the factor r smaller than p of λ
for *each factor* r **do**
  Compute $x'_G = x_G \mod r$
  Compute $y'_G = y_G \mod r$
  Compute $a' = a \mod r$
  Compute $d' = d \mod r$
  if $x'_G \neq 0$ *and* $y'_G \neq 0$ *and* $a'd'(a' - d') \neq 0$ *and* a' *a square and* d' *a no-square* **then**
    break ▷ r verifies the lemma 9.5.2
  else
    r does not work
  end
end

Determine the small curve $\mathcal{E}(\mathbb{F}_r)$ with parameter c' (or a') and d' , and a point $P'(x'_G, y'_G)$ is on that curve.
Determine the combined curve $\mathcal{E}(\mathbb{Z}_{pr})$ with parameter $C = CRT(c, c')$ (or $A = CRT(a, a')$) and $D = CRT(d, d')$ ▷ using properties 9.5.1 and 9.5.2.

Online phase

$(X_{pr} : Y_{pr} : Z_{pr}) = \text{ECSM}(P, k, \mathcal{E}(\mathbb{Z}_{pr}))$ ▷ without test on the point and on the scalar value
$(X_r : Y_r : Z_r) = \text{ECSM}(P', k, \mathcal{E}(\mathbb{F}_r))$ ▷ without test on the point and on the scalar value
if $(X_{pr} \bmod r : Y_{pr} \bmod r : Z_{pr} \bmod r) = (X_r : Y_r : Z_r)$ **then**
  return $(X_{pr} \bmod p : Y_{pr} \bmod p : Z_{pr} \bmod p)$
else
  return error
end

liptic curve of equation (9.4)*, the parameter* λ *is the integer satisfying the following relationship in* $\mathbb{Z}$*:* $ax_G^2 + y_G^2 = 1 + dx_G^2 y_G^2 + \lambda p$.

9.5.1. Edwards curves

Let p be a prime and an Edwards curve over $\mathbb{F}_p$ as per definition 18, parameterized by c, d. Let λ as per definition 20.

Let r be a prime number $r < p$, such that $c^2 + \lambda p$ is a non-zero square in $\mathbb{F}_r$, $x_G \bmod r \neq 0$ and $y_G \bmod r \neq 0$. The set of points which satisfy $\mathcal{E}_r : x^2 + y^2 = c'^2(1 + d'x^2y^2) \mod r$ with $c'^2 = c^2 + \lambda p \mod r$ and $d' = \frac{dc^2}{c^2+\lambda p} \mod r$ is an Edwards curve, generated by the point $(x'_G, y'_G) = (x_G \bmod r, y_G \bmod r)$.

If the parameters c' and d' satisfy $c'd'(1 - c'^4 d') \neq 0$ and d' is not a square in the finite field $\mathbb{F}_r$, then the ECSM computation on this *small* Edwards curve $\mathcal{E}(\mathbb{F}_r)$ is complete, i.e., can be computed without point or scalar conditional tests. The detail of the proof is in the online version [247, Lemma 1 Sec 5.1].

For the purpose of the modular extension countermeasure, we extend the notion of Edwards curve to rings[2] (such as $\mathbb{Z}_{pr}$).

Let an Edwards curve defined on $\mathbb{F}_p$ with the parameters c, d and the point (x_G, y_G). If a random number r verifying the lemma 9.5.1 can be found to define the Edwards curve $\mathcal{E}(\mathbb{F}_r)$ with the parameters c', d', then $C = CRT(c, c')$ and $D = CRT(d, d')$ are the parameters of an Edwards elliptic curve over the rings $\mathbb{Z}_{pr}$. This curve parameters permit to detect a fault thanks to the comparison at line 15 in the Algorithm 6. We introduce the following notations:

- We denote by Pt_p with p in index a point named Pt computed on the $\mathcal{E}(\mathbb{F}_p)$;
- We denote by Pt_r with r in index a point named Pt computed on the $\mathcal{E}(\mathbb{F}_r)$;
- We denote by Pt_{pr} with pr in index a point named Pt computed on the $\mathcal{E}(\mathbb{Z}_{pr})$.

The input value of the two ECSMs verify the equality using the projective coordinates, because we have as input (x_G, y_G) for the combined curve and (x'_G, y'_G) for the *small* curve:

$$\begin{cases} X\text{- coordinate: } x'_G &= x_G & \bmod r, \\ Y\text{- coordinate: } y'_G &= y_G & \bmod r, \\ Z\text{- coordinate: } 1 &= 1 & \bmod r. \end{cases} \tag{9.6}$$

[2] Similar idea can be found in [222, 241, 242]; we explicit it here for the article to be self-contained.

The ECSM computation over the combined curve on the ring extension $\mathbb{Z}_{pr}$ and the *small* curve over finite field $\mathbb{F}_r$ do consist in the same sequence of addition operations (ECADD-complete-unified).

Let P_{pr} and P_r be two points such that $X_{P_r} = X_{P_{pr}} \mod r, Y_{P_r} = Y_{P_{pr}} \mod r, Z_{P_r} = Z_{P_{pr}} \mod r$. Let Q_{pr} and Q_r be two points such that $X_{Q_r} = X_{Q_{pr}} \mod r, Y_{Q_r} = Y_{Q_{pr}} \mod r, Z_{Q_r} = Z_{Q_{pr}} \mod r$.

We compute R_r the result of ECADD-complete-unified between P_r and Q_r over $\mathcal{E}(\mathbb{F}_r)$, and R_{pr} the result of ECADD-complete-unified between P_{pr} and Q_{pr} over $\mathcal{E}(\mathbb{Z}_{pr})$.

The computation of the projective coordinates of R_{pr} is composed by addition, subtraction, multiplication over the ring $\mathbb{Z}_{pr}$ using the projective coordinates of P_{pr} and Q_{pr} and the two curve parameters C and D.

The computation of the projective coordinates of R_{pr} is composed by addition, subtraction, multiplication over the ring $\mathbb{Z}_{pr}$ using the projective coordinates of P_r and Q_r and the two curve parameters c' and d'.

By construction $C = CRT(c, c')$ and $D = CRT(d, d')$, we have $C \bmod r = c'$ and $D \bmod r = d'$, so the projective coordinates of R_{pr} are pairwise equal modulo r with the projective coordinates of R_r.

As the ECADD-complete-unified operation conserves the equality of the point coordinates value modulo r, we conclude that the ECSM computation conserves the equality of the point coordinates value modulo r between the computation over the ring extension and over the finite field $\mathbb{F}_r$.

9.5.2. Twisted Edwards curves

If a, d, p, λ verify the conditions defined in definition 21, then if we can choose a prime factor r of λp such that $x_G \bmod r \neq 0$ and such that the point $(x'_G, y'_G) = (x_G \bmod r, y_G \bmod r)$ generates the curve $\mathcal{E}(\mathbb{F}_r) : a'x^2 + y^2 = 1 + d'x^2y^2$ where $a' = a \bmod r$ and $d' = d \bmod r$.

If the parameters a' and d' satisfy $a'd'(a' - d') \neq 0$, a' is a square and d' is a non-square in the finite field $\mathbb{F}_r$, then the ECSM computation on this *small* twisted Edwards curve $\mathcal{E}(\mathbb{F}_r)$ requires no point and scalar tests. The detail of the proof is in the online version [247, Lemma 2 Sec 5.2]. For the purpose of the modular extension countermeasure depicted in Alg. 6, we extend the notion of twisted Edwards curve to ringsnote1 (such as $\mathbb{Z}_{pr}$). Let a twisted Edwards curve defined on $\mathbb{F}_p$ with the parameters a, d and the point (x_G, y_G). If a random number r verifying the lemma 9.5.2 can be found to define the twisted Edwards curve $\mathcal{E}(\mathbb{F}_r)$ with the

parameters a', d', then $A = a$ and $D = d$ are the parameters of a twisted Edwards curve over the ring $\mathbb{Z}_{pr}$.

If $x_G \bmod r \neq 0$ then the point (x'_G, y'_G) is not the point at infinity. So, this point (x'_G, y'_G) is a generator of a non-trivial subgroup of the elliptic curve $\mathcal{E}(\mathbb{F}_r)$.

This curve parameters permit to detect a fault with the comparison at line 15 in the Algorithm 6. The input value of the two ECSM verify the equality using the projective coordinates, because we have as input (x_G, y_G) for the combined curve and (x'_G, y'_G) for the *small* curve, as described previously in Eqn. (9.6).

The ECSM computation over the combined curve on the ring extension $\mathbb{Z}_{pr}$ and the *small* curve over finite field $\mathbb{F}_r$ consist in the same sequence of addition operations (ECADD-complete-unified). Namely, the sequence is given in Alg. 7, where $\mathcal{E}$ is either $\mathcal{E}(\mathbb{Z}_{pr})$ or $\mathcal{E}(\mathbb{F}_r)$.

By construction $A = a, D = d$ and we have $a \bmod r = a'$ and $d \bmod r = d'$, so the projective coordinates of R_{pr} are equal modulo r two by two with the projective coordinates of R_r.

As the ECADD-complete-unified operation conserves the equality of the point coordinates value modulo r, we conclude that the ECSM computation conserves the equality of the point coordinates value modulo r between the computation over the ring extension and over the finite field $\mathbb{F}_r$.

9.5.3. Discussion

About small curve requirements Both for Edwards and twisted Edwards curves, the small curve is of course not a cryptographic-grade curve. Indeed, the modulus r is too small and the curve might have points of low order. However, the small curve is not intended to be the support of a secure cryptographic operation: the computation on this curve actually remains internal to fault-detection-enabled ECSM. That is, the small curve is intended here to carry out exactly the same computation as that done in the curve on the extended ring, in order to enable the integrity verification.

Resistance to some attacks As a general guideline, additional protection against the *common point* attack [252] shall be enforced. This attack is based on curve parameters alteration, with the hope that the obtained curve is weak. Thus, to thwart this attack, the curve parameters shall be tested before and after the computation.

Table 9.1: Theory of the elliptic curve addition cost for th Edwards and twisted Edwards curves

Curves type	ECADD-complete-unified on $\mathbb{F}_p$	ECADD-complete-unified on $\mathbb{Z}_{pr}$	ECADD-complete-unified on $\mathbb{F}_r$	Total cost of the countermeasure	Computational overhead with: $n'=8$	Computational overhead with: $n'=16$
Edwards	$11.8n'^2+7n'$	$11.8n'^2+30.6n'+18.8$	19.8	$11.8n'^2+30.6n'+38.6$	$\approx+28\%$	$\approx+13\%$
Twisted Edwards	$11.8n'^2+7n'$	$12.8n'^2+32.6n'+29.8$	19.8	$12.8n'^2+32.6n'+49.6$	$\approx+39\%$	$\approx+21\%$

9.6. Performance

Our implementation uses the projective coordinates described in [249, Sec. 4, page 9]. Projective unified addition version takes $10\mathcal{M}+1\mathcal{S}+1\mathcal{C}+1\mathcal{D}+7\mathcal{A}$ where $\mathcal{M}$ is the cost of multiplication, $\mathcal{S}$ is the cost of square, $\mathcal{C}$ is the cost of multiplying by c, $\mathcal{D}$ is the cost of multiplying by d, and $\mathcal{A}$ abbreviates addition. The ECSM is the algorithm *add-always left-to-right* like described in Alg. 7. The bitwidth of the modulus is denoted by n (e.g., $n=256$ for Ed25519). We denote by n' the number of words of the modulus, that is $n'=256/32=8$ on 32-bit platforms (or $n'=256/16=16$ on 16-bit platforms). We consider that cost of a multiplication of two numbers composed by n' words is n'^2, cost of a square $\mathcal{S}$ is $0.8\mathcal{M}$ and the addition $\mathcal{A}$ is n'. The Tab. 9.1 permits to compare the time of each ECADD-complete-unified, depending of the number of words n'.

Algorithm 7: Add-always left-to-right scalar multiplication on elliptic curve $\mathcal{E}$.

Input : $P\in\mathcal{E}$, $k=\sum_{i=0}^{n-1}k_i2^i$ (n is the scalar size in bits, where $k_i\in\{0,1\}$)
Output : $[k]P$

1 $R_0\leftarrow P$
2 $R_1\leftarrow P$
3 $j\leftarrow n-2$
4 $b\leftarrow 0$
5 **while** $j\geq 0$ **do**
6 $\quad R_0\leftarrow R_0+R_b$ ▷ ECADD-complete-unified
7 $\quad b\leftarrow b\oplus k_j$
8 $\quad j\leftarrow j+k_j-1$
9 **end**
10 **return** R_0

9.6.1. Edwards curve example

For our experiment, we generate a Edwards curve on the finite field $\mathbb{F}_{2^{255}-19}$ defined by $x^2+y^2=1-6x^2y^2 \mod 2^{255}-19$.

Using the Prop. [253, sec 3.1], this Edwards curve corresponds to an elliptic curve defined by $\mathcal{E}: v^2=u^3+a_2u^2+a_4u$ on $\mathbb{F}_{2^{255}-19}$, with $a_2=-5$ and $a_4=49$. The

number of elements defined on the curve computed by MAGMA tool [254] is:

$$\#\mathcal{E}(2^{255}-19) = 2^{255} + 138694172605265013181071149003381840660.$$

We find a generator point (x_G, y_G) on the Edwards curve with:

$$x_G = 53746514586250388770967951861766021561817370662802863797712166095360241234126,$$
$$y_G = 19570081233560550597987439135529516381390903225319934175948181057081969418594.$$

The co-factor of the curve is 4. For the small curve, we can choose $r = 2147499037$; hence we have $c' = 1800340494, d' = 1430405543, x'_G = 28751952$ and $y'_G = 1290929995$. These parameters verify the lemma 9.5.1. The probability of fault non-detection is about equal to 2^{-31}.

Remark The probability namely "$c^2 + \lambda p$ is a non-zero square in $\mathbb{F}_r$" is about 1/2 and the probability namely "$\frac{dc^2}{c^2+\lambda p}$ is a non-zero no-square in $\mathbb{F}_r$" is about 1/2. To generate 500.000 random primes $r < 2^{32}$ verifying the lemma 9.5.1, using online version on MAGMA tool [254], the time is 110.769 seconds. The number of random prime number generated is 1.999.238. The probability that a random prime r meets the requirement of lemma 9.5.1 is less than 1/4 verified by this experimental part.

9.6.2. Twisted Edwards curve example: Curve25519 / Ed25519

On the finite field $\mathbb{F}_{2^{255}-19}$, the elliptic curve Curve25519 defined by the equation $v^2 = u^3 + 48662u^2 + u$ is birationally equivalent to the twisted Edwards Curves Ed25519 defined by equation $-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$.

This equivalence is given by:

$$\begin{cases} x = \frac{u}{v}\sqrt{-48664} \\ y = \frac{u-1}{u+1} \end{cases} \quad \text{or} \quad \begin{cases} u = \frac{1+y}{1-y} \\ v = \frac{1+y}{(1-y)x}\sqrt{-48664} \end{cases} . \tag{9.7}$$

The Curve25519 is a Montgomery curve, where very efficient computations can be carried out using only the X and Z coordinates. We find a generator point (x_G, y_G)

Table 9.2: Prime Factors $< p$ of λ for the generator point (x_G, y_G) given in example (curve Ed25519 defined in Sec. 9.6.2)

Prime factors r	2	3	17	47	78857	843229	159962189299
Length in bit of r	2	2	5	7	16	19	40
r verifies the lemma 9.5.2	False	False	False	False	**True**	**True**	False

on the twisted Edwards curve Ed25519 with:

$$x_G = 24727413235106541002554574571675588834622768167397638456726423682521233608206 3,$$
$$y_G = 15549675580280190176352668710449542251549572066445060580507079593062643049417.$$

The prime factors of λ (recall definition 21) smaller[3] than p are stored in the Tab. 9.2.

For the small curve like described in Tab. 9.2, we can choose:

1. $r = 78857, a' = 32865, d' = 47471, x'_G = 71670$ and $y'_G = 16752$, or
2. $r = 843229, a' = 839079, d' = 43998, x'_G = 96826$ and $y'_G = 488894$.

These parameters verify the lemma 9.5.2.

The probability of fault non-detection is about equal to 2^{-16} for the first case and to 2^{-19} for the second case.

Important remark we notice that the small verification field $\mathbb{F}_r$ cannot be chosen at random. Instead, the value of r is highly constrained, as shown in Tab. 9.2. This limitation of the ring extension countermeasure was not previously known.

9

9.6.3. Comments about results

One can see in Tab. 9.1 that the global time computation increases by 28% or 39% for each addition operation using a 256-bit curve with a 32-bit processor ($n' = 8$). The computation overhead decreases when the curve parameters and the security increase. Remarkably, the implementation code is the same for the two ECSM computations. The memory storage requirement is increased by two word registers for each variable.

[3] Actually, there is in λ only one factor larger than p, of length ≈ 900 bits, hence of no practical use—it is indeed more efficient to perform the computation several times or to verify the signature.

9.7. Conclusions

It is well known that detecting faults while computing elliptic curve cryptography can be achieved thanks to ring extension. In this paradigm, two entangled computations are carried out in the extended ring, allowing to tightly produce the functional result along with a redundant one, which can be checked independently. However, classical methods fail because the redundant computation evaluated standalone or entangled can be different, owing to some tests being independently evaluated when the elliptic curve formulae are not complete. Edwards curves and twisted Edwards curves have complete formulae, hence are not concerned with the issue of consistent tests requirement. Still, the application of ring extension involves some technicalities, we discuss in the paper. Namely, Edwards curves require an adaptation with Chinese reminder theorem of the curve constant parameter. As for twisted Edwards curves, the modulus extension can only be performed with a factor of λ, which is related both to the curve parameters and to the base point. The outcome is a provable fault detection method for (twisted) Edwards curves which, despite its simplicity, is novel, elegant and effective.

10

A novel physical EM Fault coutermeasure

Electromagnetic injection (EMI) is a powerful and precise technique for fault injection in modern ICs. This intentional fault can be utilized to steal secret information hidden inside of ICs. Unlike laser fault injection, tedious package decapsulation is not needed for EMI, which reduces an attacker's cost and thus causes a serious information security threat. In this paper, a PLL-based sensor circuit is proposed to detect EMI reactively on chip. A fully automatic design flow is devised to integrate the proposed sensor together with a cryptographic processor. A high fault detection coverage and a small hardware overhead are demonstrated experimentally on an FPGA platform.

10.1. Introduction

Embedded systems play an important role in a current advanced information society and will become fundamentally critical in a coming Internet of Things (IoT) era. Although IoT could benefit a human life significantly, the hardware security of the IoT embedded system becomes a serious technical issue. Since the IoT devices are distributed anywhere and everywhere in the human life and collect precious private information, the systems could be a potential target of physical attacks.

Side-channel and fault attacks are well-known physical attacks. Side-channel attack (SCA [255]) is passive in nature, which basically search for sensitive information by observing unintentional but natural side-channel leakage of physical

parameters, such as power consumption, EM radiation, and operation timing of ICs. Fault injection attack (FIA [2]) is on the other hand active in nature, which intentionally turns the target device in an abnormal operate condition and induces computation faults. This fault operation can be exploited to reveal sensitive information. The abnormal operate conditions can be realized by several methods. The most common and low-cost methods include under-powering, over-clocking, and extreme-heating/cooling. These techniques have a global impact and thus lack precision in location and type of faults. Efficient FIA needs more local fault injection in time and space domains. Sophisticated techniques such as laser injection or focused ion beam (FIB) injection are efficient but very expensive for the attackers and both of them require package decapsulation.

Electromagnetic injection (EMI) is one of the efficient and low-cost methods for fault injection in modern ICs [256]. There are two main advantages in EMI. Firstly, unlike laser injection, EMI can be performed without detailed decapsulation for opening of the target circuit since the EM field can penetrate the package and therefore no dedicated IC chip preparation is required. This reduces the attacker cost significantly by saving the time and money on careful decapsulation and accidental destruction of the chip. Moreover, secure chips often deploy several physical sensors which are triggered upon the opening of the package. Such countermeasures can be easily bypassed by EMI. The second advantage of EMI is high configurability. Depending on the various parameters like probe size, orientation, and injection pulse width/frequency, the fault injection timing and spot can be precisely adjusted.

There are a few research works done to develop countermeasures against this EMI. One of the best known methods is glitch detectors [257]. Zussa et al. proposed to integrate multiple FF-based distributed sensors and their clock networks to detect EMI-induced glitches before inducing the actual fault operation in the protected core circuit. When one of the sensors detects glitches, an alarm signal is raised. The biggest problem is its sensitivity control. Since the detector utilizes simple delay line based sensor, the EMI detection threshold is broadly fluctuated over PVT variations. In addition, there is no clear criteria on how many and how finely the sensors should be distributed to guarantee the security margin against EMI.

In this paper, a novel countermeasure to detect EMI is proposed. This countermeasure is based on PLL (phase locked loop). A PLL is a clock control circuit and is found easily in modern ASICs and FPGAs. Whenever, a PLL is started, it needs multiple cycles to generate a stable clock. This state where the PLL generates stable clock is known as a 'locked' state. When the parameters of the PLL is modified during operation, its lock is broken and it again needs multiple clock cycles to come back to the 'locked' state. Proposed countermeasure exploits this property of PLL. Precisely, the PLL monitors an internally generated clock. When an attacker

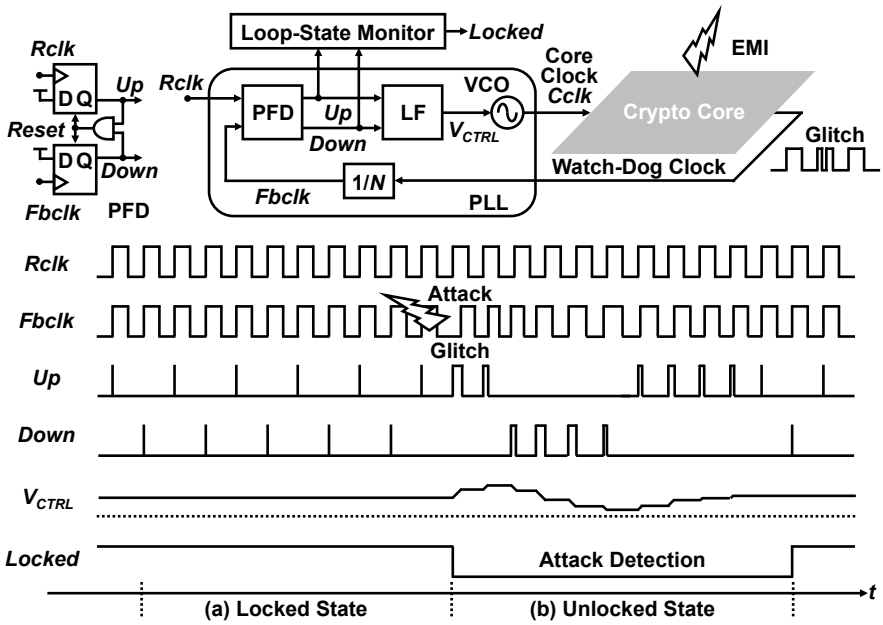


Figure 10.1: Block diagram of PLL-based EMI countermeasure and its operation waveforms in (a) lock and (b) unlock state.

attempts to perform EMI, the internal clock changes phase/frequency which leads to unlocking of PLL and thus detection of disturbances. Since the proposed system requires precise design controls in implementation that cannot be done by generic vendor tools, an automated implementation flow is devised in Xilinx FPGA environment for properly manipulating the bottom layer logic elements, merging the protection system and the security sensitive circuit to be protected.

The rest of the paper is organized as follows: Sec 10.2 describes the basic functioning of PLL and how it can be used to design EMI countermeasure. Sec 10.3 details the fully automated design flow for application of the proposed countermeasure in an FPGA platform. Experimental results are shown in Sec 13.6 and final conclusions are drawn in Sec. 10.5.

10.2. PLL-Based EMI Countermeasure

10.2.1. Concept

Phase Locked Loop or PLL, is a clock timing control circuitry which detects phase and frequency difference between two clock sources and feedback-controls to keep synchronization between them. Fundamental building blocks of PLL are a Phase-Frequency Detector (PFD), a Loop Filter (LF), and a Voltage-Controlled Oscillator (VCO) as depicted in Fig. 10.1. PFD compares two clock input timings: one is an

external reference clock *Rclk* and feedback internal core clock *Fbclk*. Based on the timing comparison, PFD produces up and down pulses *(Up, Down)* whose duration time represents phase and frequency difference between two clock sources. A simple implementation of PFD consists of only one AND gate and two FFs with reset input. LF generates the control voltage V_{CTRL} for VCO depending on the up/down pulses given from PFD to adjust phase and frequency of VCO for the synchronization. Insertion of a frequency divider $(1/N)$ in the feedback loop, enables to generate N-times higher-frequency core clock *Cclk* synchronized with respect to the external reference clock *Rclk*. PLL is therefore widely used for global clock synchronization in the LSI-based electronic systems and thus almost of all the modern high-performance SoC ASICs and FPGAs integrate PLL on-chip.

This existing PLL can be utilized for an EMI countermeasure. Fig. 10.1 describes the basic concept. PLL can be seen as a continuous monitor of internal clock stability. Since the purpose of EMI is an intentional induction of instantaneous abnormal circuit operations by large-power EM pulse, PLL can detect this EMI attack by utilizing feedback clock path as a watch dog signal. In a stable steady state of the PLL loop, both the phase and frequency are locked between the external reference clock and the internal clock sources. Thus PFD only produces almost-no or very short up/down pulses in a locked state (Fig. 10.1 (a)). Once PLL looses lock due to instantaneous glitches or jitter in the feedback clock path, PLL starts to recover lock by producing explicit up/down pulses for many clock cycles (Fig. 10.1 (b)). This locked/unlocked state can be easily distinguished by a digital Loop-State Monitor (LSM). By distributing the feedback clock path all over the security sensitive crypto core to be protected, the EMI attack can be easily captured as an unlock event in the PLL loop. In other word, PLL amplifies the instantaneous erroneous operation in the watch-dog clock as a digital attack-warning signal. Once the attack is captured on-chip, the crypto core can be protected by disabling the core or operating it in dummy operation mode, such as in [258].

For further enhancing the security level, the reference clock path is together included in the watch-dog clock paths (Fig. 10.2). This modified configuration makes it possible to detect the EMI attack before the actual fault operation is induced in the crypto core. In addition, malicious attacks, such as EMI injection into the control loop itself, can be detected since these attacks anyway cause the unlocked state of PLL. A repeater configuration (Fig. 10.2 (a)) can provide flexibility for the synchronization with the external clock source *Rclk*. This configuration exposes the clock port to the attackers however the fault attack exploiting this clock port is very difficult. Over-clocking or glitch insertion of *Rclk* can be detected by PLL in advance of the actual fault operation. Also, by carefully designing the maximum operating frequency of the crypto core to be higher than the PLL tuning range, a Fault Sensitivity Analysis (FSA [215]) is also disabled (the attacker only see the

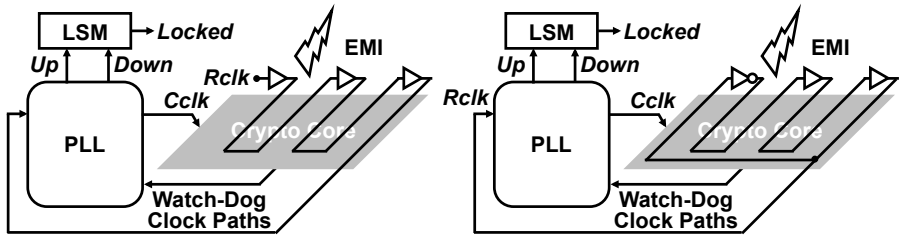


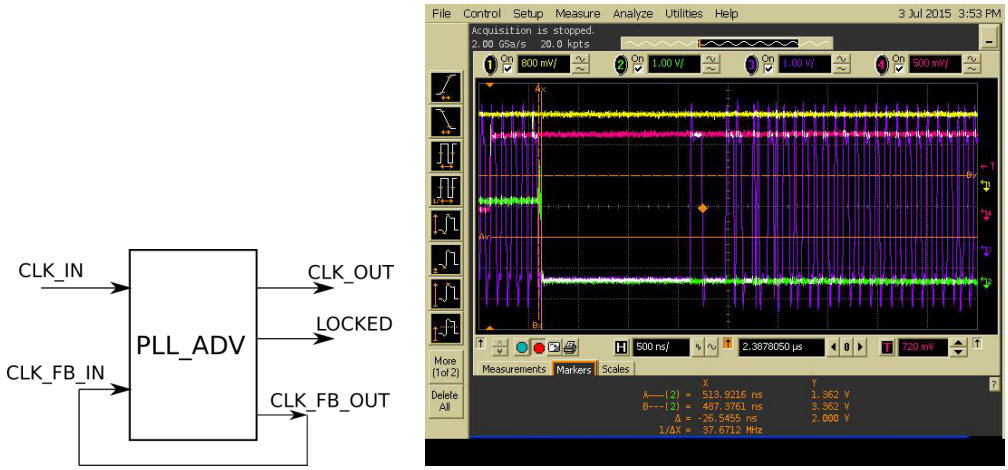
Figure 10.2: Security-enhanced implementation of EMI countermeasure with reference clock watch dog in (a) repeater and (b) ring-oscillator configuration.

fault operation of PLL). A ring-oscillator (RO) configuration (Fig. 10.2 (b)) hides the clock port on-chip for maximally enhancing the security. No exposed port is available to the attackers. In this configuration, the crypto core asynchronously communicates with other circuits in a hand-shaking manner which is common for the security core design. A test chip in this work integrates the asynchronous crypto core with the EMI countermeasure in RO configuration for the highest security demonstration. The technical challenges lie in (1) how to efficiently implement the crypto core with the countermeasure in a fully automatic design flow; (2) how to adjust the sensitivity of the attack detection to guarantee a high enough security margin i.e., countermeasure raises alarm much before the crypto core is in fault.

10.2.2. Implementation Details

The proposed countermeasure uses an active ring oscillator (RO) as a watch-dog circuit (Fig 10.2(b)). The frequency f of a RO can be defined as $\frac{1}{2*t*n}$, where t is gate delay and n is number of inverters. n is chosen as 1 to allow maximal oscillating frequency. The delay t in detail has two significant components: one is gate delay t_g and the other is routing delay t_r. As shown in Fig. 2(b), RO in our implementation is designed in a way to thoroughly envelop the sensitive module by making multiple loops to guarantee the full coverage of EMI detection. The RO delay is mostly composed of routing delay including buffers delay and therefore t_r cannot be neglected. The operating frequency f can be finally written as $f = \frac{1}{2*(t_g+t_r)}$. f tends to be as low as several MHz because long single routing is needed for the detection coverage. However, the operating frequency of the crypto core can be still increased by adjusting the clock division rate of the PLL feedback path. The performance degradation of the crypto processing can be minimized.

When an EM injection is made on the chip, the EM glitch will try to disturb the sensitive core. Since the RO is routed over the sensitive core, the glitch will change the routing time delay to a value t_r'. The modified frequency of RO will be $f' = \frac{1}{2*(t_g+t_r')}$. This sudden change in frequency will impact the phase of the RO, which



(a) (b)

Figure 10.3: Validation of the countermeasure principle on Spartan-6. (a) Configuration of PLL, (b) Observed signal on scilloscope with trigger in pink, EMI pulse in yellow, CLK_OUT in purple and LOCKED in green.

in turn will force the PLL to unlock state. Thus, the *LOCKED* signal of LSM raises an alarm.

To demonstrate this phenomena, simple experiments are performed on Spartan-6 FPGA. PLL_ADV block on Spartan-6 FPGA is used as a PLL module, which receives external clock as input. The CLK_FB_OUT is fedback to CLK_FB_IN for self-calibration, as shown in Fig. 10.3(a). The *LOCKED* output of the PLL is observed on an external pin. The results are shown in Fig. 10.3(b). When the injected pulse disturbs the internal clock routing, the PLL is unlocked and *LOCKED* signal (in green) moves to zero. EMI pulse (in yellow) appears multiple cycles after the trigger (in pink) i.e., triggering delay. As soon as the EMI pulse appears, the *LOCKED* signal goes down asynchronously and raises an alarm. The EMI pulse disturbs the phase of the CLK_FB_IN, which leads to breaking of the lock. A disturbance on the (asynchronous) reset pin of the PLL can also have similar affects. When the EM pulse is removed, the PLL starts to stabilize itself and the *LOCKED* signal stays low during this period for multiple cycles.

A similar behavior is observed by changing the phase of externally-input reference clock CLK_IN. However a gradual and slow change in frequency of CLK_IN does not triggers the alarm. The attacker can possibly exploit the external clock to perform a FSA [215] in the crypto core. By carefully designing the maximum operating frequency of the crypto core to be higher than the tuning range of PLL, FSA can be disabled. However this is not always possible for any crypto core and existing PLL pair. The clock generated by the RO is therefore utilized as the reference clock input to derive the clock source for the crypto core. This configuration

enhances the PVT variation tolerance of the countermeasure. Since a change in process, supply voltage, and temperature will impact both the crypto core and RO in a similar way, susceptibility (sensitivity) against EMI is also changed similarly. In addition, the precise fault control by intentional disturbance of supply voltage and temperature becomes difficult.

An EM attack sensor [258] was presented as a countermeasure against a passive side-channel attack. This sensor was build on a LC coil acting as a probe sensor by detecting frequency shift with respect to another calibrating coil. It can detect micro EM probes which approach the circuit at a proximity within 0.1 mm. In our approach, the EM-probe detection range is as long as few cms as it detects active EM probes. These two techniques can be together integrated to cover the protection against a passive proximity EM analysis attack and an active EMI.

10.3. Design Automation

An automatic design flow is devised to implement the proposed countermeasure together with the cryptographic processor to be protected. The detailed explanation is separated into two parts. First, the procedure to automatically place and route the watch-dog RO in an FPGA is explained. Second, the detailed design flow to integrate RO and PLL together with the crypto core.

10.3.1. Controllable RO Routing Flow

Due to limited routing freedom in FPGAs, routing manipulation is almost impossible in an automated manner with standard vendor tools. To implement the RO in the proposed countermeasure, the routing from inverter output to its own input should be sufficiently long to thoroughly cover all the sensitive block. The EM probe trying to inject fault will therefore interfere with RO. However, using the standard commercial router only results in very uncontrollable and short loops, as the tools are designed for resource optimization by default. In order to properly control the RO routing path, a customized tool is constructed over Rapidsmith library [259] to manipulate the logic and routing in a Xilinx environment. The proposed design flow mainly comprises of 4 distinct stages.

1. 4 single-inverter RO modules are instantiated by a HDL source code, and deployed on the 4 corners of a rectangular region to be covered. 3 of the 4 ROs are actually dummy for positioning the region boundary of the watch-dog clock path. The preliminary design is implemented by generic FPGA design flow to create the post P&R ncd file, as shown in Fig 10.4(a).

2. The ncd file is parsed by the customised analysis tools to be converted into xdl format, which is human readable and contains all the implementation details mapping to a specific device. The basic routing element in Xilinx environment is "node" that is used to concatenate different instances and populate a complete routing network. The nodes of the 4 ROs are extracted from each RO modules in the xdl file [260]. For the main RO, a source node i.e., output pin of inverter (labeled node 1.1) and the sink node i.e., input node of inverter (labeled node 1.2) are extracted as shown in Fig 10.4(b).

3. In this step, the dummy ROs are simply removed from the xdl, only their nodes are left in the design. On the contrary, the LUT of the main RO is intact, but all the routings are removed, except the node 1.1 and node 1.2.

4. Since the vendor router is just capable of routing a path from source to sink, a custom **node router** is developed that can partially route a path between the two designated nodes. Following the previous steps, 4 independent routings are performed using the node router: node 1.1 → node 2 → node 3 → node 4 → node 1.2. Each individual route follows the generic routing protocol which results in the shortest path between two nodes. Next, a long routing path is completed that is positioned to be a rectangular by the selected corner nodes (1.1, 1.2, 2, 3, 4) as seen in Fig 10.4(d).

The same flow can be simply extended when the RO has a zig-zag shape (Fig. 10.3) and more than 4 nodes can be identified.

10.3.2. Co-Integration Flow of Sensor and Crypto Core

This section presents an automatic design flow to implement the countermeasure with a cryptographic processor core. The module to be protected can be any other sensitive circuit, such as CPU, key storage, etc. In this work, a cryptographic core of Simon block cipher is considered as an design example. All the steps are automated and concatenated using Java, which calls Xilinx tools in command line mode. The xdl files are also processed using customised tools [260]. The main procedure is sketched in Fig. 10.5 and involves the following steps:

1. The flow starts with the HDL description of the cipher i.e., cipher.v and the specific placement constraints on logic region. The cipher is synthesized, placed and routed to extract the ncd, followed by xdl file of the cipher implementation.

2. The placement constraints from the previous steps are also used to de-

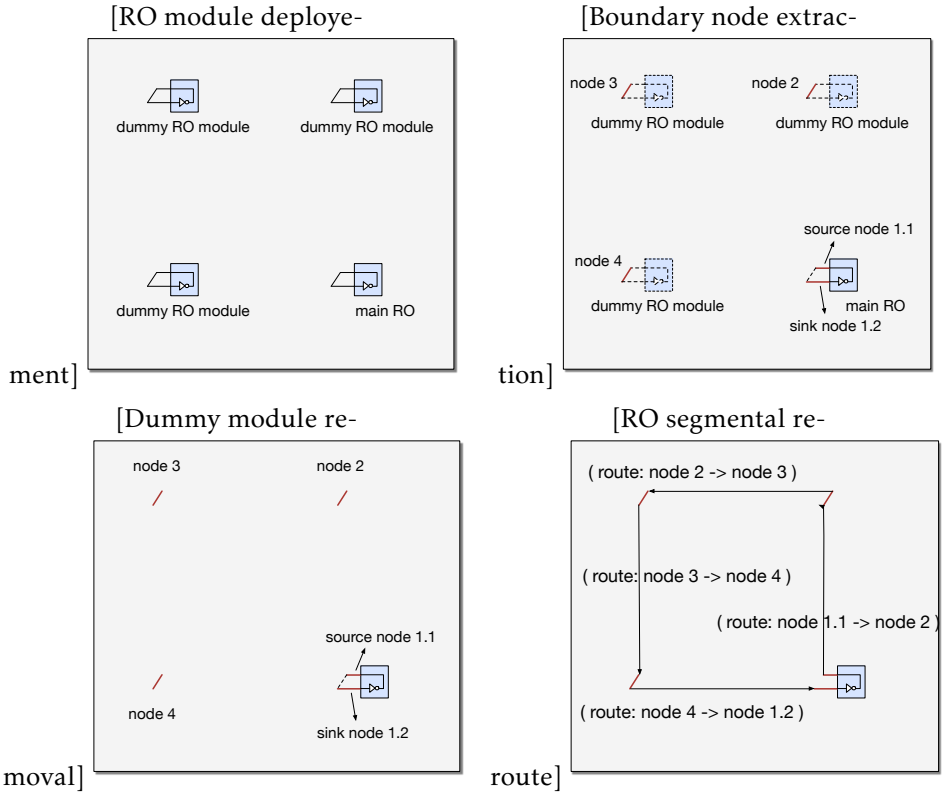


Figure 10.4: Exemplary design flow to implement a RO with restrained routing requirements.

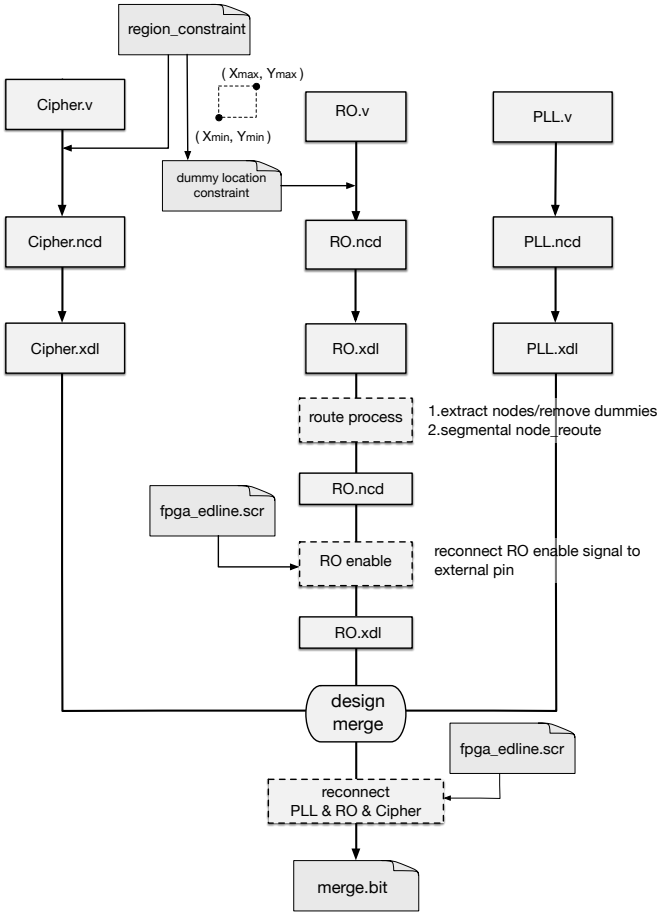


Figure 10.5: Design flow of automatic co-integration of crypto core and countermeasure.

fine the corner nodes of the 4 ROs implemented. 3 out of 4 ROs are dummy modules in nature and are eventually removed. Nevertheless, the corner nodes can also be extracted from `cipher.xdl` rather than defining in the constraint file.

3. The EM countermeasure previously explained is also implemented and converted to `xdl` file.

4. The `RO.xdl` is then modified to envelop the cipher module. This procedure is explained in previous subsection (see Fig. 10.4). An enable signal is also added to the final RO in an automatic way using `fpga_edline` commands at this step.

5. In the penultimate step, the three `xdl` designs are combined (cipher, PLL, modified_RO) and concatenated.

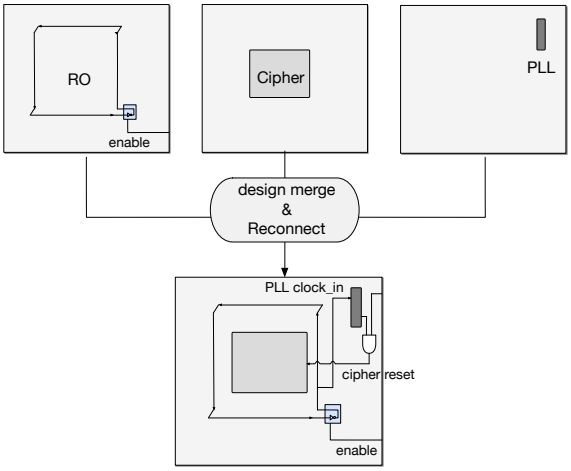


Figure 10.6: Conceptual overview of design flow.

6. The final step is the bitstream generation from the merged design.

A high-level overview of the flow is illustrated in Fig 10.6. The automation methodology can also be extended to ASIC design flow, of course with different tool tricks.

10.4. Experimental Evaluation

10.4.1. Experimental Setup

The proposed countermeasure is validated on SASEBO-W board, which carries a Spartan-6 FPGA from Xilinx. Spartan 6 FPGA is packaged in frontside (FG) package which makes it easier to perform EM injection as compared to other FPGA that are packaged in flip chip (FF). The chip package stays intact as EMI does not need package removal.

The full setup for EMI test is presented in Fig. 10.7. The signal generator is used to fine tune the frequency with a precision of 1 ps. A 300W (55dB) broadband, 400MHz class A amplifier allows injection of very short EM pulses of width of 1.5-ns. A remotely controllable XYZ axes table (not shown) is used to perfom a full fault cartography with a spacial resolution of 1.0 μm. The XYZ table scans the entire chip and generates a cartography as a 30×30 array.

10.4.2. Target Circuit

The countermeasure is validated on a hardened cryptographic module i.e., a block cipher which has fault detection capabilities in-built. Using such hardened cryptographic primitives simplifies the analysis part and it is easier to distinguish ex-

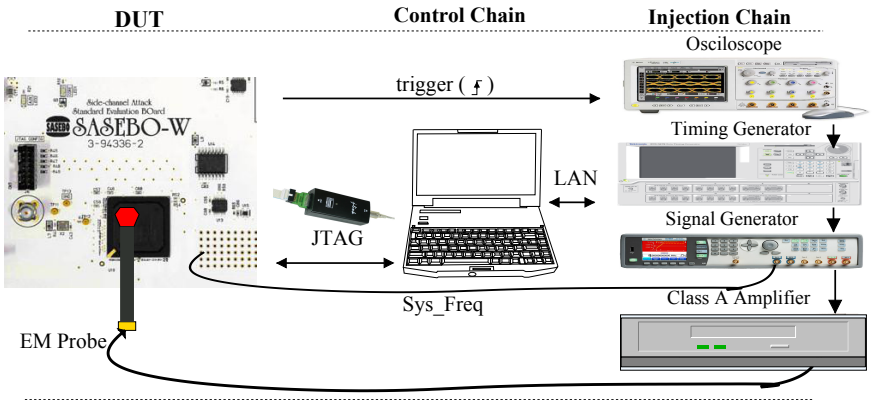


Figure 10.7: Picture of experimental setup for fault injection.

ploitable faults from faults which occur on the peripherals. The target circuit is a one cycle per round encryption only implementation of Simon32/64 block cipher implemented in Verilog. It is hardened for fault detection using encoded circuits [261] with a linear code of dual distance of 5. Encoded circuits operate by encoding the internal state of the circuit, using a linear code, which is then mixed with a random number. If a fault is injected inside the encoded circuit, it can be simply detected by decoding the random number and comparing with the initial random number. Any fault in the computation of cryptography will modify the code and hence the decoded random number will differ. More details on this technique can be found in [261]. Encoded circuits based hardening is applicable to any digital circuit. The validation is done on Simon32/64 core, as it stays small in area even after hardening and it is easier to analyze the faults.

Next, the proposed countermeasure is implemented on hardened Simon32/64 core using the design flow detailed in Sec. 10.3. The RO is routed using 12 nodes and the design flow converges in single iteration, without any conflicts. The overall cost of hardened Simon32/64 is 578 *flip-flops* and 1658 *LUTs* for Simon32/64 and the UART communication interface. Maximal operating frequency of the design is 50 MHz. Proposed countermeasure uses one *PLL_ADV* primitive for the PLL functionality and single *LUT* as an inverter along with routing resources to realize the RO. The circuit is driven close to maximal frequency so that faults are easier to inject. Moreover, the frequency of oscillation of the RO can be estimated from FPGA editor and PLL can be tuned accordingly. The hardened Simon core along with proposed countermeasure is validated on SASEBO-W platform. The analysis is conducted using the EMI platform explained in Sec.10.4.1. In the next section, the results of EMI injection are presented and analyzed.

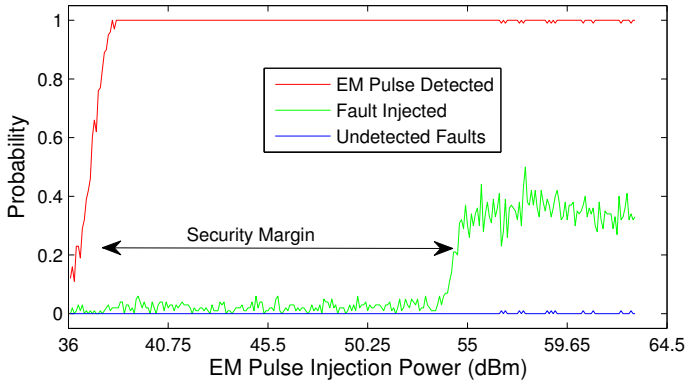
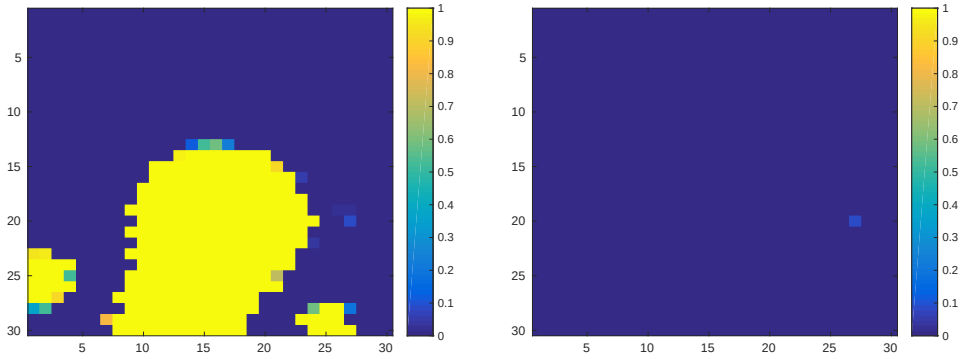


Figure 10.8: Sensitivity of the proposed countermeasure against EM pulse injection.

10.4.3. Experimental Results

The experiments are devised to test two parameters i.e., the sensitivity of countermeasure and the spatial coverage of the countermeasure. Sensitivity means that the countermeasure is expected to raise an alarm much before the EM pulse faults the sensitive circuit. This is done by fixing the position of the probe on a point on top of the sensitive core. The impact of injecting EM pulses with varying power intensity is shown in Fig. 10.8. The proposed countermeasure raises an alarm for EM pulses of power as low as 38dBm. On the other hand, faults in sensitive core (Simon) start appearing at a minimum power of 54dBm. The security margin of the proposed countermeasure can be estimated to 19dBm from Fig. 10.8. At 64dBm, the faults discontinue, i.e., the EMI setup has reached its practical limit. Moreover, repetition of this experiment at several other points over SIMON32/64 depicts similar trend. None of the performed experiment shows reverse trend i.e., faults in sensitive core are injected without raising the alarm.



(a) (b)

Figure 10.9: EM cartography of the (a) detection rate, (b) undetected faults on Spartan-6 FPGA implementing Encoded Simon32/64 and proposed countermeasure under EMI setup.

Next experiments studies the spatial coverage of the fault injection. The FPGA is scanned by a remotely controlled XYZ table carrying the EM injection probe and the FPGA is scanned using 30×30 grid. The fault detection capability of the EMI countermeasure is shown using a color map in Fig 10.9(a). The current results are at injection power of 53dBm i.e., in the region of fault injection on Simon. The expected region over Simon core and related I/0 (i.e., center of FPGA) reports 100% fault detection. The fault detection probability falls to less than 100% in certain regions. However, the core is totally encircled by the watch-dog RO including some margin, thus such injection do not impact the Simon core.

There were a couple of pixels on the cartography, where some unexpected behavior occurred, as shown in Fig. 10.9(b). These were around pixel (27,18). Although no part of Simon core or the countermeasure is placed around these pixels but still some faulty output can be observed. Moreover, this faulty output turned out to be non-exploitable. From these observations, it can only be speculated that the EM pulse violates an internal (hidden) sensor which disturbs the FPGA functioning. It could also be due to disturbance of some power delivery or clock network or PLL itself. However, as the internal architecture of FPGA is not known, this hyposthesis cannot be confirmed. Reffering back to Fig. 10.8, at higher power some undetected faults are injected with a probability lower than of 0.01. When the the power of EM pulse is high, an injection triggers similar sensors and unexpected behavior is observed.

All the experiments were performed without any chip decapsulation. So it is hard to point exact location of crypto-core and sensor routing on the cartography. The FPGA die only covers a part of the package. Future work can explore techniques like X-ray imaging for deeper analysis.

10.4.4. Discussion

This section compares countermeasure proposed in this work with the state of the art. In [257], authors use multiple glitch detectors to detect local EM injection. Their idea is simple and works very well. Now comparing with proposed scheme, glitch detectors can have several shortcomings. First of all, there is no defined methodology to find the required number and placement of glitch detectors, which makes automation difficult. Proposed scheme needs only one RO and an existing PLL, and can be easily automated. The need of PLL on-chip is one limitation of proposed scheme however PLL exists almost of all the modern FPGAs and ASICs. The placement of proposed scheme is easy as it needs to only envelope all the sensitive core thoroughly with wide enough security margin. Every glitch detector works as stand-alone environment sensor, and thus every detector has different sensitivity to local PVT and process variation. The sensitivity of proposed counter-

Property	Glitch Detector [257]	This Work
Design Flow	Semi-Automatic	Fully-Automatic
Placement	Difficult	Envelope target
PLL Necessity	No Need	PLL (Existing)
Composition	n Detectors	1 PLL + 1 RO
PVT Variation Immunity	Low	High

Table 10.1: Qualitative Comparison of Proposed Countermeasure with State of The Art

measure is majority routing spread over a large area, which make PVT and process variation sensitivity limited. These properties are summarized in Table **??**.

10.5. Conclusions

EMI has recently emerged as a precise and powerful fault injection technique. In this paper, a circuit level countermeasure against EMI is proposed. It uses a PLL circuit and a free running oscillator. A fully automated design flow is provided for supporting the implementation of the proposed countermeasure system. The protected scheme is validated on Spartan-6 FPGA. Results show that the proposal detects all the faults targeting the sensitive core with significant security margin of 19dBm. Further work will focus on better understanding of unexpected fault behavior from hidden architecture of FPGAs, using techniques like X-ray imaging.

11

Reconfigurable LUT: A Double Edged Sword for Security-Critical Applications

Modern FPGAs offer various new features for enhanced reconfigurability and better performance. One of such feature is a dynamically Reconfigurable LUT (RLUT) whose content can be updated internally, even during run-time. There are many scenarios like pattern matching where this feature has been shown to enhance performance of the system. In this paper, we study RLUT in the context of secure applications. We describe the basic functionality of RLUT and discuss its potential applications for security. Next, we design several case-studies to exploit RLUT feature in security critical scenarios. The exploitation are studied from a perspective of a designer (ex. designing countermeasures) as well as a hacker (inserting hardware Trojans).

11.1. Introduction

Field Programmable Gate Arrays (FPGAs) have had a significant impact on the semiconductor market in recent years. FPGAs came into the VLSI industry as successor of programmable read only memories (PROMs) and programmable logic devices (PLDs) and has been highly successful due to its reconfigurable nature.

A standard FPGA can be defined as islands of configurable logic blocks (CLBs) in the sea of programmable interconnects. However, with time, FPGAs have become more sophisticated due to the addition of several on-chip features like high-density block memories, DSP cores, PLLs, etc. These features coupled with their core advantage of reconfigurability and low-time to market have made FPGA an integral part of the semiconductor industry, as an attractive economic solution for low to medium scale markets like defense, space, automotive, medical, etc. The key parameters for FPGA manufacturers still remain area, performance and power. However, during these recent years, FPGA manufacturers have started considering security as the fourth parameter. Most recent FPGAs support bitstream protection by authentication and encryption schemes [262]. Other security features like tamper resistance, blocking bitstream read-back, temperature/voltage sensing, etc. are also available. FPGA has also been a popular design platform for implementations of cryptographic algorithms due to its reconfigurability and in house security. Apart from the built-in security features, designers can use FPGA primitives and constraints to implement their own designs in a secure manner. In [263], authors show several side-channel countermeasures which could be realized on FPGAs to protect one design. Another work [264] demonstrates the efficient use of block RAMs to implement complex countermeasures like masking and dual-rail logic. DSPs in FPGAs have also been widely used to design public-key cryptographic algorithms like ECC [265, 266] and other post-quantum algorithms [267]. Moreover, papers like [268] have used FPGA constraints like *KEEP*, *Lock_PINS* or language like *XDL* to design efficient physical countermeasures.

The basic building block of an FPGA is logic slices. Typically a logic slice contains look up tables (LUTs) and flip-flops. LUTs are used to implement combinational logics whereas flip-flops are used to design sequential architectures. Every LUT contains an *INIT* value which is basically the truth table of the combinational function implemented on that LUT. This *INIT* value is set during the programming of the FPGA through bitstream. Generally this *INIT* value is considered to be constant until the FPGA is reprogrammed again. However, in recent years, a new feature has been added to the FPGAs which allows the user to modify the *INIT* value of some special LUTs in the run time, without any FPGA programming. These special LUTs are known as reconfigurable LUTs or RLUTs as they can be reconfigured during the operation phase to change the input-output mapping of the LUT. To the best of our knowledge, RLUTs have found relevant use in pattern matching and filter applications [269]. Side channel protection methodology using RLUT is presented in [270] where the authors have combined different side channel protection strategies with RLUTs and have developed leakage resilient designs. However, in that work the authors have concentrated mainly on constructive use of RLUTs, not on destructive applications which is covered by our paper.

In this paper, we aim to study the impacts and ramifications of these RLUTs on cryptographic implementations. We have provided a detailed study of RLUTs and have deployed it in many security related applications. We propose several industry-relevant applications of RLUT both of constructive and destructive nature. For example, an RLUT can be easily (ab)used by an FPGA IP designer to insert a hardware Trojan. On the other hand, using RLUT, a designer can provide several enhanced features like programming secret data on client-side. The contribution of the paper can be listed as follows:

- This paper provides a detailed analysis of RLUTs and how it can be exploited to create extremely stealthy and deadly hardware security threats like hardware Trojans (destructive applications).
- Moreover, we also propose design methodologies which uses RLUTs to redesign efficient and lightweight existing side channel countermeasures to mitigate power based side channel attacks (constructive applications)
- Thus, in this paper we show that how RLUTs provide a gateway of creating efficient designs for both adversary and normal users and act as double-edged swords for security applications. To the best of our knowledge, this is the first study which provides a detailed security analysis of RLUTs from both constructive and destructive points of view.

The rest of the paper is organized as follows: Sec. 11.2 describes the rationale of an RLUT and discusses its advantages and disadvantages. Thereafter several destructive and constructive applications of RLUT are demonstrated in Sec. 11.3 and Sec. 11.4 respectively. Finally conclusions are drawn in Sec. 11.5.

11.2. Rationale of the RLUT

RLUT is a feature which is essentially known to be found in *Xilinx* FPGAs. A *Xilinx* RLUT can be inferred into a design by using a primitive cell called *CFGLUT5* from its library. This primitive allows to implement a 5-input LUT with a single output whose configuration can be changed. *CFGLUT5* was first introduced in Virtex-5 and Spartan-6 families of *Xilinx* FPGAs. As we will show later in this section, the working principle of *CFGLUT* is similar to the shift register or the more popularly known *SRL* primitives. Moreover, some older families of Xilinx which do not support *CFGLUT5* as a primitive, can still implement RLUT using the *SRL16* primitive. In the following, for sake of demonstration, we stick to the *CFGLUT5* primitives. Nevertheless the results should directly apply to its alternatives as well.

As stated earlier, a RLUT can be implemented in Virtex-5 FPGAs using a *CFG-*

LUT5 primitive. The basic block diagram of CFGLUT5 is shown in Fig. 11.1. It is a 5-input and a 1-output LUT. Alternatively, a CFGLUT5 can also be modeled as a 4-input and 2-output function. The main feature of CFGLUT5 is that it can be configured dynamically during the run-time. Every LUT is loaded with a $INIT$ value, which actually represents the truth table of the function implemented on that LUT. A CFGLUT5 allows the user to change the $INIT$ value at the run-time, thus giving the user power of dynamic reconfiguration internally. This reconfiguration is performed using the CD_I port. A 1-bit reconfiguration data input is shifted serially into $INIT$ in each clock cycle if the reconfiguration enable signal (CE) is set high. The previous value of $INIT$ is flushed out serially through the CD_O port, 1-bit per clock cycle. Several CFGLUT5 can be cascaded together using reconfiguration data cascaded output port (CD_O).

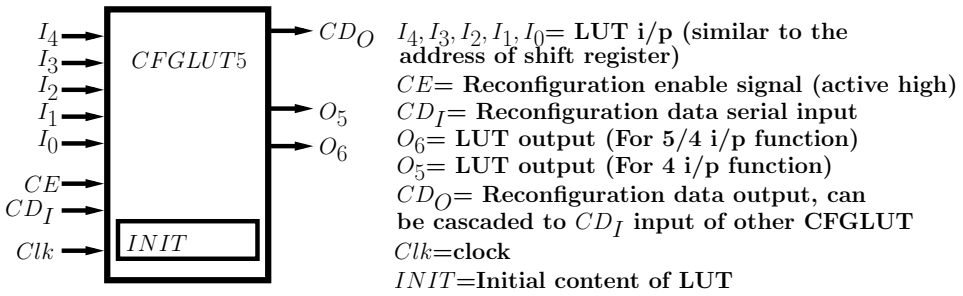


Figure 11.1: Block diagram of CFGLUT5

The reconfiguration property of CFGLUT5 is illustrated in Fig. 11.2 with the help of a small example. In this figure, we show how the value of $INIT$ gets modified:

- from value $O = (O_0, O_1, O_2, \ldots, O_{30}, O_{31})$,
- to a new value $N = (N_0, N_1, N_2, \ldots, N_{30}, N_{31})$.

This reconfiguration requires 32 clock cycles. As it is evident from the figure, reconfiguration steps are basic shift register operations. Hence if required, reconfiguration of LUT content can be executed by using shift register primitives ($SRL16E\_1$) in earlier device families. The CD_O pin can also be fed back to the CD_I pin of the same *CFGLUT5*. In this case, the original $INIT$ value can be restored after a maximum of 32 clock cycles without any overhead logic. We will exploit this property of RLUT later to design hardware Trojans.

There are two different kinds of slices in a Xilinx FPGA i.e., *SLICE_M* and *SLICE_L*. Whereas a simple LUT can be synthesized in either of the slices, CFGLUT5 can be implemented only in *SLICE_M*. *SLICE_M* contains LUTs which can be configured as memory elements like shift register, distributed memory along with combinational logic function implementation. CFGLUT5, when instantiated, is essentially mapped into a *SLICE_M*, configured as shift register ($SRL32$) as shown in Fig 11.3.

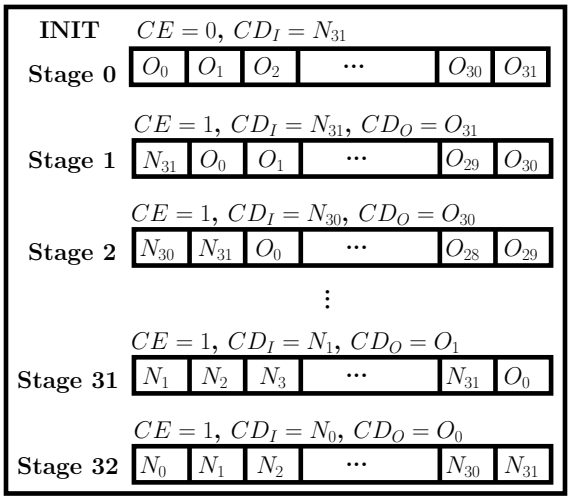


Figure 11.2: *INIT* value reconfiguration in CFGLUT5

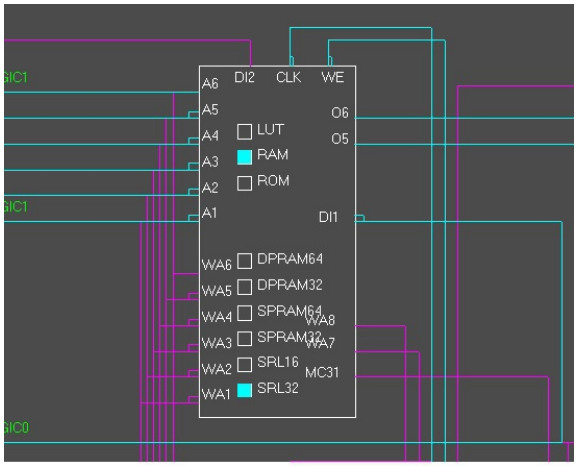


Figure 11.3: CFGLUT5 mapped in LUT as SRL32 as shown from Xilinx FPGA Editor

11.2.1. Comparison With Dynamic Configuration

Another alternative to reconfigure FPGA in run-time is to use partial or dynamic reconfiguration. This reconfiguration can also be exploited to implement secure architectures [271]. In partial reconfiguration, a portion of the implemented design is changed without disrupting operations of the other portion of the FPGA. This operation deploys an Internal Configuration Access Ports (ICAP) and the design needing reconfiguration must be mapped into a special *reconfigurable region*. Reconfiguration latency is in order of milliseconds. Partial reconfiguration is helpful when significant modification of the design is required. However, for small modification, using RLUT is advantageous as it has very small latency (maximum 32

clock cycles) compared to partial reconfiguration. RLUT is configured internally and no external access to either JTAG or Ethernet ports are required for reconfiguring RLUTs. Additionally, traditional DPR requires to convey an extra bit file which is not required in case of RLUT, making RLUT ideal for small reconfiguration of the design, in particular for Trojans.

11.2.2. RLUT and Security

Since we have described the functioning of RLUT in detail, we can clearly recognize some properties which could be helpful or critical for security. A typical problem of cryptographic implementations is its vulnerability to statistical attacks like Correlation Power Analysis (CPA) [111]. For instance, CPA tries to extract secret information from static cryptographic implementations by correlating side-channel leakages to estimated leakage models. A desirable feature to protect such implementations is reconfiguration of few internal features. A RLUT would be a great solution in this case as it has the power to provide reconfigurability at minimal overhead and with no external access. It is important to reconfigure internally to avoid the risk of any eavesdropping. On the other hand, RLUT can also be used as a security pitfall. For example, an efficient designer can simply replace a LUT with RLUT in a design keeping the same *INIT* value. Until reconfiguration, RLUT would compute normally. However upon reconfiguration, the RLUT can be turned into a potential Trojan. In the following sections, we would show some relevant applications of constructive or deadly nature. Of course it is only a non-exhaustive list of RLUT applications into security.

11.3. Destructive Applications of RLUT

In earlier sections, we have presented the basic concepts of RLUTs with major emphasis on *CFGLUT5* of Xilinx FPGAs. Though *CFGLUT5* provides user unique opportunity of reconfiguring and modifying the design in run-time, it also gives an adversary an excellent option to design efficient and stealthy hardware Trojan. In this section, we focus on designing tiny but effective hardware Trojan exploiting reconfigurability of RLUTs.

A hardware Trojan is a malevolent modification of a design, intended for either disrupting the algorithm operation or leaking secret information from it. The design of hardware Trojan involves efficient design of Trojan circuitry (known as payload) and design of trigger circuitry to activate the Trojan operation. A stealthy hardware Trojan should have negligible overhead, ideally zero, compared to the original *golden* circuit. Moreover, probability of Trojan getting triggered during the functional testing should be very low, preventing accidental discovery of the

Trojan. The threat of hardware Trojans is very realistic due to the fabless model followed by the modern semiconductor companies. In this model, the design is sent to remote fabrication laboratories for chip fabrication. It is very easy for an adversary to make some small modification in the design without violating the functionality of the design. The affected chip will give desired output in normal condition, but will leak sensitive information upon being triggered. More detailed analysis of hardware Trojans can be found in [272–274].

Researchers have shown that it is possible to design efficient hardware Trojans on FPGAs also. In [275] the authors have designed a Trojan on a Basys FPGA board which get triggered depending upon the 'content and timing' of the signals. On the other hand, authors in [276] have designed a hardware Trojan which can be deployed on the FPGA via dynamic partial reconfiguration to induce faults in an AES circuitry for differential fault analysis.

In this section, we will focus on effective design of hardware Trojan payload using RLUT. But before going into the design methodologies of payload using RLUTs, we will first describe the other two important aspects of the proposed hardware Trojans: Adversary model and Trigger methodologies.

11.3.1. Adversary Model

It is a common trend in the semiconductor industry to acquire proven IPs to reduce time to market and stay competitive. We consider an adversary model where a user buys specific proven IPs from a third party IP vendor. By proven IPs, we mean IPs with well-established performance and area figures. Let us consider that the IP under consideration is a cryptographic algorithm and the target device is an FPGA. An untrusted vendor can easily insert a Trojan in the IP which can act as a back-door to access sensitive information of other components of the user circuit. For instance, an IP vendor can provide a user with an obfuscated or even encrypted netlist (encrypted *EDIF*). Such techniques are popular and often used to protect the rights of the IP vendor. A Trojan in an IP is very serious for two major reasons. First, the Trojan will affect all the samples of the final product and secondly it is almost impossible to get a golden model. Moreover, research in Trojan detection under the given attack model is quite limited. The user does not have a golden circuit to compare, thus making hardware Trojan detection using side channel methodology highly unlikely. Additionally, this adversary model also makes the Trojan design challenging. Generally, before buying an IP, user will analyze IPs from different IP vendors for performance comparison. This competitive scenario does not leave a big margin (gate-count) for Trojans.

Using RLUT, we can design extremely lightweight hardware Trojan payload as we can reconfigure the same LUTs, used in the crypto-algorithm implementation, from

correct value to malicious value. This reduces the overhead of the hardware Trojan and makes it less susceptible to detection techniques based on visual inspection [277]. We can also restore the original value of RLUT to remove any trace of Trojan, of course, at minor overheads. An IP designer can easily replace a normal LUT with RLUT. In this case, the designer has only one restriction of replacing a LUT implemented in *SLICE_M*. It is not difficult to find such a LUT in a medium to big-scale FPGA which is often the case with cryptographic modules. Moreover, if the designer chooses to insert the trojan at RTL level, the present restriction would not even apply.

Instantiation of *CFGLUT5* does not report any special element in the design summary report, **but a LUT modeled as *SRL32***. A shift register has many usages on the circuit. For example, a counter can be very efficiently designed on a shift register using one hot encoding. Moreover, lightweight ciphers employs extensive usage of shift registers for serialized architectures. Thus any suspicion of malicious activity will not arise in the user's mind by seeing the design report.

The only requirement is efficient triggering and a reconfiguration logic which will generate the malicious value upon receiving trigger signal. However, in this paper we will show that once triggered, **malicious value for the hardware Trojan can be generated without any overhead**, thus giving us extremely lightweight and stealthy design of hardware Trojans. The basic methodology is same for all the Trojans, which can be tabulated as follows:

- Choose a sensitive sub-module of the crypto-algorithm. For example, one can choose a 4×4 Sbox (can be implemented using 2 LUTs) as the sensitive sub module.
- Replace the LUTs of the chosen sub-module with *CFGLUT5s* without altering the functionality. A 4×4 Sbox can also be implemented using two *CFGLUT5*.
- Modify the *INIT* value upon trigger. As shown in Fig. 11.1, reconfiguration in *CFGLUT5* takes place upon receiving the *CE* signal. By connecting the trigger output to the *CE* port, an adversary can tweak the *INIT* value of *CFGLUT5* and can change it to a malicious value. For example, the 4×4 Sboxes, implemented using *CFGLUT5* can be modified in such a way that non-linear properties of the Sboxes get lost and the crypto-system becomes vulnerable to standard cryptanalysis. The malicious *INIT* value can be easily generated by some nominal extra logic. However, in the subsequent sections, we will show that it is possible to generate the malicious *INIT* value without any extra logic.
- Upon exploitation, restore original *INIT* value.

11.3.2. Trigger Design the Hardware Trojans

A trigger for a hardware Trojan is designed in a way that the Trojan gets activated in very rare cases. The trigger stimulus can be generated either through output of a sensor under physical stress or some well controlled internal logic. The complexity of trigger circuit also depends on the needed precision of the trigger in time and space. Several innovative and efficient were introduced as a part of Embedded Systems Challenge (2008) where participants were asked to insert Trojans on FPGA designs. For instance, one of the the proposition was *content & timing* trigger [275], which activates with a correct combination of input and time. Such triggers are considered practically impossible to simulate. Other triggers get activated at a specific input pattern. A more detailed analysis with example of different triggering methodologies and their pros and cons can be found in [278].

Moreover, modern devices are loaded with physical sensors to ensure correct operating conditions. It is not difficult to find voltage or temperature sensors in smartcards or micro-controllers. Similarly, FPGA also come with monitors to protect the system for undesired environmental conditions, Virtex-5 FPGAs contain *system monitor*. Though system monitor is not a part of cipher, they are often included in the SoC for tamper/fault/ temperature variation detection. These sensors are programmed to raise an alarm in event of unexpected physical conditions like overheating, high/low voltage etc. Now an adversary can use this system monitor to design an efficient and stealthy hardware Trojan trigger methodology. The trick is to choose a trigger condition which is less than threshold value but much higher than nominal conditions. For instance, a chip with nominal temperature of $20-30$ and safety threshold of 80, can be triggered in a small window chosen from the range of $40-79$. Similarly, user deployed sensors like the one proposed in [279] can also be used to trigger a Trojan. In our case study, we used the temperature sensor of Virtex-5 FPGAs *system monitor* to trigger the Trojan, more precisely on SASEBO-GII boards. The heating required to trigger the Trojan can be done by a simple $5 hair-dryer easily available in the market. The triggering mechanism is explained in Appendix .8. In the following to not deviate from the topic, we focus mainly on the payload design of the Trojan using RLUT. We let the designer choose any of the published techniques (including one proposed in Appendix .8) or innovate one. We precisely propose the design of the Trojan and the required triggering conditions.

11.3.3. Trojan Description

Before designing Trojan payload for a given hardware, we first demonstrate the potential of RLUT in inserting malicious activity. Let us consider a buffer which is a very basic gate. Buffers are often inserted in a circuit by CAD tools to achieve

desired timing requirements. For FPGA designers, another equivalent of buffer is route-only LUT. These buffers can be inserted in any sensitive wires without raising an alarm. In fact, sometimes the buffers might already exist.

These buffers are implemented in a *LUT6* with `INIT=0xAAAAAAAAAAAAAAAA` and can be easily replaced by *CFGLUT5*. A simple Trojan would consist in changing the *INIT* value of CFGLUT5 to `0xAAAAAAAA` and feedback CD_O output to CD_I input (see Fig 11.1). The *CE* input is connected to the trigger of the Trojan. Now, when the Trojan is triggered once (one clock), *INIT* value changes to `0x55555555` which changes the functionality of the gate to **inverter**. Another trigger brings back the *INIT* value to `0xAAAAAAAA` i.e., a **buffer**. The operations are illustrated in Fig. 11.4, where red block shows Trojan inverter and black blocks show a normal buffer. Thus by precisely controlling the trigger, an adversary can interchange between a buffer and inverter. Such a Trojan can be used in many scenarios like injecting single bit faults for Differential Fault Attacks [208] or controlling data multiplexers or misreading status flags, etc.

In the above example, we see how a buffer can be converted to an inverter by reconfiguring the *CFGLUT5* upon receiving the trigger signal. One important observation is that we do need need any extra reconfiguration logic to modify the *INIT* value of the *CFGLUT5*. The modification of the *INIT* value is achieved by the connecting the reconfiguration input port CD_I to the reconfiguration data output port CD_O. In other words, we can define the malicious *INIT* value in following way

$$INIT_{malicious} = CS_i(INIT_{normal})$$

where CS_i denotes cyclic right shift by i bits. The approach of RLUT is harder to detect because the malicious payload does not exist in the design. It is configured when needed and immediately removed upon exploitation. In normal LUT, the malicious design is hardwired (requires extra logic) and risk detection, whereas RLUT modifies existing resources and enables us to design design hardware Trojans without any extra reconfiguration logic. We will use similar methodologies for all the proposed hardware Trojans in this paper.

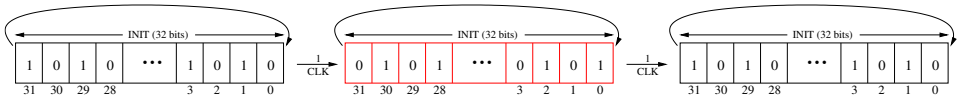


Figure 11.4: Operations of CFGLUT5 to switch from a buffer to inverter and back

Next, we target a basic *AES* IP as a Trojan target. The architecture of the AES design is shown in Fig. 11.5. The AES takes 128 bits of plaintext and key as input and produce 128 bit cipher-text in 11 clock cycles. The control unit of the AES encryption engine is governed by a 4 bit mod-12 counter and generates three different control signals which are as follows:

1. *load:* It is used to switch between plaintext and MixColumns output.

During the start of the encryption, this signal is made high to load the plaintext in the AES encryption engine.

2. *S.R/M.C:* It is used to switch between the ShiftRows and MixColumns output in the last round of AES.

3. *done:* It is used to indicate the end of encryption.

These signals are set high for different values of the counter. In our Trojan design, we mainly target the control unit of the AES architecture to disrupt the flow of the encryption scheme so that we can retrieve the AES encryption key. For this, we have developed four different Trojans and have deployed them on the *AES* implementation. The objective of the developed Trojan is to retrieve the *AES* key with only one execution of hardware Trojan or single bad encryption. Indeed, it has been shown that only one faulty encryption, if it is accurate in time, suffices to extract a full 128-bit key [280]. Triggering conditions can be further relaxed if several bad encryptions are acceptable. Each Trojan has trigger with different pulse-width or number of clock cycles. For different payloads, the RLUT content varies, hence variation in the trigger.

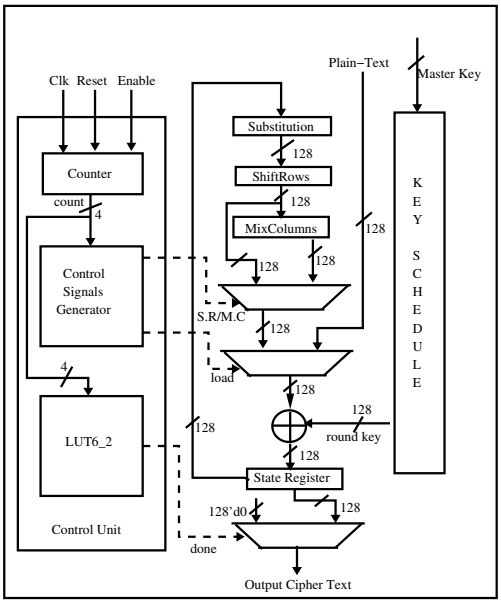


Figure 11.5: AES architecture without any Hardware Trojans

The detailed description of the developed Trojans are as follows:

Trojan 1

As we have stated earlier, the control unit of *AES* is based on a counter which also generates a *done* flag to indicate completion of the encryption cycle and is set to high only if counter value reaches 11. Signal *done* as shown in Fig. 11.5, is driven by a LUT6_2, which takes 4 bit counter value as input, and under normal operation it should contain *INIT* value 0x00000800 (it means only the 11^{th} bit is set to one i.e. condition required for *done* signal). To insert a Trojan we replace this LUT with *CFGLUT5* with INIT=0x80000800. It must be noted that though the *INIT* value of LUT6_2 and *INIT* value of CFGLUT5 is not same, both will essentially produce the same output upon receiving the 4 bit counter value. This is because truth-table of a function of 4 variables consists of 16 bits only, hence any change in the upper 16 bits of the *INIT* value will not change the functionalities of the LUT.

The CD_O output is feedback into CD_I input as in the example above. A trigger of 2 clock cycles at the *CE* input activates the Trojan (INIT=0x00002002) and produces the round 0 output (at round 0, counter value is 1) as the ciphertext. By knowing the plain-text, one can easily extract the full key with one wrong encryption. Again, we can see that malicious value of the *INIT* is generated by cyclic shift of the original *INIT* value of the *CFGLUT5*, hence we do not need any extra logic to generate the new *INIT* value. After extracting the key, a trigger of 10 clock cycles will restore the normal operations of the AES (INIT=0x00800800). This *INIT* value need not to be the same value, with which we started the computation (INIT=0x80000800), as long as the LUT generates correct output. The transition of *INIT* to activate the Trojan and restore back is shown in Fig 11.7(a) and the modifications in the AES architecture is shown in Fig. 11.6.

In the above Trojan description, we need 2 clock cycles to modify the *CFGLUT5* to malicious Trojan configuration and 10 clock cycles to restore it to the original correct value. So in total, we require 12 clock cycles.

Keeping this in mind, we have implemented three different versions of the same Trojan, depending on the precision of the trigger.

1. **Trojan 1a** needs a 1 cycle trigger synchronized with the start of the encryption. This trigger is used to enable a FSM which generates 12 clock cycles for *CE* of the $CFGLUT$, in order to activate the Trojan and restore it back after exploitation. Because of this, the overhead of the developed Trojan is 6 **LUTs and** 4 **flip-flops**.

2. **Trojan 1b** is a **zero** overhead Trojan. It assumes an adversary to be slightly stronger than Trojan 1a who can generate a trigger signal active for precisely 12 cycles and synchronized with the start of encryption.This overhead is absent in Trojan 1b as the trigger itself act as the

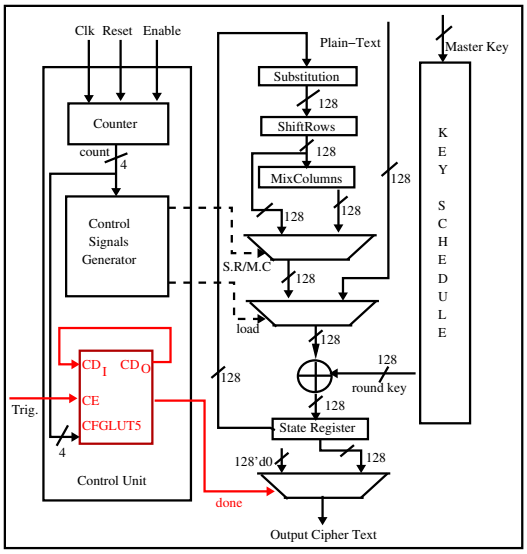


Figure 11.6: AES architecture with Trojan 1 CE signal of RLUT.

3. **Trojan 1c** relaxes the restriction on the adversary seen at previous case. It assumes that there are some delays of $n \gg 10$ clock cycles between two consecutive encryption. The choice of $n \gg 10$ is due to the fact that we need 2 clock cycles to reconfigure the RLUT into malicious Trojan payload, and 10 clock cycles to restore it back to good value. Hence the gap between two consecutive AES encryption should be greater than 10. The adversary provides a trigger of two clock cycles (not necessarily consecutive) before the start of current encryption. After the faulty encryption is complete, the adversary generates 10 trigger cycles (again not necessarily consecutive) to restore back the cipher operations. The overhead for this Trojan is 2 **LUTs** , due to routing of RLUT.

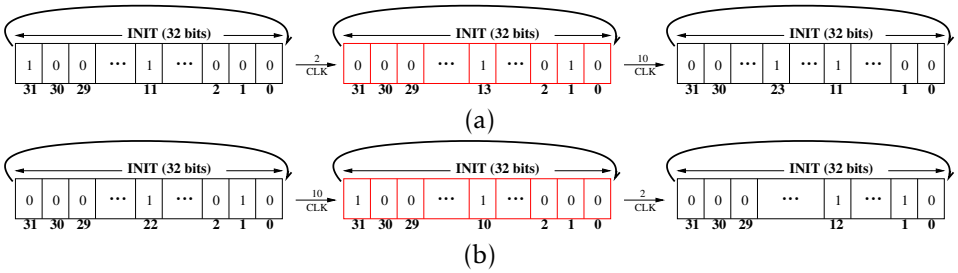


Figure 11.7: Operations of CFGLUT5 to activate the Trojan and restore to normal operations for (a) Trojan 1; (b) Trojan 2. Bit positions not shown contain '0'

Trojan 2

This Trojan targets a different signal in the control unit of the *AES* design. As shown in Fig. 11.5, the design contains a multiplexer which switches between MixColumns output and input plaintext depending on the round/count value. The output of the multiplexer is produced at input of AddRoundKey operation. Under normal operation, multiplexer passes the input plaintext in round 0 (*load* signal of multiplexer is set to 1) and MixColumns output (ShiftRows output in the last round) in other rounds (*select* signal of multiplexer is set to 0). To design the Trojan, we have replaced the LUT6_2 (with INIT=0x00000002) which generates *load* signal of the multiplexer with *CFGLUT5*, containing INIT= 0x00400002. As we have observed for Trojan 1, the difference in the *INIT* value in LUT6_2 and *INIT* value of *CFGLUT5* will essentially produce the same output.

In this case also CD_O port of *CFGLUT5* is connected to CD_I port, enabling cyclic shift of the *INIT* value. Upon a trigger of 10 clock cycles, the *INIT* value gets modified to INIT=0x80000400 (it means load will set to one during the last round). This actually change the multiplexer operation, modifying it to select the plaintext in the last round computation. From the resulting ciphertext of this faulted encryption, we can easily obtain the last round key, given the plaintext. Further a trigger of 2 clock cycles restores the normal operation (INIT=0x00001002) as shown in Fig 11.7(b). Again the value over bit position 12 is not a problem as the *select* signal is controlled by a mod-12 counter and the value is never reached. The counter value 0 indicates idle state, $1-10$ encryption and 11 indicates end of encryption. This Trojan also has a **zero** overhead as reconfiguration of the CFGLUT5 is obtained by cyclic right shifting of *INIT*. Hence, the triggering cost is same as Trojan 1b.

Tab. 11.1 summarizes the nature, trigger condition and cost of the four Trojans.

Table 11.1: Area overhead of the Trojans on Virtex-5 FPGA. Trigger is given in clock cycles and *s* subscript indicates trigger must be consecutive synchronized with the start of encryption.

Trojan	Trigger	LUT	Registers	Payload Overhead	Frequency (MHz)
AES (No Trojan)		1594	260	*X*	212.85
Trojan 1a	1_s	1600	264	6 LUTs & 4 flip-flops	212.85
Trojan 1b	12_s	1594	260	0	212.85
Trojan 1c	12	1596	260	2 LUTs	212.85
Trojan 2	12_s	1594	260	0	212.85

The above described Trojans can also be designed using normal LUTs also. The zero overhead Trojans described above can be designed using 2 LUT overhead (One LUT for Trojan operation and other for selecting between Trojan and normal oper-

ations). But such Trojan designs can be easy to detect as the Trojan operated LUT is always present on the design unlike CFGLUT5, where the Trojan operated LUT is created by run time reconfiguration.

11.4. Constructive Applications for RLUT

In the previous section, we discussed some application of RLUT for hardware Trojans into third party IPs. However, RLUT do have a brighter side to their portfolio. The easy and internal reconfigurability of RLUT can surely be well exploited by the designers to solve certain design issues. In the following, we detail two distinct cases with several applications, where RLUT can be put to good use.

11.4.1. Customizable Sboxes

A common requirement in several industrial application is dynamic or cutomizable substitution boxes (Sboxes) of a cipher. One such scenario which is often encountered by IP designers who design **secret ciphers** for industrial application. A majority of secret ciphers use a standard algorithm like AES with modified specification like custom Sboxes or linear operations. Sometimes the client is not comfortable to disclose these custom specifications to the IP designer. Common solutions either have a time-space overhead or resort to dynamic reconfiguration, to allow the client to program secret parameters at their facilities. A RLUT can come handy in this case.

There are several algorithms where the Sboxes can be secret. The former Soviet encryption algorithm GOST 28147-89 which was standardized by the Russian standardization agency in 1989 is a prominent example [281]. The A3/A8 GSM algorithm for European mobile telecommunications is another example. In the field of digital rights management, Cryptomeria cipher (C2) has a secret set of Sboxes which are generated and distributed to licencees only.

There are certain encryption schemes like DRECON [282], which offers DPA resistance by construction, exploiting tweakable ciphers. In this scheme, users exchange a set of tweak during the key exchange. The tweak is used to choose the set of Sboxes from a bigger pool of precomputed Sboxes. In the proposed implementation [282], the entire pool of Sboxes must be stored on-chip. Using RLUT, the Sboxes can be easily computed as a function of the tweak and stored on the fly. Similarly, a low-cost masking scheme RSM [264] can also benefit from RLUT to achieve desired rotation albeit at the cost of latency. Thus there exist several applications where customizable Sboxes are needed.

Architecture of Sbox Generator:

As a proof of concept, we implement the Sbox generation scheme of [282]. The original implementation generates a pool of 32 4×4 Sboxes and stores it into BRAMs, while only 16 are used for a given encryption. It uses a set of Sboxes which are affine transformations of each other. For a given cryptographically strong Sbox $S(\cdot)$, one can generate 2^n strong Sboxes by following: $F_i(x) = \alpha S(x) \oplus i$ for all $i = 0, \cdots, 2^n - 1$, where α is an invertible matrix of dimension $n \times n$. α can also be considered a function of the tweak value t i.e. $\alpha = f(t)$. Since affine transformation does not change most of the cryptographic properties of Sboxes, all the generated Sboxes are of equal cryptographic strength [282].

The sbox computation scheme of [282] can be very well implemented using RLUT as follows. The **main objective** of this Sbox generator is to compute a new affine Sbox from a given reference Sbox, and store it in the same location. The architecture is shown in Fig 11.8. As we have stated earlier, each *CFGLUT5* can be modeled as 2 output 4 input function generator, we can implement a 4×4 Sbox using two *CFGLUT5* as shown in Fig 11.8. We consider that the reference 4×4 Sbox is implemented using 2 *CFGLUT5*. We compute the new Sbox and program it in the same 2 *CFGLUT5*. The reconfiguration of the Sbox is carried through following steps:

1. Read the value of the Sbox for input 15.
2. Compute the new value (4-bits {3,2,1,0}) of the Sbox using affine transformer for the Sbox input 15.
3. Now *CFGLUT5* is updated by the computed value, 2 bits for each *CFGLUT5* ({3,2},{1,0}). However, only one bit can be shifted in *CFGLUT5* in one clock cycle. Hence we shift in two bits, 1-bit in each CFGLUT5 ({0,2}) and store the other 2-bit ({1,3}) in two 16 bit registers.
4. After the 2-bits ({0,2}) of new value of Sbox is shifted in to position 0 of each *CFGLUT5*, old value for the position 15 is flushed out. The old value at position 14 is moved up to position 15. Thus the address is hard-coded to 4'd15.
5. Repeat steps 1 – 4 until whole old Sbox is read out i.e. 16 clock cycles.
6. After 16 clock cycles, we start to shift in the data which we stored in the shift register bits ({1,3}) for 16 Sbox entries, which takes another 16 clock cycles. This completes Sbox reconfiguration.

The architecture requires **56 LUTs, 38 flip-flops with a maximum operating frequency of 271 MHz.** To reconfigure one Sbox, we need 32 clock cycles. Now depending on the application and desired security the sbox recomputation can be

done after several encryption or every encryption or every round. It is a purely security-performance trade-off.

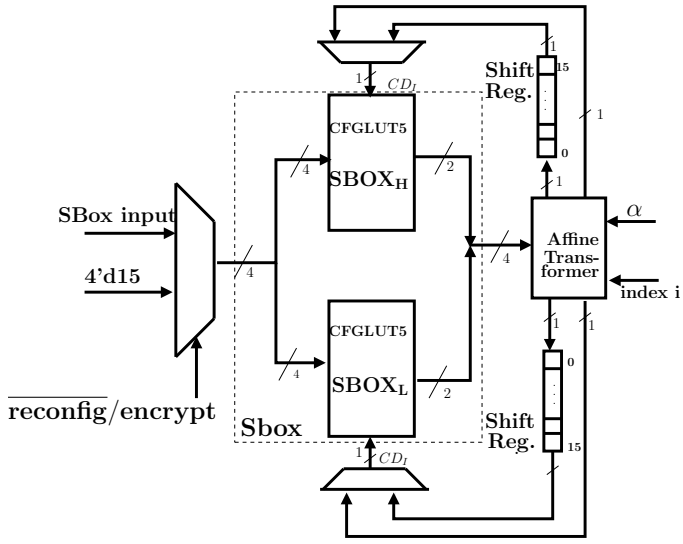


Figure 11.8: Architecture of Sbox Computation using affine transformation and storing in RLUT

11.4.2. Sbox Scrambling for DPA Resistance

RLUT also have the potential to provide side-channel resistance. The reconfiguration provided by RLUT can be very well used to confuse the attackers. A beneficial target would be the much studied masking countermeasures [263] which suffer from high overhead due to the requirement of *regular mask refresh*. One of the masking countermeasures which was fine-tuned for FPGA implementation is Block Memory content Scrambling (BMS [263]). This scheme claims first-order security and, to our knowledge, no practical attack has been published against it. However, Sbox Scrambling using BRAM is inefficient on lightweight ciphers with 4X4 sboxes due to underutilization of resources. Hence we propose a novel architecture using RLUT to address this. Nevertheless, this mechanism can easily be translated to AES also.

The side channel countermeasure using RLUT, shown in [270] is different from the proposed design architecture. The design of [270] implements standard Boolean masking scheme, where each round uses a different mask. Here, we propose a lightweight architecture of SBox scrambling scheme presented in [263]. These two countermeasures have similar objectives but quite different designs.

The BMS scheme works as follows: let $Y(X) = P(SL(X))$ be a round of block cipher,

where X is the data, $P(\cdot)$ is the linear and $SL(\cdot)$ is the non-linear layer of the block cipher. For example in PRESENT cipher [283], the non-linear layer is composed of 16 4×4 Sboxes and the linear layer is bit-permutation. According to the BMS scheme, the masked round can be written as $Y_M(X) = P(SL_M(X_M))$, where X_M is masked data $X \oplus M$ and $SL_M(\cdot)$ is the Sbox layer of 16 scrambled Sbox. Now each Sbox $S_m(\cdot)$ in SL_M is scrambled with one nibble m of the 64-bit mask M. The scrambled Sbox $S_m(\cdot)$ can be simplified as $S_m(x_m)) = S(x_m \oplus m) \oplus P^{-1}(m)$, where x is one nibble of round input X. Next in a dual-port BRAM which is divided into an active and inactive segment, where the active segment contains $SL_{M0}(\cdot)$ i.e. Sbox scrambled with mask $M0$ is used for encryptions. Parallely, another Sbox layer $SL_{M1}(\cdot)$ scrambled with mask $M1$ is computed in an encryption-independent process and stored in the inactive segment. Every few encryption, the active and inactive contents are swapped and a new Sbox scrambled with a fresh mask is computed and stored in the current inactive segment. This functioning is illustrated in Fig. 11.9.

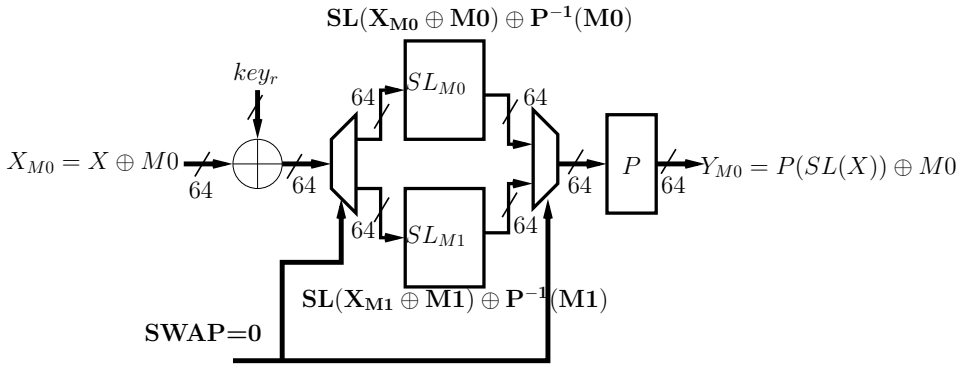


Figure 11.9: Architecture of Modified PRESENT Round. SL_{M0} is the (precomputed) active SLayer while SL_{M1} is being computed as in Fig. 11.10.

BMS is nice countermeasure and shown to have reasonable overhead of 44% for LUTs, 2× BRAMs and roughly 3× extra flip-flops in FPGA. Another advantage of BMS is that it is generic i.e., it can be applied to any cryptographic algorithm. BMS can be viewed as a *leakage resilient* implementation, where the cipher is not called enough with a fixed mask for an attack to succeed. The memory contexts are swapped again with a fresh mask. However, for certain algorithms BMS could become unattractive. For example in a lightweight algorithm like PRESENT, a 4×4 Sbox can be easily implemented in 4 LUTs. In newer FPGA families which support 2-output LUT, 2 LUTs are enough to implement a Sbox. Using a BRAM in such a scenario would lead to huge wastage of resources.

11

Sbox Scrambling using RLUT:

In the following, we use RLUT to implement BMS like countermeasure. Precisely we design a PRESENT cryptoprocessor protected with a BMS like scrambling scheme

but using RLUTs to store scrambled Sboxes. Rest of the scheme is left same as [263]. The architecture of Sbox scrambler using RLUT is shown in Fig 11.10. $SBOX_P$ is the PRESENT Sbox. A mod16 counter generates the Sbox address $ADDR$ which is masked with Mask m of 4-bits. The output of Sbox is scrambled with inverse permutation of the mask to scramble the Sbox value. Please note the the permutation must be applied on the whole 64-bits of the mask to get 4-bits of the scrambling constant for each Sbox. Each output of the scrambler is 4-bits. As stated before, each 4×4 Sbox can be implemented in 2 CFGLUT5 each producing 2-bits of the Sbox computation. Let us call the CFGLUT5 producing bits $0, 1$ as $SBOX_{ML}$ and bits $2, 3$ as $SBOX_{MH}$. The 4-bit output of the scrambler is split into two buses of 2-bits ({3,2},{1,0}). Bits {3,2} and {1,0} are then fed to the CD_I of $SBOX_{ML}$ and $SBOX_{MH}$ respectively, through a FIFO. The same scrambler is used to generate all the 16 Sboxes one after the other and program CFGLUT5. In total it requires 16×32 clock cycles to refresh all 16 inactive Sboxes. We implement two parallel layers of SBoxes. When active layer is computing the cipher, inactive is being refreshed. Thus cipher operation is not stalled. 16×32 clocks (16 encryptions) are needed to refresh the inactive layer and this means that we can swap active and inactive SBoxes after every 16 encryptions. Swap means that active SBox become inactive and vice versa. The cipher design uses active SBox only. The area overhead comes from the scrambler circuit and multiplexers used to swap active/inactive Sboxes. We implemented a PRESENT crypto-processor and protected it with Sbox scrambling countermeasure. The area and performance figures of original design and its protected version are summarized in Tab. 11.2.

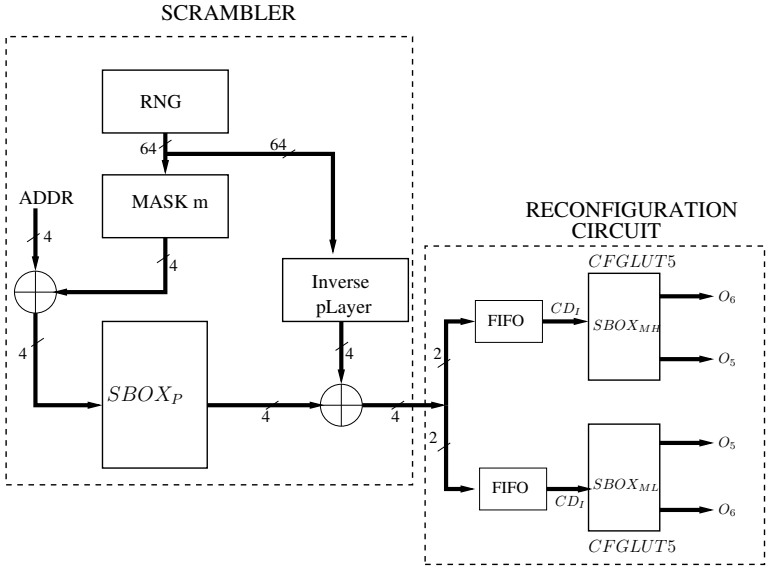


Figure 11.10: Architecture of Sbox Scrambler

Table 11.2: Area and Performance Overhead of Scrambling Scheme on Virtex-5 FPGA

Architecture	LUTs	Flip-flops	Frequency (MHz)
Original	208	150	196
Scrambled	557	552	189
Overhead	2.67×	3.68×	1.03×

11.5. Conclusions

This paper addresses methods to exploit reconfigurable LUTs (RLUTs) in FPGAs for secure applications, with both views: destructive and constructive. First it has been shown that the RLUT can be used by an attacker to create Hardware Trojans. Indeed the payload of stealthy Trojans can be inserted easily in IP by untrusted vendors. The Trojans can be used to inject faults or modify the control signals in order to facilitate the key extraction. This is illustrated by a few examples of Trojans in AES. Second the protective property of RLUT has been illustrated by increasing the resiliency of the Sboxes of cryptographic algorithms. This is accomplished either by changing dynamically the Sboxes of customized algorithms or scrambling the Sboxes of standard algorithms.

To sum up, this paper clearly shows that RLUT is a double-edged sword for security applications on FPGAs. Due to the obvious positive application of RLUTs in security, one cannot simply restrict the use of RLUT in secure applications. This motivates further research in two principal directions. Firstly, there is need for Trojan detection techniques at IP level. This detection techniques should be capable of distinguishing a RLUT based optimizations from potential Trojans. Finally certain new countermeasures totally based on RLUTs, including the trigger part, can be studied.

V

Security design principles

This section refers to the ellipse located at the center of Figure 1.1, which represents the process of incorporating greater rigor in the design of systems with security constraints. Here, we examine the concept of security from a designer's viewpoint and provide a methodology and a set of guidelines for designers seeking to create secure systems.

The goal of the methodology outlined in this section is to assist designers in producing systems with a higher level of security by systematically identifying and addressing potential vulnerabilities in the design process. This approach involves a thorough understanding of the system being designed, the potential threats it may face, and the security requirements it must meet.

The set of guidelines presented in this section is intended to provide designers with a framework for designing secure systems. These guidelines cover various aspects of the design process, including threat modeling, system architecture, cryptographic algorithms and protocols, implementation, and testing.

By following these guidelines, designers can ensure that the security of their systems is not an afterthought, but rather a central consideration from the beginning of the design process. This approach results in systems that are more secure, more reliable, and less vulnerable to attack.

12

Security *is* an Architectural Design Constraint

In state-of-the-art design paradigm, time, space and power efficiency are considered the primary design constraints. Quite often, this approach adversely impacts the security of the overall system, especially when security is adopted as a countermeasure after some vulnerability is identified. In this position paper, we motivate the idea that security should also be considered as an architectural design constraint in addition to time, space and power. We show that security and efficiency objectives along the three design axes of time, space and power are in fact tightly coupled while identifying that security stands in direct contrast with them across all layers of architectural design. We attempt to prove our case utilizing a *proof-by-evidence* approach wherein we refer to various works across literature that explicitly imply the eternal conflict between security and efficiency. Thus, security has to be treated as a design constraint from the very beginning. Additionally, we advocate a security-aware design flow starting from the choice of cryptographic primitives, protocols and system design.

12.1. Introduction

Historically, design of computer systems have revolved around the notion of performance, primarily measured in terms of *time* and *space* efficiency. It was only with the advent of low-footprint portable devices during the 80's that *power* featured as a potential measure of performance. Careful scrutiny and evaluation over a couple of decades confirmed that power efficiency complements and contradicts

the notions of time-and-space efficiency in practical systems, and hence, it should be considered as a fundamental constraint in design. We have come a long way since then, and for over a decade, we have unequivocally considered *time*, *space* and *power* as the three primary constraints of architectural design. Orthogonally, one can also observe the exponential growth of Cyber-Physical Systems (CPS) and Internet-of-Things (IoT), serving as the best examples of increasing heterogeneity in existing digital infrastructure that has become pervasive in our lives. As these systems are expected to be deployed in critical infrastructure and adverse environments, the *security* of such systems demand serious attention. During the last decade, the boom of heterogeneous architecture has triggered several research initiatives towards the development of so called *security-aware* systems [284, 285]. In this paper, we argue that it is time to consider the notion of security as the fourth axis in the space of architectural design, instead of incorporating security as a mere add-on feature over existing systems.

Security as a Design Axis

Security as a design axis seems to both complement and contradict the other three axes of power, space and time. The security axis is strikingly peculiar, as one is always concerned about meeting the lower bound (e.g., 128/192/256 bit security), usually fixed keeping in mind the the attacker's capabilities, while not worrying about improving it any further as it is generally considered an overkill, taking a toll on the design parameters on the other three axes. Thus, optimal design points usually lie within a very thin strip over the security axis, just satisfying the strict lower bound. In contrast, designers always strive to improve the design parameters over the other three axes to the maximum extent, allowing themselves a much wider working range. In other words, designs are always expected to be as *time-efficient*, *space-efficient* and *power-efficient* as possible, but in terms of security, designs are expected to be *security-compliant*.

While it is possible to derive an analytic relationship between the tightly coupled time-space-power design parameters, incorporation of security into the same relationship might not be possible. This makes it difficult to understand the relationship of the security design axis with the other three. However, it is intuitive, and we would like to hypothesize that the security axis always contradicts the other three axes. In order to argue our case, we adopt the strategy of *proof-by-evidence* by citing enough examples from literature that illustrate the various instances of the eternal conflict between security and efficiency. It is indeed not possible to cite all attacks, and we do not claim that all these attacks were *caused* by the push for efficiency. We would simply like to establish that incorporation of techniques to improve efficiency along any of the time-space-power design axes has to be done with extreme caution, so that it does not compromise or weaken the security of the

system. We do not claim that this is always the case with efficiency, but only advice the designer to employ time-space-power optimization techniques with extreme caution.

We present the notion of *security* across three layers — cryptographic primitives, security protocols and security systems — as depicted in Table 12.1. It is evident from the literature that there have been various reported cryptanalytic results and security breaches at each of these three layers and we identify those instances that clearly demonstrate the trade-off between efficiency and security. Quite often, architectural designs focussed on the fundamental principles of time-space-power efficiency introduce security vulnerabilities in cryptographic primitives, security protocols and security systems. Most critical vulnerabilities are generally noticed in the systems layer, spread across the range of hardware systems, software systems and hardware-software interfaces, where the time-space-power design constraints are considered to be of highest priority. This is where we advocate the inclusion of *security* as a fundamental architectural constraint to complete the design landscape.

Primitive Level	Mathematical Models	Pseudo-Random Generators, Functions, and Permutations
	Key-usage Paradigms	Symmetric Key Cryptography, Asymmetric Key Cryptography
	Cryptographic Modules	Block Ciphers, Stream Ciphers, Hash Functions, Signatures, etc.
	Cryptographic Modes	Encryption, Authentication, Authenticated Encryption, etc.
Protocol level	Transport Layer Security, Secure Sockets Layer, Secure Shell, IP Security, Wireless Security, etc.	
System Level	Software Abstraction, Hardware Abstraction, Software-Hardware Interface, Operating Systems, etc.	

Table 12.1: The layers of Security — Cryptographic Primitives, Security Protocols and Security Systems

Security-Efficiency Trade-Off

The trade-off between security and efficiency is as old as the dawn of cryptography. Formal notion of information-theoretic security, as introduced by Shannon in 1950s, warrants the use of perfectly random one-time pads, which are absolutely useless in terms of practical efficiency. Cryptography practitioners, hence, introduced the notion of pseudo-random generators to approximate the desirable properties of one-time pads, and we followed the path of computational security. In a similar vein, we discarded the inefficient symmetric key-exchange mechanism

of Merkle Puzzles that provided a quadratic computational gap between the users and the adversary to adopt the efficient asymmetric key-exchange mechanism like Diffie-Hellman, providing an exponential advantage to the users. Even today, most of the theoretical proofs in security rely on the random-oracle property of compression functions, whereas practical instantiations could only remain efficient till the construction of standard hash functions. It is evident that security and efficiency do not go hand-in-hand. In this paper, we provide practical evidences to present a two-way argument — on one hand, security generally comes at the cost of efficiency, while on the other, efficiency may not always come at the cost of security, but the designers should be extremely cautious of such a possibility. We will henceforth adopt a top-down approach in this paper, wherein we refer to practical instances where this security-efficiency trade-off can be observed at various layers of the stack — starting with the abstraction of cryptographic primitives, progressively moving down to the implementations.

Depending on the application and the use case in hand, a secure design always looks to achieve a multitude of different objectives like fast performance, low resource utilization, low power consumption and many more. In this paper, we will concentrate on the three fundamental efficiency parameters — time, space and power. In the quest for optimizing these resources, practical instantiations of secure algorithms often render themselves vulnerable. There are various categories of such vulnerabilities, some of which are as follows:

- Inherent security-efficiency trade-off is done when deciding the various parameters for a given secure algorithm.
- Efficient instantiations of a given secure algorithm might yield very good performance compared to a random or a generic instantiation of the same, but the same efficient instance might pave way for unforeseen vulnerabilities.
- Cryptographic primitives when implemented in a *standalone* mode may be secure, but quite often, an efficient encapsulation of the primitive into a broader class of security protocols might lead to vulnerabilities.
- Careless optimization technique implemented on a secure algorithm might lead to leakage of information.
- Generic security-agnostic performance enhancement approach developed for a specific platform might lead to creation of side channels, thus weakening the implementation of any secure algorithm on the same platform.
- Certain efficient implementation strategies providing time-efficiency open gates to side channel leakage.

– Optimizations employed by (semi-)automated tools over a given implementation of a secure algorithm (in most of the case, the source code) might discard inefficient features that ensured security in the first place.

We broadly classify the literature of security vulnerabilities introduced due to performance improvements into three categories of efficiency — time, space and power — affecting the three layers of security — primitives, protocols and systems. The cross-layer cross-category taxonomy in context of this paper is set as XXX.YY, where XXX denotes the affected security layer, and YY denotes the efficiency node, which causes the security loophole. Table 12.2 provides a brief description of the various types of instances that exhibit the trade-off between security and efficiency obseravble across multiple levels of the applied cryptography stack. Sections 12.2, 12.3 and 12.4 present in details the evidences of security-efficiency trade-off from the literature in a more systematic format (layer-wise) to support our argument.

	TE	SE	PE
PRI	*Time-Efficiency* vs Security trade-off observable at the *Primitive* level Ref.: [286–299]	*Space-Efficiency* vs Security trade-off observable at the *Primitive* level Ref.: [291, 296–315]	*Power-Efficiency* vs Security trade-off observable at the *Primitive* level Ref.: [298, 299, 305–315]
PRO	*Time-Efficiency* vs Security trade-off observable at the *Protocol* level Ref.: [316–318]	*Space-Efficiency* vs Security trade-off observable at the *Protocol* level Ref.: [319, 320]	*Power-Efficiency* vs Security trade-off observable at the *Protocol* level –
SYS	*Time-Efficiency* vs Security trade-off observable at the *System* level Ref.: [87, 321–350]	*Space-Efficiency* vs Security trade-off observable at the *System* level Ref.: [87, 330–333, 336–353]	*Power-Efficiency* vs Security trade-off observable at the *System* level Ref.: [354]

Table 12.2: Literature of Security-Efficiency trade-off : Layers of Security vs Efficiency Considerations

12.2. Primitive Level

The computational security notion governing the security of both private key and public key cryptographic primitives are quite well understood. While the security of public key cryptographic primitives are derived through polynomial time

reductions from provably hard mathematical problems, security of private key primitives are derived from constructions like Feistel structures and Substitution-Permutation networks governed by well defined mathematical concepts like confusion and diffusion. Though there have been a number of reported attacks and vulnerabilities of these primitives in literature [286, 292, 293], none of them are catastrophic but merely point out to the existence of certain corner cases, weak instances and insecure algorithmic optimizations. We would like to focus on such instances in this section that especially argue our case of the conflict between security and efficiency at the primitive level. We separately analyse classical public key, post-quantum public key and symmetric key cryptographic primitives.

12.2.1. Public Key Cryptography

The traditional public key cryptographic primitives like RSA and ECC based cryptographic systems used in almost all secure communications derive their security guarantees from hard problems based in the field of number theory. While the security of RSA depends on factorization of a product of two large prime numbers, ECC relies on the hardness of solving the discrete logarithm problem. Though the underlying hard problems of these schemes are rendered intractable by classical computers, a number of weaknesses and vulnerabilities are known to have been exploited leading to practical attacks on the RSA and ECC based cryptographic schemes. And following the argument of our paper, it is not surprising to know that many of those vulnerabilities stem from the presence of the cross-layer phenomenon between security and efficiency, which will be covered in the following discussion.

Exploiting Reduced Entropy in KeyPair Generation (Type PRI.TE)

Keypair generation is crucial in RSA and ECC based cryptography, and reuse of randomness is a common implementation strategy used to improve efficiency. But this technique does not have a good track record in security as it has led to a number of well known attacks.

Exploiting reuse of operating group and primes: Adrian et al.[286] reported the famous "LogJam" attack in 2015, an MITM (Man In The Middle) attack on TLS connections in which servers could be tricked into using "Export Grade" Diffie-Hellman that operated over 512-bit groups. The main vulnerability stemmed from the usage of same 512 bit group across 8.4% of Alexa Top Million websites and the same 1024 bit group 3.4% of all HTTP servers, thus a massive precomputation step could be used to amortize the attack time over multiple entities using the same group. Heninger et al. [287] performed the then largest network survey of TLS and SSH servers in 2012 and reported vulnerabilities due to usage of keys with insuffi-

cient entropy and usage of same key in shared hosting conditions. Another similar vulnerability due to reuse of *ephemeral* keys in Elliptic Curve Digital Signature algorithm was reported by a hacker group named *FailOverflow* on Sony PlayStation 3.

Exploiting use of efficient prime generation algorithms: Švenda et al.[288] performed statistical analysis on a large number of public key and moduli used for RSA generated from a variety of cryptographic libraries and smart cards and observed that a given key could be classified into its correct key source with a very high accuracy of 85%, thus exposing anonymity of users. This is due to the existence of multiple efficient algorithms for prime generation like random sampling method, incremental search algorithm, rejection sampling, use of "Square" regions etc. which leave an observable signature for themselves allowing for easy detection. The same authors further discovered that the prime generation algorithm used by the cryptographic library *RSAlib* from *Infineon Technologies AG* only generated primes that were of the form

$$p = k * M + (65537^a \bmod M)$$

wherein the RSA prime p generated only depends on a and k and M is known. The primes of this form were shown to be easily factorizable and also were easy to be fingerprinted due to the abnormal decrease in entropy.

Efficient Parameter Instantiations (Type PRI.TE)

Modular exponentiation used in RSA algorithms is very costly in terms of performance and resource utilization. Thus, use of efficient parameters to speed up implementations is very common. For example, use of a small secret key exponent for signatures will significantly speed up signature generation, but Wiener [289] showed that private key exponents satisfying the bound $d < N^{0.25}$ where d, N are the private key exponent and modulus respectively leads to a break of the RSA cryptosystem. by Boneh et al. [292] to $d < N^{0.292}$. Similarly, usage of a small private key exponent has been shown to be exploited by a number of attacks like Hastad Broadcast attack [294], partial key exposure attack [290] using variants of the Coppersmith's theorem.

Efficient Techniques for Modular Exponentiation (Type PRI.TE)

The *Chinese Remainder Theorem* (CRT) is a well known efficient technique to perform modular exponentiation which computes over primes half the size as that of the original modulus leading to a speed-up up to a factor of four. But Boneh et al. [295] showed that a single fault injected during computation using one of the prime factors in a CRT optimized RSA signature generation procedure results in trivial retrieval of the key from the faulty signature. A recent report by

Weimer [293] showed that this simple fault classical attack still poses a threat to real world systems using TLS with RSA signature schemes. The countermeasure against the fault attack leads to significant decrease in performance as it requires an additional signature verification and hence is not widely deployed.

12.2.2. Post Quantum Public Key Cryptography

The cross-layer phenomenon not only is observable in classical cryptography, but also extends its presence into post quantum cryptographic primitives. The cryptographic community is actively working towards standardization of quantum resistant public key cryptographic primitives, better known as "Post-Quantum" cryptography. There have been several proposals for post quantum cryptography from varied fields of mathematics among which lattice based cryptography and code based cryptography seem to be the more promising proposals that provide both quantum resistance guarantees along with practical efficiency comparable on a scale with traditional public key cryptography.

Lattice based cryptography (Type PRI.TE & PRI.SE)

Lattice based cryptography, in its infancy was considered to be near impractical due to the schemes suffering from asymptotically large key sizes and operation counts ($\mathcal{O}(n^2 log(n))$) where n is the security parameter. But, the security of these schemes were based on hard problems on general lattices which were considered to be $NP - Hard$ in the worst case, thus offering very good security guarantees. A lot of research then was focussed on increasing the efficiency of lattice based cryptographic schemes, with the main direction being development of schemes with hardness on algebraically structured ideal lattices [296, 297], yielding asymptotic efficiency in both space and time, with reduced key sizes and computation time ($\mathcal{O}(nlog(n))$), since arithmetic could be done over polynomials in rings as opposed to matrix vector arithmetic in the case of general lattices. This triggered a large body of work towards efficient implementation of lattice based cryptographic primitives on a range of devices from the smallest 8-bit AVR microcontrollers [355, 356] to reconfigurable hardware [357, 358]. But, the caveat present here is that the same hard problems over the structured ideal lattices which determines the security guarantees of these efficient schemes are not known to be as hard as that on general lattices. Even with extensive cryptanalytic efforts on these structured variants [359, 360], there are not any known weakness still known that could be exploited from their algebraic structure. With many of the efficient lattice based cryptographic schemes basing their security over hard problems on algebraically structured lattices [361, 362], cryptanalysis of lattice based cryptographic schemes will be intensely scrutinized over the coming years.

Code based cryptography (Type PRI.TE & PRI.SE)

One can also observe very similar trends in code based cryptography where there is a dilemma in a choice between structured but efficient instantiations as opposed to unstructured but inefficient instantiations of code based cryptographic schemes. The first code based cryptographic scheme, the McEliece encryption scheme [302] was proposed using binary Goppa codes, but this scheme suffered from large sizes for the public keys along with complex decoding procedures. A large body of work concentrated on development of efficient but secure choices of algebraically structured linear codes like Reed-Solomon [300], Reed-Muller codes [291], quasi-cyclic and quasi-dyadic codes [301] and many more. But, most of them are known to be broken with only the initial proposal of the Binary-Goppa codes [302] and the QC-MDPC codes [303] still considered to be secure. According to the state of the art, the QC-MDPC code based schemes offer very compact keys ($1 - 2KB$) while at the same time being very efficient, but have a certain error probability associated with their decryption procedures, which was shown to be exploitable through the GJS reaction attack reported in [304], provided the same key is used across many number of encryptions. But, the relateively inefficient binary Goppa code variant of the McEliece encryption scheme still stands unscathed even with about close to 40 years of cryptanalysis efforts.

12.2.3. Symmetric Key Cryptography

Security of Private-Key Primitives (Type PRI.SE & PRI.TE & PRI.PE)

The security of all symmetric key cryptographic primitives are directly related to the size of the shared secret, which is commonly indicated by the bit security level. A bit security of n bits indicates that a black box attacker has to perform at the most 2^n operations to retrieve the secret key. The bit security level is determined based on the best known attacks against the symmetric key primitive and thus need not be equal to the bit size of the secret. Moreover, due to the sustainable decrease in the cost of computational power, recommended security levels for various cryptographic applications are regularly increased, with the most recent instance being the declaration of any security level below 112 bits to be insecure according to NIST [15], thus phasing out the use of PRESENT-80 [305] and LED-64 [306] light weight block ciphers. Thus, upgrading the bit security level of any symmetric key primitive would indicate increasing the bit size of the key, implying larger storage, more operations on the key and ultimately a larger resource footprint.

The area of lightweight cryptography has attracted a lot of attention which has spurred the development of many light weight cryptographic designs like efficient block ciphers (PRESENT, LED, SIMON/SPECK [307], SKINNY [308], GIFT [309]), stream ciphers (Grain [310], Plantlet [311], Fruit [312], Lizard [313]) and Hash

Functions (PHOTON [314]). The main reason can be attributed to the emergence of embedded device technologies like Bluetooth, Internet-of-Things (IoT), Wireless Sensor Networks (WSNs), Wearable Devices etc. which primarily operate on low power over computationally constrained platforms. While most of these ciphers achieve competitive bit-security levels, they build upon less secure and more efficient building blocks leading to low resource consumption, but require a higher number of iteration rounds which adds up to processing time. Besides, since this field is relatively new, the security gap between these lightweight primitives and their old trusted counterparts (AES, SHA-2 [363] , SHA-3 [364]) has not been extensively studied, thus leading to restrain from using these lightweigtht designs from use in high security and sensitive applications.

Post-Quantum Security of Private-Key Primitives (Type PRI.SE & PRI.TE & PRI.PE)

Unlike public-key primitives, there are no known quantum attacks on private-key primitives except for Grover's Search Algorithm [298], which can speed up brute-force search attack from 2^n to $2^{n/2}$. Hence, post-quantum private-key primitives have to be at least twice as large as their classical counterparts in order to achieve the same security level, which, again, shows the trade-off between efficiency and security [299].

Evaluation of CAESAR Candidates (Type PRI.SE & PRI.TE & PRI.PE)

The CAESAR competition [365] was announced in 2013, to allow the academic community to choose a portfolio of authenticated encryption algorithms. Over 5 years, more than 50 submissions have been intensively studied, evaluating their security, software performance and hardware efficiency. In March, 2018, the CAESAR competition was concluded by selecting 7 final proposals, divided into three use cases — (a) Lightweight applications (resource constrained environments): ACORN and Ascon, (b) High-performance applications: AEGIS, MORUS and OCB, and (c) Defense in depth: COLM and Deoxys-II.

In [315], the authors have studied the hardware performance, area and efficiency of all the third round candidates of the competition, by implementing them for ASIC. Their results showed that, when comparing ciphers designed for use cases segregated as lightweight and defense-in-depth applications, there is a clearly observable 10x gap in the throughput/area efficiency, where the lightweight candidates are significantly both faster and smaller than there defense-in-depth counterparts. Thus, all lightweight cryptographic designs clearly demonstrate instances of conflict between security and all types of efficiencies like Space efficiency (SE) through small designs, time efficiency (TE) through high throughput rates and power efficiency (PE) through reduced power consumption.

12.3. Protocol Level

In almost all real world systems, crytograhic primitives are not implemented in a *standalone* mode, but are encapsulated in a larger cryptographic protocol along with other cryptographic primitives to achieve different security objectives. The TCP/IP (Transmission Control Protocol/Internet Protocol) stack is one of the most used communication protocols used in most of the computer networks around the world. It has a modular architecture with multiple layers, with each layer secured with different cryptographic protocols that are required to interact with each other to provide end-to-end security. Incorporating such sucurity measures at each layer is considered costly sometimes, but there are several trivial attacks like Packet Sniffing, Sppofing, Cache Poisoning, Proxy routing table updates, DoS style of attacks and many more that are possible if all the layers are not properly secured. But, there have been multiple other instances where application of certain optimization techniqeus have compromised the security of even a provably secure protocol, with the Transport Layer Security (TLS) protocol being the main focus of this section.

12.3.1. Data compression techniques used in TLS protocol

Transport Layer Security (TLS) (Previously known as Secure Sockets Layer (SSL)) is one of the most widely used cryptographic application in the world, which mainly provides security to the transport-layer of the TCP/IP stack. Data compression techniques were widely being utilized to decrease network traffic congestion, but this compression mechanism leaked information about the internal state of the data. This has been known to be exploited by a number of vulnerabilities like BREACH [320], CRIME [319] and TIME [366] attacks.

CRIME Attack (Type PRO.SE)

Both HTTP requests from the client and responses from the servers in cleartext are typically compressed by the TLS protocol using the DEFLATE[1] compression technique before they are encrypted to be sent over the insecure channel. Juliano Rizzo and Thai Duong [319] reported *Compression Ratio Info-leak Made Easy* (CRIME), a side channel attack that can retrieve information about session tokens and cookies. The attacker maliciously injects information into the victim's HTTP request and observes the size of the encrypted request. By adaptively altering the injected information depending on the observed sizes of the encrypted requests, an attacker can easily deduce information regarding some secret tokens embedded in the HTTP request.

[1] https://tools.ietf.org/html/rfc1950

TIME Attack (Type PR0.SE)

Following the CRIME attack, the major vendors deprecated the use of TLS compression technique at both the client and server sides which successfully thwarted the CRIME attack. Later, Tal Be'ery and Amichai Shulman reported *Timing Info-leak Made Easy* (TIME) attack [366], a variant of the CRIME attack but mainly targetting HTTP responses. The attacker carefully crafts additional information to be padded into the victim's HTTP requests and observes a larger RTT (Round Trip Time) for those manipulated requests in which the added information matches wih the internal data. Using the observable time difference due to compression, the attacker can retrieve internal information about the HTTP responses.

BREACH Attack (Type PR0.SE)

Gluck et al. [320] reported a variant of the CRIME attack called the *Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext* (BREACH) attack, which targets the size of the HTTP compressed responses (instead of TLS compressed requests as in CRIME) to reveal secret information about secret tokens and cookies in the body of the response.

12.3.2. Attacks on the encryption mode used in TLS protocol

There is another class of attacks that specifically target the CBC (Cipher Block Chaining) mode of encryption used in the context of the TLS protocol. Block ciphers are usually used in different modes to encrypt large amounts of data, ECB (Electronic Code Book) mode, Counter mode and CBC (Cipher Block Chaining) mode to name a few. The CBC mode for block ciphers is known for its efficient properties like reuse of ciphertext as initialization vector during encryption and its ability to decrypt data in parallel. The CBC mode in a standalone configuration is secure, but has caused a lot of security concerns when used in the TLS protocol [316–318].

BEAST Attack (Type PR0.TE)

TLS records are typically first authenticated using the HMAC construction, padded with deterministic data to align the data to the block size and then encrypted. Different error responses were invoked for the cases when the padding is correct but the HMAC was wrong or when the padding itself is wrong. The attacker tweaks the ciphertexts to evoke response regarding the correctness of the padding to reveal information about the plaintext. This attack which was first published by Vaudenay [316], which was later shown to be practical by Duong et al. [317] in 2011, famously known as the BEAST attack. The BEAST attack was made possible

due to a number of reasons, but one of the two main reasons were the differential error response on either incorrect padding or authentication and use of the last ciphertext of the previous packet as the IV of a new packet for want of time efficiency, with the attack very well aided by the structure of the CBC mode. The attack could be thwarted by evoking the same response for both incorrect padding and incorrect authentication. But, in doing so, the sender has to recalculate both the padding and the MAC, even in cases when packet has been correctly authenticated, increasing the computation times upon failure.

LUCKY13 Attack (Type PR0.TE)

Even on evoking the same response from the server upon failure due to different reasons to avoid information leakage, the attacker can still learn about the number of padded bytes based on the time taken for authentication. A padding error evokes a faster error response but the attacker can observe a slower response upon correct padding but incorrect authentication. Here again, the timing leakage is caused due to want of efficiency to avoid calculating the MAC even upon noticing an incorrect padding. Thus, fixing this requires the server to calculate MAC for both correctly and incorrectly padded messages. But, there still existed a timing leakage due to the difference in times for authenticating the data which gave information about the number of padded bytes. This attack which is very similar to the BEAST attack but uses a timing oracle was proposed by Nadhem AlFardan and Kenny Paterson [318], which is famously known as the LUCKY13 attack. These two attacks led to abandoning the use of CBC mode atleast in the context of TLS protocol as MAC-then-encrypt along with CBC has too many issues which could not be resolved very easily, while also looking like a source of many other hidden vulnerabilities.

12.4. System Level

The theoretically secure cryptographic primitives and the corresponding cryptographic protocols are ultimately required to be implemented on real world systems through which one can leverage upon their security guarantees. They are implemented in a wide array of real world systems, ranging from the smallest microcontrollers used in wireless sensor networks to the most powerful general purpose computers powering the data centres. System designers are always in pursuit of efficient yet secure implementations of cryptographic primitives and protocols as they are always considered to be adding a significant overhead in terms of efficiency to the application in hand. This pursuit of implementation efficiency has been constantly plagued with security issues as well. Following the trend observed in the higher layers of the cryptographic stack, we observe that this cross layer phe-

nomenon has made its presence felt in various aspects of system level security as well, which we cover in this section. We organize this section into three parts — (a) Hardware Security, (b) Software Security, and (c) Hardware/Software Interface Security.

12.4.1. Hardware Security

Hardware security as a discipline encompasses multiple fields dealing with cryptographic engineering and security such as hardware trojans, physical attacks, protection of the IC supply chain both pre-silicon and post-silicon, development of hardware root of trust and security enhanced hardware infrastructure. But following the line of work from the previous sections, we will focus on hardware security challenges from the stand point of efficient implementations of cryptographic primitives and protocols. Side channel attacks (especially power analysis) and fault attacks usually pose as a major threat towards secure cryptographic designs from the standpoint of hardware security, which will be the main focus of this section.

Generic Side channel protection approaches

Side-channel countermeasures against power analysis attacks have been developed on two different lines: leakage hiding and leakage randomisation.

Leakage Hiding Countermeasures — Leakage hiding aims at data independent processing which removes any side-channel basis. Dual-rail precharge logic (DPL) are a fair representative of this class of countermeasures [330]. DPL leads to over two times overhead both in area and time/performance. It suffers from two security vulnerabilities i.e. early propogation effect and routing imbalance [331]. Fixing any of these vulnerabilities leads to more elaborate design leading to area/performance overheads [332, 333] and thus compromising security with efficiency (Type SYS.TE and SYS.SE). Similarly, there are leakage countermeasures that work in the time domain that work by randomizing the occurence of the sensitive operations. But, Clavier *et al.* in [367] showed that desyncrhonizing the sensitive operations within a time window of t will only lead to a linear increase in the number of execution traces required by the attacker to perform the attack, while increasing t clearly hampers performance of the design (Type SYS.TE).

One of the most fundamental requirements for a side channel resistant design is to run in constant time so as to not leak information about data through side channels. But, constant time implementations are usually slower and time consuming to implement. For instance, the variable time operation of the base field operations in WolfSSL or OpenSSL implementations were exploited through timing attacks reported in [334]. Though not all timing variations are directly related to the

secret, which might seem to mitigate known side channel attacks, they have also been shown to open the door to new attacks [335]. These instances can be classified under the Type SYS.TE.

Leakage Randomisation Countermeasures — The alternate line of countermeasures is built upon leakage masking which is used to randomise the leakage of sensitive computations. We will consider private circuits [352], which form the basis of all masked implementations which assume an attacker with very strong probing capabilities. A t-order private circuit requires the attacker to probe $t+1$ signals to get intelligible inormation on $1-bit$. Since each bit is represented by $t-bits$ of the masked implementation, the area/performance overheads are evident. A practical study on implementation aspects of private circuits on FPGA was reported in [353] which reported a very large overhead of about 38× in the number of slices and 9× in performance as compared to the unprotected design. They also demonstrated that CAD tools in the bid to decrease CLB utilization and increase performance, optimize away the security measures employed in the private circuit design and thus require to provide explicit constraints in order to prevent insecure optimizations. Thus, these instances can be classified under the Type SYS.SE.

Similarly, building efficient algorithmic level masking countermeasures for asymmetric key cryptographic primitives based on ECC and RSA also remains an elusive task. With a plethora of known attacks with different capabilities [328], development of an efficient yet secure countermeasure to thwart all known (and future) attacks is a daunting task. For example, data re-randomisation countermeasure for secure ECC designs against the powerful single execution attacks yield almost a two times increase in computational time [329]. These instances fall under the Type SYS.TE.

Security Oblivious Power Management (Type SYS.PE)

Power, energy and thermal management has become a very crucial characteristic in today's modern commodity hardware, right from the ubiquitous embedded systems that are battery powered to the power hungry enterprise level systems. For instance, Dynamic Voltage Frequency Scaling (DVFS) is one of the most used techniques for energy management, wherein power consumption is regulated based on runtime task demands, by controlling the two crucial factors that majorly contribute to power consumption of the device - voltage and frequency. Tang et al. [354] demonstrated a remote fault attack on the ARM Trustzone CPU possible due to a fundamental flaw in the security oblivious DVFS technique. Pervasive software access was provided to the hardware registers which were used to control the voltage and frequency parameter of the device, that allowed an attacker to inject faults into the computation through remote software commands. This instance clearly demonstrates the need to employ efficiency improvements in secure

designs with extreme caution and also stresses the need for widespread security measures at every level of a secure design.

12.4.2. Software Security

For a long time, work on compiler optimizations have only focussed on ensuring functional coherence between the source code and its compiler optimized version [368, 369]. But there have been a number of works across literature that have revealed a very clear case of bumping into security errors where many a time compiler optimizations have lead to violating a security guarantee made by the original source code [324, 351, 370]. The formalization of this problem was first done by Silva *et al.* in [370] coining the term of *correctness-security* gap in compiler optimizations. This triggered a large body of work to study and diminish the effect of compiler optimizations on security [324, 371]. Silva *et al.* [370] point out to three types of vulnerabilities introduced by compiler optimizations — (a) Information Leakage through Persistent State, (b) Code Elimination through Undefined Behaviour, and (c) Side Channel Attacks.

Information Leakage through Persistent State (Type SYS.TE)

One of the most famous compiler optimizations that is known for its security flaws is the *dead store elimination* (DSE). A secret key residing in memory is usually overwritten with random data after use (or deleted), so as to avoid its persistence in memory. This scrubbed data is never read again by the program, so this is sensed by the compiler and thus the last instruction which accesses the memory location (i.e) the scrubbing instruction is optimized away thus leading to security issues. Though this issue has been known for quite sometime and has been claimed to be preventable through a variety of techniques [321–323], a recent work by Yang *et al.* [324] evaluated the known techniques used in various security projects and noticed that many of them are flawed. They propose a scrubbing-safe DSE optimization, but it still remains to be seen that DSE can truly be trusted to be secure. Another well known optimization is the *inline* function call, which eliminates overhead steps of explicitly calling a function. But it has the implication of merging of the stack frames of the caller and callee function. Thus, any secret variable used inside a function lives for a longer time as it now becomes a part of the callee function, which is a typical example of violating boundaries of trust-separated domains where a variable could trespass the boundary implemented by the programmer. Code motion is another widely adopted optimization technique, through which the compiler re-orders code by examining dependencies between the instructions. This might again lead to a situation of a persistence of a secret variable in memory for a longer duration that desired.

Code Elimination through Undefined Behaviour (Type SYS.SE)

Wang *et al.* [351] point out to a curious case of software bugs which they term as *Optimization Unstable* codes. These are code segments that can invoke undefined behaviour by the program, for eg. divide by zero, referencing a null pointer, shifting an n bit integer by more than n places etc. These type of codes are deemed to function erratically and thus the compilers more often than not take advantage of these code segments and optimize them away under the assumption that these undefined behaviours do not exist. Thus, any sanity checks like checking for an integer overflow or a null pointer reference will always evaluate to false and will be considered to be dead code by the compiler to be optimized away. These optimizations can sometime result in vulnerabilities based on buffer overflow or memory allocation. Wang *et al.* [351] also make a crucial observation of a general trend in compilers becoming more and more aggressive with successive generations in doing away with codes with undefined behaviour thus rising security concerns about their optimization characteristics.

Information Leakage through Side Channel (Type SYS.TE)

Compiler optimizations are also considered to be notorious in removing certain timing guarantees of the source code that were placed intentionally by the developer to ensure constant time implementations. Well known optimizations like strength reduction, Peephole optimizations, Common subexpression elimination etc. are techniques typically used by compilers for combination, simplification and replacement of certain parts of codes trying to achieve more performance and lesser resource utilization. But, usage of these optimizations more often than not, could lead to possible exploits through a number of side channel attack vectors. Time critical parts of the code are sometimes written in *inline* assembly to ensure that compilers do not perform any alteration leading to vulnerabilities. Similarly, there are countless instances littered across literature dealing with compiler optimizations that clearly demonstrate yet another clear case of trying to reach the ever elusive sweet-spot that remains hidden between the security and efficiency guarantees.

In complex applications like Internet of Things (IoT) and smart autonomous cars, developers are using many generic complex software stacks wherein most cases, the knowledge of the low level architechture behaviour is abstracted away. In such a scenario, implementation of cryptographic primitives encapsulated in several software layers can lead to unforeseen security vulnerabilities. Automated countermeasure insertion against side channel attacks at compilation time from a high level abstract language is a deep research topic. Though design using high level languages makes the code easier to write, read and verify for implementation bugs and flaws, most of the underlying levels are abstracted away. Both the underlying

micro-architechture and the compilation can introduce hidden information leakage [325–327]. The more the number of abstracted levels, lesser is the control over the actual behaviour of implementation, thus might lead to unforeseen security bugs and vulnerabilities.

12.4.3. Hardware/Software Interface Security

Apart from hardening devices against attacks purely exploiting vulnerabilities either in hardware or software, it is also important to know that there is also considerable leakage present at the interface between hardware and software, which is more commonly known as the *microarchitectural* level. The trillion fold increase in computational power over the last sixty years [372] can be attributed to a number of microarchitectural optimization strategies such as Cache memories, Pipelining, Branch Prediction, Multi-threading, out of order execution, virtualization etc. But, these optimizations also brought along with them hidden vulnerabilities that have been shown to be exploited by a number of attacks, which together can be bracketed under the term of *microarchitectural* attacks.

Cache Memory hierarchy - A necessary evil — The ever increasing gap between processor and memory speeds, is greatly attributed to the bisection of the semiconductor industry into two parts- Microprocessor and Memory [373]. While the microprocessor industry laid emphasis on increasing the speed of the processor, capacity had been the main driver for the memory industry. Cache memories were introduced in the 1960s in order to bridge this fast growing gap between processor and memory. Staring with a single level cache, architectures evolved to have upto 3 levels of caches - with the caches closer to the processor smaller and faster compared to the ones farther. There is an observable cache sharing hierarchy, wherein all cache levels except the Last Level Cache are local to the processor. This resource sharing which leads to an observable resource contention at multiple cache levels allows for visibility of activities of other co-located entities that contest for resources at the same level. This granularity in visibility also increases as one moves up from the Last Level Cache (LLC) to the first level cache which is the closest to the processor.

Caches were traditionally used to store instruction and data to increase system performance. But, there are also other smaller caches that are local to a processor core. Translational look-aside buffer, (TLB) which store page translation addresses used during page mapping and Branch Target Buffer (BTB) which store the branching addresses of upcoming branch instructions with the help of the Branch Prediction Unit (BPU) are a few examples of the same. In addition to speeding up information access from memory, the cache access times almost always depend on either the data value or the address or both. This differential behaviour of cache memory

access towards data has been exploited in a wide variety of attacks which together can be referred to as *Cache Timing Attacks*.

Achieving Parallelism through Resource Sharing (Type SYS.SE and SYS.TE) — Instrucion level parallelism is another key objective that had been the main focus of architecture designers to improve processor resource utilization and achieve execution of multiple instructions per clock cycle. Several architectural level design optimizations like symmetric multi-processing, hardware multi-threading, out-of-order execution and speculative execution were used to achieve the afore mentioned objective. This resource sharing resulted in observable resource contention at a very fine level at various execution units like ALUs, FPUs, memory controllers, system buses, interconnects etc. Thus, any entity like a parallel thread, process or a Virtual Machine will be able to observe the footprint of other similar co-located entities on contended resources at the same level. For eg. one can observe resource contention of execution units and L1 cache at a thread level or between processes or VMs running on the same core, while the resource contention between two entities located on different cores is only observable at lower levels like the Last Level Cache, system bus etc. Resource contention at multiple levels acrorss the processor hierarchy renders visibility of behaviour of other co-locatd entities mainly through timing-channels, which has lead to a number of microarchitectural attacks.

Reported Attacks

Gu et al. [336], in their survey of microarchitectural timing attacks broadly classify the same based on two axes - according to the *level of sharing* and the *degree of concurrency* required for the attack. While an attacker at the top level enjoys a very fine grained visibility of a co-located victim's behaviour, an attacker working at the bottom, at the bus level can only observe something close to the overall throughput variation. Similarly, an attacker at the thread level only requires to perform pre-emptive multitasking, while an attacker at the Last Level Cache requires true concurrency with the victim process to perform the attack.

Based on Attack styles

Exploitation of resource contention at caches have been achieved through different attack styles like PRIME+PROBE, FLUSH+RELOAD, EVICT+TIME, MELTDOWN and SPECTRE. These side channel attacks typically rely on timing leakage coming from resource contention observable across various levels of the cache hierarchy. Refer to Tab.12.3 for the description of the various reported styles of microarchitectural timing attacks.

There are other types of attacks that use cache as a covert channel to perform Denial of Service attacks [374] that can saturate the caches with the attacker's own

Class of Attack	Description
PRIME+PROBE	The attacker first *primes* the cache by filling it with its own lines in one or more sets. Once the victim finishes its execution, the attacker *probes* the previously loaded lines to observe any observable timing difference in the memory accesses.
FLUSH+RELOAD	An exact inverse of the PRIME+PROBE attack, wherein the attacker flushes the cache indexed by virtual addresses and lets the victim to execute. Once the victim has finished execution, the attacker reloads the same lines to check if the victim has accessed any of the same lines.
EVICT+TIME	A similar approach as that of the PRIME+PROBE attack, but first lets the victim run to observe the average run time. The attacker then evicts certain lines of interest and lets the victim run again to observe timing differences based on which some inference can be made on the victim's internal state.
MELTDOWN	Relies on execution of so called transient instructions (instructions which follow after a branch instruction or an exception) that are not meant to be executed by the victim. The out-of-order execution technique used to increase time efficiency is exploited in this kind of attack, whose activity can be observed in the shared caches across various levels. MELTDOWN results in a privilege escalation vulnerability specific to Intel processors through execution of instructions after an instruction trap.
SPECTRE	These attacks exploit the speculative execution technique used to predict direction of branching instructions, thus relying on execution of transient instructions. Though the results of these transient instructions are thrown away, their footprints are not erased from the shared caches across various levels. These attacks which can result read arbitrary memory from victim's process, apply to Intel, AMD and ARM processors.

Table 12.3: Different styles of microarchitectural timing attacks exploiting resource contention at the shared cache memories

data leading to serious performance degradation for the victim. These style of attacks have also been known to be performed on other types of caches like TLB [337] and BPU [339].

Based on level of sharing:

(Type SYS.SE and SYS.TE) Given the visibility of co-located entities rendered possible by resource sharing, attacks have been reported over multiple levels of the processor hierarchy. They are *Thread* Shared State, *Core* Shared State, *Package* Shared State, and *System* Shared State.

An attacker present at the thread level usually observes contention at thread shared resources like ALUs [338], BTB [339], FPU [340], BPU [341], return stack buffers [342] etc. Attacks at the *Core* shared state [87, 343, 344] typically target activity in the L1 and L2 level caches due to contention among different threads and processes running on the same core. Attacks at the *Package* shared state [345–347] target activity in the Last Level Cache which is typically shared by multiple processors on the same core. These attacks typically have to work with lesser granularity in observing the victim's behaviour and require more concurrency with the victim entity residing on the other core. Contention on system level resources like System buses, processor interconnects and system interfaces like PCIExpress have also been shown to be exploitable by a number of covert channel, side channel and DOS style attacks [348–350].

With the above cited literature on microarchitectural attacks across the entire processor hierarchy, we can clearly see that major architectural optimization techniques have been employed without foreseeing the possible security threats. These instances observable at the interface between hardware and software again stand as evidence of the eternal conflict between security and efficiency.

12.5. Proposal for a Security Aware Design Flow

In Sections 12.2,12.3,12.4, we surveyed as many references as possible available in literature from academia and the industry to provide evidence of the ever existing trade-off between security and efficiency across multiple layers of the cryptography stack. We could see that certain optimization strategies opened gates to some unforseeable security vulnerabilities, or protecting against powerful attacks becomes a very costly affair from a designer's perspective, who always targets high performance and low resource utilization. Since vulnerabilities can be introduced at any level, a security engineer's job to ensure security at each and every level becomes a paramount task.

No longer can security be considered as an afterthought for a digital system design.

Incorporation of security to existing digital systems using an ad-hoc approach has only lead to a number of attacks as seen in previous sections [317, 319, 320, 366]. Also some of the optimization techniques like cache memory hirarchy, out of order execution, speculative execution, branch prediction etc. were introduced long before security was being seriously considered as being a threat to digital computer systems. With an ever increasing number of types of attacks and the attack surfaces, it is now supremely important to integrate the notion of security into each and every level of the design flow.

We would like to propose a fully security aware design flow that would be useful for a security engineer who is required to incorporate all the required and necessary security measures and functionalities for any given application. Refer Fig.12.1 for our proposal for a security aware design flow, which tries to incorporate security at each and every step of the design flow for both hardware and software implementations.

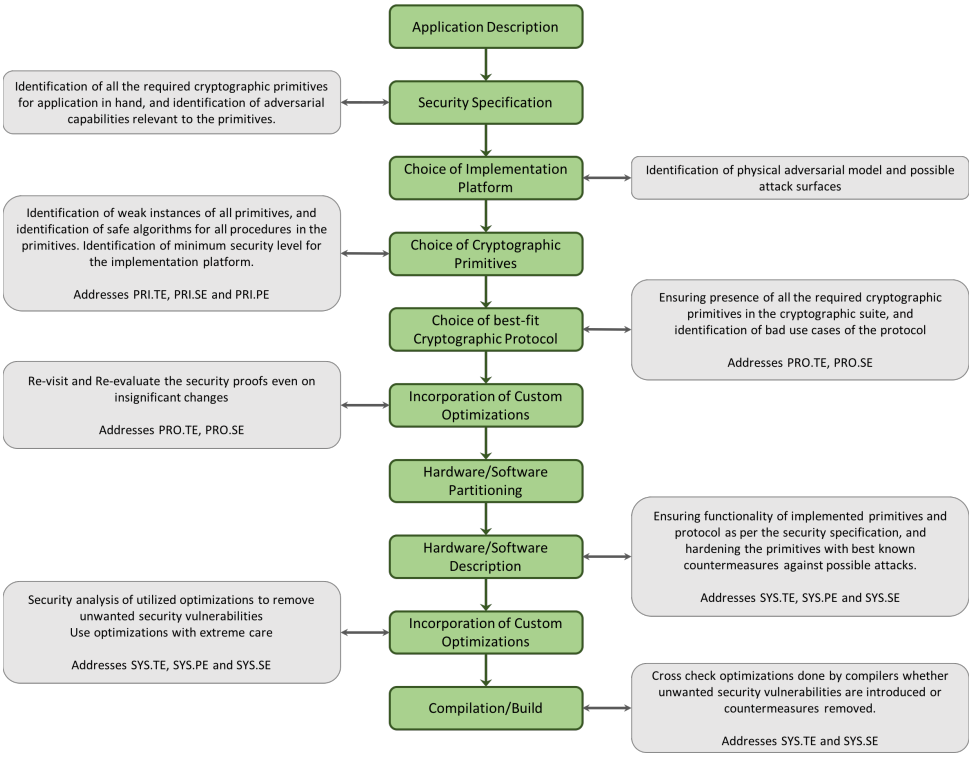


Figure 12.1: Our proposal for a Security Aware Design Flow

12.6. Conclusion

In this position paper, we have shown numerous examples about how security and efficiency stands in sharp contrast with each other. This is a fact that is not yet well understood in the design community, leading to regular and severe security breaches. The vulnerabilities presented in this paper shows security issues across multiple design layers and due to the pursuit to achieve different performance objectives, e.g., space, time and power. Consequently, we advocate a security-aware design flow that includes security as an architectural design constraint. This work calls for the development of an early design space exploration tool that includes security as a quantifiable metric. Further interesting directions could be to study the interplay between security wrappers in different design layers.

VI

Application of design guidelines: Lightweight ciphers secure implementations

In this Part, we focus on implementing the precepts outlined in the previous chapter in a real-world scenario. Specifically, we apply our security-aware design flow to the selection and implementation of a symmetric encryption algorithm for mass-market IoT security (Security of Private-Key Primitives (Type PRI.SE PRI.TE PRI.PE). in the previous chapter). When it comes to mass-market applications that require secure firmware updates, general-purpose microcontrollers are the preferred platform. However, in such applications, side-channel attacks can be a major concern.

To mitigate timing attacks, a constant-time implementation is necessary. One efficient way to achieve a constant-time implementation of a block cipher on microcontrollers is to use the bitslicing technique, which also happens to be masking-friendly. We introduce a new technique called "fix-slicing," which allows for the development of constant-time, efficient implementations. This technique can be applied to other ciphers beyond the one chosen for demonstration (GIFT). Note that in this study, fault injection resistance is not considered.

13

Fixslicing: A New GIFT Representation

The GIFT *family of lightweight block ciphers, published at CHES 2017, offers excellent hardware performance figures and has been used, in full or in part, in several candidates of the ongoing NIST lightweight cryptography competition. However, implementation of* GIFT *in software seems complex and not efficient due to the bit permutation composing its linear layer (a feature shared with* PRESENT *cipher).*

In this article, we exhibit a new non-trivial representation of the GIFT *family of block ciphers over several rounds. This new representation, that we call fixslicing, allows extremely efficient software bitsliced implementations of* GIFT*, using only a few rotations, surprisingly placing* GIFT *as a very efficient candidate on microcontrollers. Our constant time implementations show that, on ARM Cortex-M3, 128-bit data can be ciphered with only about 800 cycles for* GIFT-64 *and about 1300 cycles for* GIFT-128 *(assuming pre-computed round keys). In particular, this is much faster than the impressive* PRESENT *implementation published at CHES 2017 that requires 2116 cycles in the same setting, or the current best* AES *constant time implementation reported that requires 1617 cycles. This work impacts* GIFT*, but also improves software implementations of all other cryptographic primitives directly based on it or strongly related to it.*

13.1. Introduction

In parallel to the rise of pervasive computing and IoT, lightweight cryptography has naturally been a very hot topic in the past decade. Many new primitives have been proposed, from block ciphers to hash functions and authenticated encryption schemes, for various goals such as minimization of area, energy or power consumption, latency, etc. One can remark that there is no single algorithm that is more efficient than all others on every possible platform. Even though designers try to produce a primitive aiming at a particular class of platforms while maintaining good performance otherwise, we can generally observe that hardware-oriented ciphers tend to be less efficient on software and vice-versa. For example, the NSA did not propose only a single lightweight block cipher, but two of them [375]: one oriented for constrained hardware platforms (SIMON) and one oriented for constrained software platforms (SPECK).

In hardware, it seems that the community is reaching a limit in terms of performances, with recent schemes [375–377] that can be implemented efficiently using a very small data-path (minimizing area and power), while allowing also efficient trade-offs for fast and low-energy implementations. Yet, constrained software platforms such as small micro-controllers will play a very important role in the future. Even though hardware-oriented designs use a very small total number of bitwise operations when compared to classical designs such as AES, their situation in software is not so bright: many of these ciphers use hardware-friendly diffusion layers and an important number of cycles will be required to move these bits around, without much possibility to benefit from vectorization. This is especially true for ciphers using bit permutation such as PRESENT [283] or GIFT [377]. Since this bit permutation is basically free in hardware (it consists of simple wirings), designers concentrated on how to maximize security when choosing this permutation layer. For example, GIFT permutation layer has been chosen with security as the only criterion (more precisely, maximizing its resistance against differential and linear attacks).

When high parallelism can be achieved in the operating mode where the primitive will be placed, one can always use highly bitsliced implementations (see performances of SIMON, SKINNY and GIFT on recent Intel processors with AVX2 instructions [377]) that can lead to excellent performance: these ciphers again use a very small number of bitwise operations and the high parallelism will allow to strongly reduce the cycles wasted in moving bits around by unrolling the implementation. However, this strategy will not be applicable in the case of constrained micro-controllers, as these devices will not offer enough registers to perform such highly bitsliced implementations efficiently. These highly bitsliced implementations will also not be possible for serial operating modes, which are quite widespread in prac-

tice and are even more relevant for lightweight cryptography as it can save some area.

It remains rather unexplored how efficient hardware-oriented ciphers can be in software. Yet, this topic is quite important with the ongoing NIST LightWeight Cryptography (NIST LWC) competition, that started in 2018, with the goal of selecting the future authenticated encryption standard(s) for constrained environments. A first answer was given at CHES 2017, with a new very efficient implementation of PRESENT cipher on various micro-controllers [378]. It is based on a decomposition of the permutation layer over two consecutive rounds, resulting in a more software-friendly representation.

However, PRESENT has a rather low security margin with regards to linear cryptanalysis and its advanced extensions. It also has the disadvantage to only come in a 64-bit block version, which is to be avoided [379] unless a Beyond-Birthday-Bound (BBB) operating mode can be used (generally much more costly). Actually, one can observe that none of the NIST LWC candidates use PRESENT as internal primitive, even though it is widely considered as one of the first lightweight ciphers. Recently, at CHES 2017, the GIFT family of block ciphers was proposed to correct these two issues with PRESENT. GIFT has a 128-bit version and provides a much stronger resistance against linear cryptanalysis than PRESENT, thanks to a careful choice of its S-box, its diffusion layer and how they operate together. It has actually been used as a basic block for several NIST LWC candidates, such as ESTATE [380], GIFT-COFB [381], HYENA [382], LOTUS-AED [383], LOCUS-AED [383], SIMPLE [384], SUNDAE-GIFT [385] and TGIF [386]. The problem is that software performance of GIFT is believed to be poor on micro-controllers, because even using table-based implementations, moving the bits around for the diffusion layer will cost many expensive rotations, shifts, masks, exclusive-ORs, etc. To the best of our knowledge, no micro-controller implementation has been previously reported for GIFT.

Our Contributions. In this article, we propose a new non-trivial representation of both versions of the GIFT cipher over several rounds. More precisely, we show how the seemingly-complex bit permutation of GIFT-64 can be rewritten over 4 consecutive rounds, using only a few simple operations. This new very clean representation, that we named *fixslicing*, allows an efficient bitsliced implementation of GIFT-64 on ARM Cortex-M3, requiring only about 800 cycles to cipher two 64-bit input blocks. Our setting assumes that round keys are precomputed, but we also provide an efficient implementation of the GIFT-64 key schedule.

The situation is more difficult for GIFT-128, as its bit permutation operates on twice as many bits. Yet, a more systematic search approach led to a new representation of GIFT-128 over 5 rounds, again using only a few simple operations. This new

very clean representation allows an efficient bitsliced implementation of GIFT-128 on ARM Cortex-M3, requiring only about 1300 cycles to encrypt one single 128-bit input block.

Our implementations show that GIFT is very efficient in software, as they are much faster than the impressive PRESENT implementation published at CHES 2017 that requires 2116 cycles in the same setting, or the current best AES constant time implementation reported that requires 1617 cycles. This work impacts GIFT, but also all other cryptographic primitives directly based on it or strongly related to it. In particular, we benchmarked that GIFT-COFB runs at 79 cycles per byte on ARM Cortex-M3 for long messages, placing this scheme as a very fast candidate.

13.2. The GIFT family of block ciphers

In this section, we will describe the GIFT family of block ciphers but refer to [377] for the full specifications.

GIFT is a family of lightweight block ciphers, with two members: GIFT-64 and GIFT-128 which have a block size of 64 and 128 bits respectively. They are composed of a Substitution-Permutation Network (SPN) with a key length of 128-bit. They are 28-round and 40-round iterative block ciphers respectively, with identical round function.

There are different ways to perceive GIFT-64 and GIFT-128. The classical one is to represent it with an SPN view (see Section 2 of [377] for a graphical representation), which looks like a PRESENT-like cipher with 16 (or 32) 4-bit S-boxes and a 64-bit (or 128-bit) bit permutation (see Figure 13.1 that illustrates 2 rounds of GIFT-64). Since we will be proposing new bitsliced implementations, we will be using the bitsliced description instead, which is similar to Appendix A of the GIFT paper.

Instead of collecting the input stream S-box per S-box, we can also consider that the data arrives already in bitsliced ordering. This changes absolutely nothing to the quality of the cipher, as a bit permutation is simply applied at the start (plaintext) and at the end (ciphertext) of the encryption process (to compensate for the state bitslice packing/unpacking). We note that such ciphers have been already used in some NIST LWC candidates such as GIFT-COFB [381]. We will denote GIFTb-64 and GIFTb-128 the bitsliced-input versions of GIFT-64 and GIFT-128 respectively.

13.2.1. Round function

Each round of GIFT-64 (or GIFT-128) consists of 3 steps: SubCells, PermBits, and AddRoundKey.

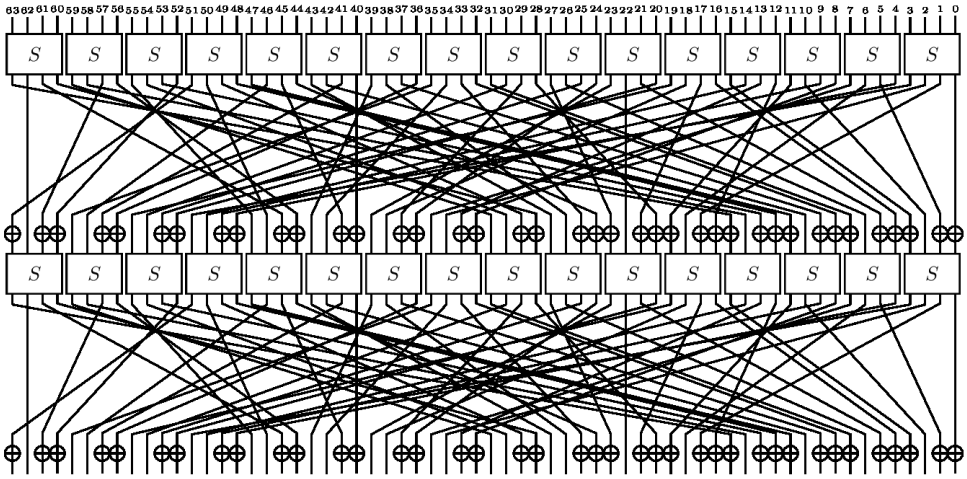


Figure 13.1: 2 rounds of GIFT-64 (from https://www.iacr.org/authors/tikz/).

Initialization. The 64-bit (or 128-bit) plaintext is loaded into the cipher state S which will be expressed as 4 16-bit (or 32-bit) segments. In the perspective of a 2-dimensional array, the bit ordering is from top-down, then right to left. Namely, for GIFT-64, we have:

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} b_{60} & \cdots & b_8 & b_4 & b_0 \\ b_{61} & \cdots & b_9 & b_5 & b_1 \\ b_{62} & \cdots & b_{10} & b_6 & b_2 \\ b_{63} & \cdots & b_{11} & b_7 & b_3 \end{bmatrix}.$$

while for GIFT-128 we have:

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} b_{124} & \cdots & b_8 & b_4 & b_0 \\ b_{125} & \cdots & b_9 & b_5 & b_1 \\ b_{126} & \cdots & b_{10} & b_6 & b_2 \\ b_{127} & \cdots & b_{11} & b_7 & b_3 \end{bmatrix}.$$

The 128-bit secret key is loaded into the key state KS partitioned into 8 16-bit words. In the perspective of a 2-dimensional array, the bit order-

ing is from right to left, then bottom-up.

$$KS = \begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} b_{127} & \cdots & b_{112} & \| & b_{111} & \cdots & b_{98} & b_{97} & b_{96} \\ b_{95} & \cdots & b_{80} & \| & b_{79} & \cdots & b_{66} & b_{65} & b_{64} \\ b_{63} & \cdots & b_{48} & \| & b_{47} & \cdots & b_{34} & b_{33} & b_{32} \\ b_{31} & \cdots & b_{16} & \| & b_{15} & \cdots & b_2 & b_1 & b_0 \end{bmatrix}$$

SubCells. The substitution layer of 16 (or 32) identical 4-bit S-boxes can be applied in parallel with the following operations.

$$\begin{aligned} S_1 &\leftarrow S_1 \oplus (S_0 \ \wedge \ S_2) \\ S_0 &\leftarrow S_0 \oplus (S_1 \ \wedge \ S_3) \\ S_2 &\leftarrow S_2 \oplus (S_0 \ \vee \ S_1) \\ S_3 &\leftarrow S_3 \oplus S_2 \\ S_1 &\leftarrow S_1 \oplus S_3 \\ S_3 &\leftarrow \ \neg \ S_3 \\ S_2 &\leftarrow S_2 \oplus (S_0 \ \wedge \ S_1) \\ \{S_0, S_1, S_2, S_3\} &\leftarrow \{S_3, S_1, S_2, S_0\}, \end{aligned}$$

where $\wedge$, $\vee$ and $\neg$ are logical AND, OR and NOT operation respectively.

PermBits. The bit permutation of GIFT has the special property that each bit located in a slice i remains in the same slice through this permutation. Now, different 16-bit (or 32-bit) permutations are applied to each S_i independently. They map a bit located at position j in slice i to position $P_i(j)$ in the same slice i. We provide in Tables 13.1 and 13.2 the $P_i(j)$ values for GIFT-64 and GIFT-128 respectively.

Table 13.1: Specifications of GIFT-64 bit permutation.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_0(j)$	0	12	8	4	1	13	9	5	2	14	10	6	3	15	11	7
$P_1(j)$	4	0	12	8	5	1	13	9	6	2	14	10	7	3	15	11
$P_2(j)$	8	4	0	12	9	5	1	13	10	6	2	14	11	7	3	15
$P_3(j)$	12	8	4	0	13	9	5	1	14	10	6	2	15	11	7	3

AddRoundKey. This step consists of adding the round key and round constant. Two 16-bit (or 32-bit) segments U, V are extracted from the key state as the round key: $RK = U\|V$. Then, for the addition of round key, U and V are XORed to S_1 and S_0 of the cipher state respectively for

Table 13.2: Specifications of GIFT-128 bit permutation.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_0(j)$	0	24	16	8	1	25	17	9	2	26	18	10	3	27	19	11
$P_1(j)$	8	0	24	16	9	1	25	17	10	2	26	18	11	3	27	19
$P_2(j)$	16	8	0	24	17	9	1	25	18	10	2	26	19	11	3	27
$P_3(j)$	24	16	8	0	25	17	9	1	26	18	10	2	27	19	11	3

j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_0(j)$	4	28	20	12	5	29	21	13	6	30	22	14	7	31	23	15
$P_1(j)$	12	4	28	20	13	5	29	21	14	6	30	22	15	7	31	23
$P_2(j)$	20	12	4	28	21	13	5	29	22	14	6	30	23	15	7	31
$P_3(j)$	28	20	12	4	29	21	13	5	30	22	14	6	31	23	15	7

GIFT-64, or S_2 and S_1 of the cipher state respectively for GIFT-128:

$$S_1 \leftarrow S_1 \oplus U, \quad S_0 \leftarrow S_0 \oplus V \qquad \text{for GIFT-64}$$
$$S_2 \leftarrow S_2 \oplus U, \quad S_1 \leftarrow S_1 \oplus V \quad \text{for GIFT-128.}$$

For the addition of round constant, S_3 is updated as follows:

$$S_3 \leftarrow S_3 \oplus 0\text{x}80\text{XY} \qquad \text{for GIFT-64}$$
$$S_3 \leftarrow S_3 \oplus 0\text{x}800000\text{XY} \quad \text{for GIFT-128}$$

where the byte XY = $00c_5c_4c_3c_2c_1c_0$.

13.2.2. Key schedule and round constants

The key schedule and round constants are the same for both versions of GIFT, the only difference is the round key extraction. A round key is *first* extracted from the key state before the key state update. For GIFT-64, two 16-bit words of the key state are extracted as the round key $RK = U\|V$

$$U \leftarrow W_6,\ V \leftarrow W_7,$$

while for GIFT-128, four 16-bit words of the key state are extracted as the round key $RK = U\|V$.

$$U \leftarrow W_2\|W_3,\ V \leftarrow W_6\|W_7.$$

The key state is then updated as follows,

$$\begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} W_6 \ggg 2 & \| & W_7 \ggg 12 \\ W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \end{bmatrix},$$

where $\ggg i$ is an i bits right rotation within the 16-bit word.

The round constants are generated using a 6-bit affine LFSR, whose state is denoted as $c_5c_4c_3c_2c_1c_0$. Its update function is defined as:

$$c_5\|c_4\|c_3\|c_2\|c_1\|c_0 \leftarrow c_4\|c_3\|c_2\|c_1\|c_0\|c_5 \oplus c_4 \oplus 1\,.$$

The six bits are initialized to zero, and updated *before* being used in a given round. The values of the constants for each round are given in the table below, encoded to byte values for each round, with c_0 being the least significant bit.

Rounds	Constants
1 - 16	01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E
17 - 32	1D,3A,35,2B,16,2C,18,30,21,02,05,0B,17,2E,1C,38
33 - 48	31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04

13.3. Naive bitsliced implementation of GIFT

Naive bitsliced implementations of the GIFT family of block ciphers can be achieved by following straightforwardly the specifications. First, in the case of GIFT-64 and GIFT-128, one has to rearrange the inputs in their bitsliced representation. This can be done using the SWAPMOVE technique [387]:

$$\begin{aligned} &\texttt{SWAPMOVE}(A,B,M,n):\\ &\qquad T = (B \oplus (A \gg n)) \wedge M\\ &\qquad B = B \oplus T\\ &\qquad A = A \oplus (T \ll n) \end{aligned}$$

which consists in swapping the bits in B masked by M with the bits in A masked by $(M \ll n)$. Regarding the substitution layer, the 4-bit S-boxes can be computed in parallel in only 13 operations as described in Section 13.2. The main difficulty lies in the diffusion layer as it refers to the least bitslice-friendly operation. For the sake of clarity, let us consider the case of GIFT-64. In order to apply the 16-bit permutation P_0 to S_0, a basic approach would be to move the bits using masks and

shifts, resulting in the following operations:

$$\begin{aligned} P_0(S_0) = (S_0 \wedge 0\text{x}0401) \quad & \vee ((S_0 \wedge 0\text{x}0008) \ll 1) \quad \vee \\ ((S_0 \wedge 0\text{x}2000) \ll 2) \quad & \vee ((S_0 \wedge 0\text{x}0040) \ll 3) \quad \vee \\ ((S_0 \wedge 0\text{x}0200) \ll 5) \quad & \vee ((S_0 \wedge 0\text{x}0004) \ll 6) \quad \vee \\ ((S_0 \wedge 0\text{x}0020) \ll 8) \quad & \vee ((S_0 \wedge 0\text{x}0002) \ll 11) \quad \vee \\ ((S_0 \wedge 0\text{x}1000) \gg 9) \quad & \vee ((S_0 \wedge 0\text{x}8000) \gg 8) \quad \vee \\ ((S_0 \wedge 0\text{x}0100) \gg 6) \quad & \vee ((S_0 \wedge 0\text{x}0800) \gg 5) \quad \vee \\ ((S_0 \wedge 0\text{x}4010) \gg 3) \quad & \vee ((S_0 \wedge 0\text{x}0080) \gg 2) \end{aligned}$$

which requires about 27 cycles on ARM Cortex-M processors. In the same way, P_1, P_2 and P_3 can be implemented in approximately 14, 27 and 18 cycles, respectively. Therefore, the diffusion layer requires about 100 cycles for a single round. This highlights why ciphers using bit permutation are generally considered inappropriate for software implementations on micro-controllers.

Still, it is possible to minimize the impact on performances by operating on several blocks in parallel for 32-bit (and above) architectures. In order to give some insights on how GIFT performs on ARM Cortex-M3 and M4 using the naive bitsliced implementation, we benchmarked a code fully written in C language, compiled by `arm-none-eabi-gcc 9.2.1` using the flag `-O3` for optimized speed results, on the STM32L100C and STM32F407VG development boards. Note that our benchmark simply measures the execution time to expand the key and to encrypt 128-bit data, without any operating mode. Implementation results are listed in Table 13.3. For encryption functions, the data in ROM refers to precomputed round constants while under RAM usage, I/O refers to the amount of memory needed to store the input and ouput plus the temporary variables (excluding the round keys).

Algorithm	**Parallel**	**Speed** (cycles/block)		**ROM** (bytes)		**RAM** (bytes)	
	Blocks	M3	M4	Code	Data	I/O	Stack
GIFT-64 key exp.	-	2 296	2 304	668	0	112	24
GIFTb-64 encryption	2	2 091	2 097	1 172	28	52	48
GIFT-64 encryption	2	2 141	2 138	1 608	28	52	48
GIFT-128 key exp.	-	3 433	3 476	644	0	360	24
GIFTb-128 encryption	1	8 456	8 375	1 508	40	52	48
GIFT-128 encryption	1	8 644	8 573	1 996	40	52	48

Table 13.3: Naive bitsliced implementation results on ARM Cortex-M3 and M4 for various versions of GIFT.

As expected, the result is that GIFT is not well suited for software bitsliced implementations on micro-controllers. While our C implementation requires about 4000 cycles to encrypt 128-bit data using GIFT-64, twice as much are required when using GIFT-128. This gap is due to the fact that, on top of having more rounds than GIFT-64, the slice permutations $P_0, \cdots, P_3$ of GIFT-128 operate on 32 bits instead of 16, increasing the number of masks and shifts to compute. However, the next section introduces a new GIFT representation which challenges this conclusion.

13.4. A new GIFT representation

13.4.1. GIFT-64

Let us consider a bitsliced representation of the cipher state: for each nibble, bit 0 is placed in the slice 0, bit 1 in slice 1, bit 2 in slice 2 and bit 3 in slice 3. For ease of description, a slice can be placed in matrix form, as shown in the top row of Figure 13.3. During the SubCells application, when each slice is stored in independent words, all the 16 S-boxes are implemented in parallel in bitslice manner, as seen in Figure 13.2. Then, according to the GIFT designers [377], the bit permutation can be implemented as follows:

- Take the transpose of each individual slice matrix
- Apply the following row swaps:
  - ◇ Slice 0 matrix: swap row 1 with 3
  - ◇ Slice 1 matrix: swap row 0 with 1, and swap row 2 with 3
  - ◇ Slice 2 matrix: swap row 0 with 2
  - ◇ Slice 3 matrix: swap row 0 with 3, and swap row 1 with 2

We give a graphical representation of 4 rounds of this process in Figure 13.3.

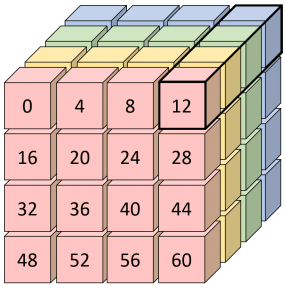


Figure 13.2: Cubic representation of the main state of GIFT-64. Each color refer to a slice matrix while the black cuboid is where an S-box is applied.

As explained in Section 13.3, the diffusion layer requires bits to be moved around individually in the slice (and not entire chunks of the slice), resulting in a significant overhead. In order to avoid these issues, we propose a new way to represent GIFT-64. The idea is to fix the first slice matrix to never move and find the easiest operations that could keep the bits of other slice matrices synchronised after application of the linear layer (so that the S-box computation that comes after will indeed involve the proper bits). This representation is given in Figure 13.4 and one can see that even though the bit positions are different, each S-box will have exactly the same bits indexes involved when compared to the classical representation given in Figure 13.3. For example, after one round, the classical representation will have bits 16/21/26/31 in row 0 and column 1 and we can see that the exact same quartet will appear as well in the new representation, but in row 1 and column 0 instead. The fact that this quartet appears in a different row/column has no impact on the actual computation of the S-box right after, since the computation is bitsliced.

The very nice property of this new representation is that it requires very few operations: each round, we only apply a row or column rotation to the three last slice matrices, while the first slice matrix is never moved. More precisely, for a round i:

- if i%4=0, rotate slice j matrix by j columns to the left
- if i%4=1, rotate slice j matrix by j rows to the top
- if i%4=2, rotate slice j matrix by j columns to the right
- if i%4=3, rotate slice j matrix by j rows to the bottom

This entire process, which applies different functions for each 4 consecutive rounds, will be much less costly in software than having to transpose and then swap rows around. Even better: the new and the classical representations are naturally fully synchronised again after applying these 4 rounds, which avoids any representation correction to be applied at the end of the cipher (since GIFT-64 has 28 rounds, which is a multiple of 4). This is due to the fact that $P_i^4 = Id$ for all i. Therefore, no matter which slice matrix is fixed, the new and the classical representations will be fully synchronised after 4 rounds anyway.

We call this technique *fixslicing*. Note that it is close to the software optimization of PRESENT in [378] which consists in decomposing the permutation over 2 rounds, as our new representation can be seen as a decomposition of $P_0, \cdots, P_3$ over 4 rounds. Actually, the fixslicing technique is a particular case for permutations which ensures that, from a bitsliced perspective, all bits within a slice remain in the same one through the permutation. Therefore, it can be applied to all permutations that verify this property, and the number of rounds to consider for the decomposition equals $min(\text{order}(P_i))$ for all i.

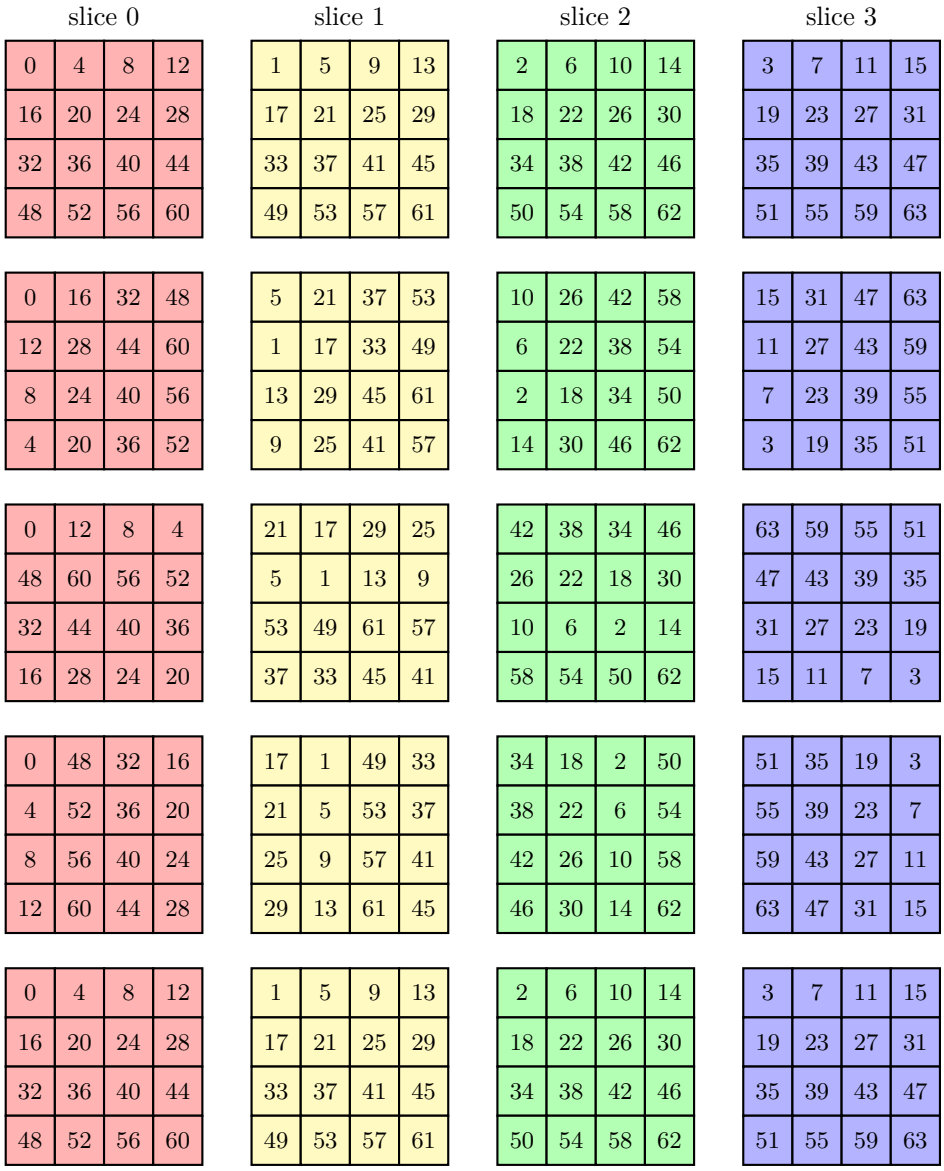


Figure 13.3: Classical representation of the GIFT-64 round function during 4 rounds. Each cell represents a bit, and the numbers in the cells then denote the actual index of that particular bit in the state. Slice 0 (resp. 1/2/3) depicted in red (resp. yellow/green/blue) represents all the bits at position 0 (resp. 1/2/3) of the S-boxes of the cipher state.

The other side of the coin of this new representation is that the round keys and round constants have to be adapted to fit the new way the bits are positioned. While this is not an issue for the round constants by using a precomputed look-up table, adapting the key schedule might result in some computational overhead. The naive approach would be to run the key schedule using the classical representation, before rearranging bits for all round keys. However, one can take advantage of the fact that after 4 rounds all key words are back in the same position within the key state (yet the words themselves will be rotated because of the rotation operations in the key schedule). In other terms, because $RK^i = U^i \| V^i$ and $RK^{i+4} = U^i \ggg 2 \| V^i \ggg 12$, each key word has to go through the same bit reordering every 4 rounds. Therefore a more efficient approach is to rearrange bits for the

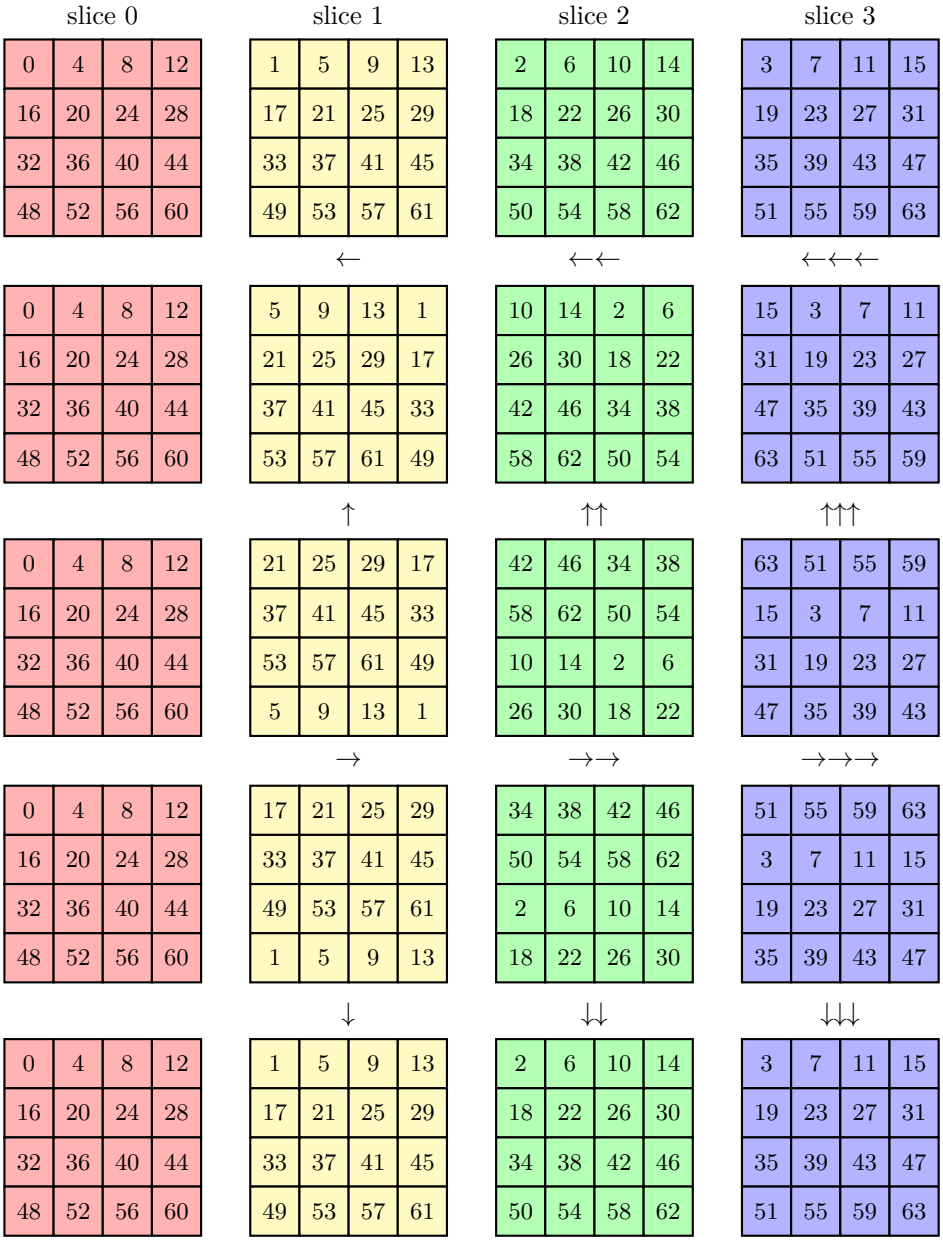


Figure 13.4: New representation of the GIFT-64 round function during 4 rounds. Each cell represents a bit, and the numbers in the cells then denote the actual index of that particular bit in the state. Slice 0 (resp. 1/2/3) depicted in red (resp. yellow/green/blue) represents all the bits at position 0 (resp. 1/2/3) of the S-boxes of the cipher state.

first 4 round keys only, and to adapt the key schedule accordingly. More details on how to compute the key schedule in the fixsliced representation are given in Appendix A.

13.4.2. GIFT-128

As for GIFT-64, we consider a bitsliced representation of the cipher state. For ease of description, a slice i can be represented as a pair of matrices i_L and i_R, as shown

in the top row of Figure 13.6. During the SubCells application, when each slice is stored in independent words, all the 32 S-boxes are implemented in parallel in a bitsliced manner, as seen in Figure 13.5.

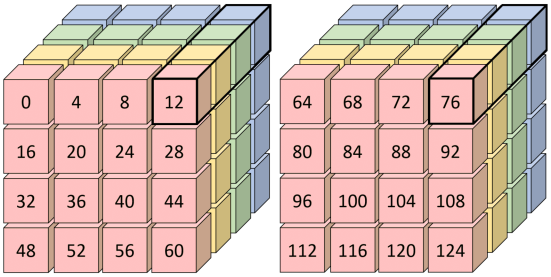


Figure 13.5: Cubic representation of the main state of GIFT-128. The black cuboid is where an S-box is applied for both matrices.

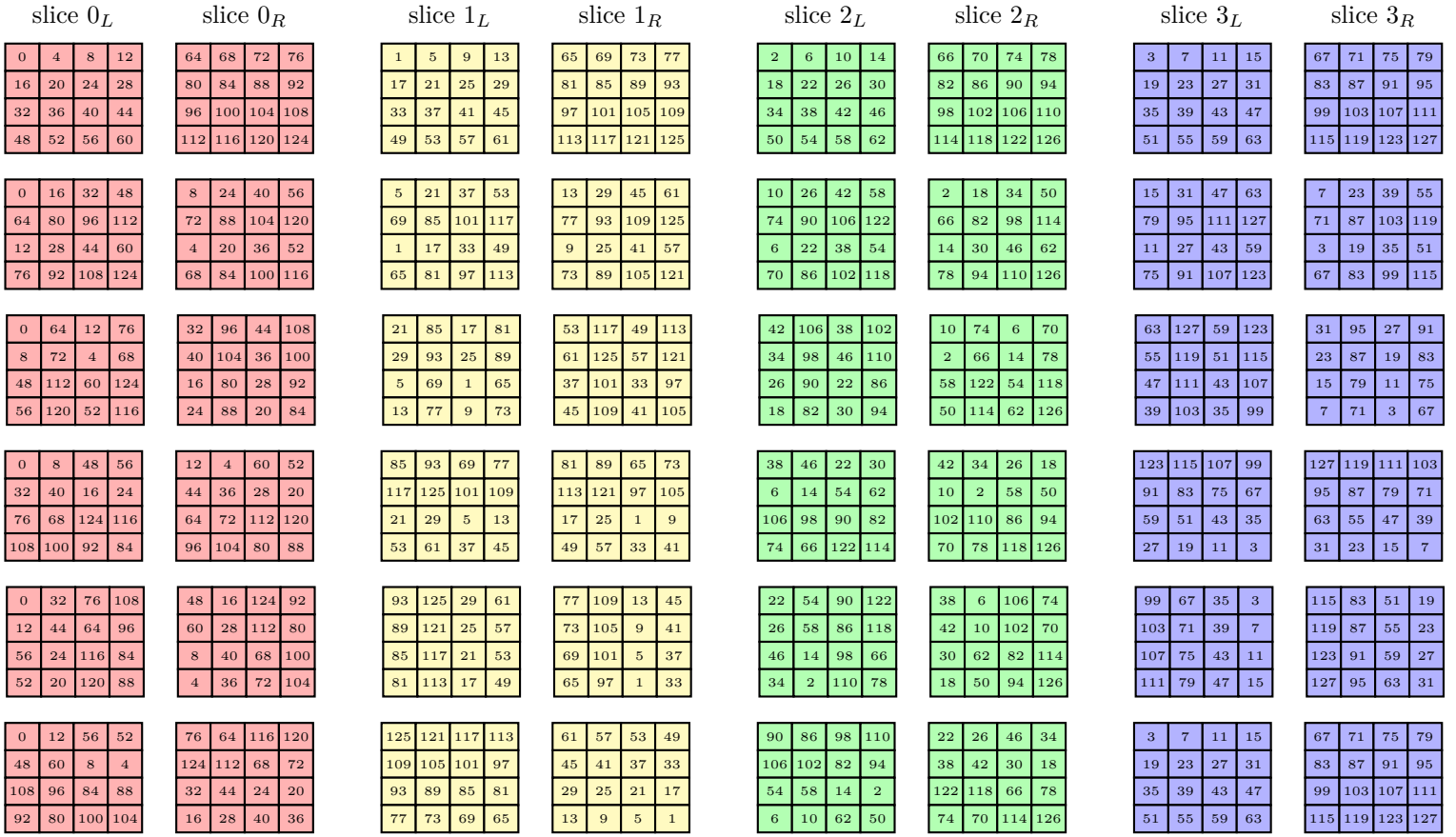
Then, according to the GIFT designers [377], the bit permutation can be implemented as follows:

- Take the transpose of each individual slice matrix
- Shuffle the left and right matrices of each slice (i.e. shuffle i_L and i_R for all i).
- Apply the following row swaps:
  - Slice 0: swap the 2 bottom halves
  - Slice 1: swap the top and bottom halves of the slices independently
  - Slice 2: swap the 2 top halves
  - Slice 3: cross swap the top and bottom halves

We give a graphical representation of 5 rounds of this process in Figure 13.6.

As for GIFT-64, one can see that the process will be very costly in software, with lots of transpositions, shuffle and swaps. We therefore propose a new way to represent GIFT-128, thanks to the fixslicing technique. However, unlike GIFT-64, note that the classical and the new representation will not be synchronised anymore after 4 rounds since $P_i^4 \neq Id$ for all i. For GIFT-128 we have $P_0^{31} = P_1^{10} = P_2^{31} = P_3^5 = Id$. In other terms, by fixing the fourth slice to never move, we can define a routine so that the classical and new representation are naturally synchronised after 5 rounds. Since GIFT-128 has 40 rounds (which is a multiple of 5), it avoids any correction to be applied at the end of the cipher. This representation is depicted in Figure 13.7. Additional illustrations are also provided in Appendix .10.

One can again see that even though the bit positions are different, each S-box will have exactly the same bit indexes involved when compared to the classical repre-



GIFT-128 round function during 5 rounds. Each cell represents a bit, and the numbers in the cells then

Figure 13.6: Classical representation of the GIFT-128 round function during 5 rounds. Each cell represents a bit, and the numbers in the cells then denote the actual index of that particular bit in the state. Slice 0 (resp. 1/2/3) depicted in red (resp. yellow/green/blue) represents all the bits at position 0 (resp. 1/2/3) of the S-boxes of the cipher state.

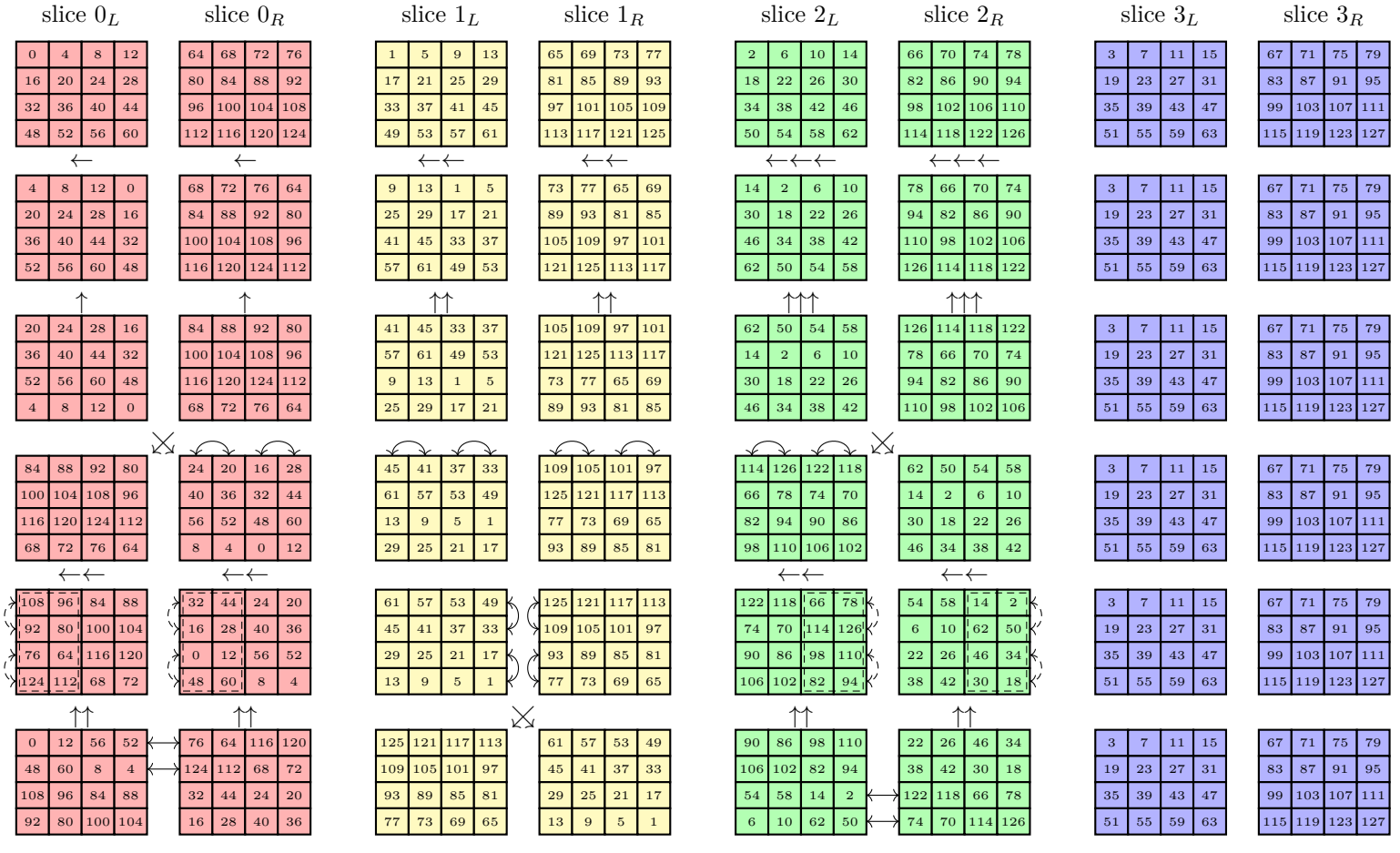


Figure 13.7: New representation of the GIFT-128 round function during 5 rounds. Each cell represents a bit, and the numbers in the cells then denote the actual index of that particular bit in the state. Slice 0 (resp. 1/2/3) depicted in red (resp. yellow/green/blue) represents all the bits at position 0 (resp. 1/2/3) of the S-boxes of the cipher state.

sentation given in Figure 13.6. We recall that this representation implies that the key schedule and constant addition have to be adapted to fit the new way the bits are positioned.

The first 2 rounds are similar to the ones used for GIFT-64. Namely, in the first round, we simply rotate each matrix of each slice i (thus i_L and i_R for all i) by i columns to the left. In the second round, we simply rotate each matrix of each slice i (thus i_L and i_R for all i) by i rows to the top. For the third round, we swap the matrices i_L and i_R for $i \in \{0, 2\}$ before swapping the first and third columns with the second and fourth ones respectively, for matrixes $0_R, 1_L, 1_R$ and 2_L. During the fourth round, we swap the first and third rows with the second and fourth ones respectively, for each matrix of slice 1. Then, for each matrix of slice 0 (resp. slice 2), we rotate by 2 columns to the left before swapping rows of the left-half block (resp. right-half block). Finally, the fifth round consists in swapping 1_L with 1_R, rotating i_L and i_R by 2 rows to the top for $i \in \{0, 2\}$ and swapping the first and second rows of each matrix for slice 0, while swapping the third and fourth rows of each matrix for slice 2. All these operations are illustrated in Figure 13.7 for greater clarity.

The above mentioned method to adapt the key schedule for GIFT-64 cannot be straightforwardly applied to GIFT-128. Indeed, the new and the classical representations of the state are synchronised after 5 rounds, but the key schedule part is almost synchronised after 4 rounds (the key word will return to its original position after 4 rounds, albeit rotated). Thus, it looks like the synchronisation will happen only every $4 \times 5 = 20$ rounds. However, one can remark that twice as much subkey material is used for GIFT-128 compared to GIFT-64, and there the key words used every two rounds are the same (albeit rotated, and for different part of the internal state). Thus, we have an almost synchronisation that will happen only every $2 \times 5 = 10$ rounds instead. In other terms, each key word has to match every new representation of the state at some point. Instead of applying the naive approach for all round keys, which consists in running the key schedule using the classical representation and then rearranging bits, we suggest to apply it only for the first 10 round keys. At this stage, all key words will be expressed in each representation, allowing to adapt the key schedule for each of them, without reordering bits. More details on how to compute the key schedule in the fixsliced representation are given in Appendix B.

13.5. Efficient software implementations of GIFT

This section shows how to take advantage of the fixslicing technique to achieve efficient implementations of GIFT on ARM Cortex-M processors. We also briefly

discuss the gap for other platforms that do not come with an inline barrel shifter or rotate instruction.

13.5.1. GIFT-64

In the case of GIFT-64, thanks to our new fixsliced representation, the linear layer consists in rotating either rows or columns depending on the round number. Depending on how the bits are arranged within the slices (i.e. row-wise or column-wise bitsliced representation), these operations refer to either half-word (16-bit) or nibble (4-bit) rotations. In the rest of this section we consider a row-wise bitsliced representation. The ARM Cortex-M being a 32-bit architecture (and since we have 4 slices in GIFT-64), two 64-bit blocks B and B' can be processed at a time. Instead of simply concatenating 16-bit slices of both blocks within a 32-bit words, we suggest to interleave the nibbles as follows:

$$
\begin{aligned}
S_0 \leftarrow & b_{60}\, b_{56}\, b_{52}\, b_{48}\, b'_{60}\, b'_{56}\, b'_{52}\, b'_{48} \cdots b_{12}\, b_8\, b_4\, b_0\, b'_{12}\, b'_8\, b'_4\, b'_0 \\
S_1 \leftarrow & b_{61}\, b_{57}\, b_{53}\, b_{49}\, b'_{61}\, b'_{57}\, b'_{53}\, b'_{49} \cdots b_{13}\, b_9\, b_5\, b_1\, b'_{13}\, b'_9\, b'_5\, b'_1 \\
S_2 \leftarrow & b_{62}\, b_{58}\, b_{54}\, b_{50}\, b'_{62}\, b'_{58}\, b'_{54}\, b'_{50} \cdots b_{14}\, b_{10}\, b_6\, b_2\, b'_{14}\, b'_{10}\, b'_6\, b'_2 \\
S_3 \leftarrow & b_{63}\, b_{59}\, b_{55}\, b_{51}\, b'_{63}\, b'_{59}\, b'_{55}\, b'_{51} \cdots b_{15}\, b_{11}\, b_7\, b_3\, b'_{15}\, b'_{11}\, b'_7\, b'_3
\end{aligned}
$$

so that 16-bit rotations are now 32-bit rotations, which can be implemented in a single cycle using the `ror` instruction. Actually, it can be computed for free by taking advantage of the inline barrel shifter, since instructions can shift or rotate one of their operands without any additional cost. Therefore, the implementation cost of the linear layer is now equivalent to 42 nibble rotations (3 have to be computed every 2 rounds). Such rotations can be computed in 3 cycles on ARM Cortex-M processors assuming that the required masks are already loaded in some general purpose registers, resulting in a total of $42 \times 3 = 126$ cycles. The following calls to the `SWAPMOVE` routine lead to the above mentioned row-wise nibble-interleaved bitsliced representation.

$S_0 \leftarrow b_{31} \cdots b_0$ $S_1 \leftarrow b'_{31} \cdots b'_0$ $S_2 \leftarrow b_{63} \cdots b_{32}$ $S_3 \leftarrow b'_{63} \cdots b'_{32}$

SWAPMOVE(S_0, S_0, 0x0a0a0a0a, 3); SWAPMOVE(S_1, S_1, 0x0a0a0a0a, 3);
SWAPMOVE(S_2, S_2, 0x0a0a0a0a, 3); SWAPMOVE(S_3, S_3, 0x0a0a0a0a, 3);
SWAPMOVE(S_0, S_0, 0x00cc00cc, 6); SWAPMOVE(S_1, S_1, 0x00cc00cc, 6);
SWAPMOVE(S_2, S_2, 0x00cc00cc, 6); SWAPMOVE(S_3, S_3, 0x00cc00cc, 6);
SWAPMOVE(S_0, S_0, 0x0000ff00, 8); SWAPMOVE(S_1, S_1, 0x0000ff00, 8);
SWAPMOVE(S_2, S_2, 0x0000ff00, 8); SWAPMOVE(S_3, S_3, 0x0000ff00, 8);
SWAPMOVE(S_0, S_1, 0x0f0f0f0f, 4); SWAPMOVE(S_2, S_3, 0x0f0f0f0f, 4);
SWAPMOVE(S_0, S_2, 0x0000ffff, 16); SWAPMOVE(S_1, S_3, 0x0000ffff, 16);

Although a bitsliced representation without interleaving the nibbles could be built for 12 SWAPMOVE instead of 16, each half-word rotation would require at least 3 cycles, therefore doubling the cost of the linear layer to at least 252 cycles. Regarding the non-linear layer, it is possible to save 1 instruction by omitting the NOT operation. Indeed, this operation applies to a slice that will be then exclusive-ORed with the round key. Therefore, we suggest to compute the NOT on the corresponding round keys. Moreover, because the key schedule is completely linear, one can simply apply the logical negation to the right chunks of the key:

$$k_{127} \cdots k_{112} \overline{k_{111} \cdots k_{96}} k_{95} \cdots k_{80} \overline{k_{79} \cdots k_{64}} k_{63} \cdots k_{48} \overline{k_{47} \cdots k_{32}} k_{31} \cdots k_{16} \overline{k_{15} \cdots k_0}$$

before computing the key schedule. Note that this can be done once, when the encryption key is being derived and/or stored on the device, therefore saving 28 cycles per 128-bit data encryption.

On the other hand, a nibble-interleaved bitsliced representation requires twice as much memory to store the round keys and constants in order to avoid extra computations on the fly. It would still be possible to store these variable as 16-bit words but one would have to pay extra cycles to expand them into 32-bit words, nibble-interleaved with theirselves. As a matter of efficiency, we did not consider this option for our implementations. The round keys and constants are stored in 32-bit words, leading to a memory requirement of 112 and 224 bytes for all the round constants and the round keys, respectively.

13.5.2. GIFT-128

Regarding GIFT-128, because only a single block can be processed at a time on 32-bit processors, we consider a row-wise bitsliced representation without any interleaving. Unlike GIFT-64, it is not possible to distinguish only 2 but 5 kind of operation since each step of the new representation requires different slice trans-

formations. At steps 1, 2, 4 and 5, these transformations can be implemented by means of nibble, half-word, byte and full-word rotations, respectively. The third step does not clearly refer to any n-bit rotation but can be simply computed using the SWAPMOVE process. Again, full-word rotations can be implemented for free on ARM thanks to the inline barrel shifter. Even though the nibble, byte and half-word rotations can be implemented in at least 3 cycles, our implementation requires 5 cycles as 2 additional cycles are spent in loading the appropriate masks into registers. This is due to the fact that, unlike GIFT-64, it is not possible to keep all the masks in registers during the entire encryption routine as 12 different ones are needed. The same statement also applies to SWAPMOVE calculations, leading to a cost of 5 cycles per process. As a result, the linear layer of GIFT-128 can be implemented in about $12 \times 5 \times 8 = 480$ cycles in total, according to our new representation.

Note that row ordering matters to match with this interpretation of the new representation. Our GIFT-128 implementations use a row ordering from top-down, which can be achieved using the 14 following calls to the SWAPMOVE process:

$S_0 \leftarrow b_{79} \cdots b_{64}\, b_{15} \cdots b_0$	$S_1 \leftarrow b_{95} \cdots b_{80}\, b_{31} \cdots b_{16}$
$S_2 \leftarrow b_{111} \cdots b_{96}\, b_{47} \cdots b_{32}$	$S_3 \leftarrow b_{127} \cdots b_{112}\, b_{63} \cdots b_{48}$
SWAPMOVE(S_0, S_0, 0x0a0a0a0a, 3);	SWAPMOVE(S_1, S_1, 0x0a0a0a0a, 3);
SWAPMOVE(S_2, S_2, 0x0a0a0a0a, 3);	SWAPMOVE(S_3, S_3, 0x0a0a0a0a, 3);
SWAPMOVE(S_0, S_0, 0x00cc00cc, 6);	SWAPMOVE(S_1, S_1, 0x00cc00cc, 6);
SWAPMOVE(S_2, S_2, 0x00cc00cc, 6);	SWAPMOVE(S_3, S_3, 0x00cc00cc, 6);
SWAPMOVE(S_0, S_1, 0x000f000f, 4);	SWAPMOVE(S_0, S_2, 0x000f000f, 8);
SWAPMOVE(S_0, S_3, 0x000f000f, 12);	SWAPMOVE(S_1, S_2, 0x00f000f0, 4);
SWAPMOVE(S_1, S_3, 0x00f000f0, 8);	SWAPMOVE(S_2, S_3, 0x0f000f00, 4);

Regarding the non-linear layer, contrary to GIFT-64, it is not possible to get rid of the NOT operation within the S-box computation as the round keys are not exclusively-ORed to S_0. Therefore, our implementation of the non-linear layer follows straightforwardly the specification and requires $13 \times 40 = 520$ cycles in total.

13.5.3. Without rotate instruction

Thanks to the inline barrel shifter, our fixsliced implementations fit very well the ARM architecture since the linear layer can be computed for free every 2 and 5 rounds for GIFTb-64 and GIFTb-128, respectively. However, one could ask oneself how it would perform on platforms that do not come with an inline barrel shifter and/or rotate instructions. For instance, RISC-V has no rotate instruction with-

out an appropriate extension (e.g., Bitmanip [388]). In this case, one rotation can be computed by means of 2 shifts and 1 OR, resulting in at least 3 cycles. Therefore, instead of having the linear layer for free every 2 and 5 rounds, it would require at least $4 \times 3 = 12$ cycles, leading to a minimum overhead of $12 \times 14 = 168$ and $12 \times 8 = 96$ cycles for GIFTb-64 and GIFTb-128, respectively. Moreover, nibble, byte and half-word rotations on RISC-V cannot be computed in 3 but 5 cycles because the barrel shifter is not inlined, resulting in an additional overhead of $2 \times 4 \times 14 = 112$ for GIFT-64. On the other hand, this should not affect GIFT-128 since our implementation spends 5 cycles for all these rotations because 2 additional cycles are spent to load the appropriate masks in registers. While ARM Cortex-M processors only have 14 general purpose registers, RISC-V has 32 such registers, so all the masks can be kept in registers during the entire encryption process. Finally, the SWAPMOVE process would require 6 cycles instead of 4, increasing the cost to pack the input and unpack the output to $16 \times 4 = 64$ and $14 \times 4 = 56$ cycles for GIFT-64 and GIFT-128, respectively. Note that it would also add $(6-5) \times 3 \times 8 = 24$ cycles to GIFTb-128 since it relies on 3 SWAPMOVE calls in order to compute the linear layer every 5 rounds.

As a result, on platforms without inline barrel shifter or rotate instruction, we estimate a total overhead of $168 + 112 = 280$ (i.e. 140 per block) and $96 + 24 = 120$ cycles for our fixsliced implementations of GIFTb-64 and GIFTb-128, respectively. Taking into account the overhead to pack/unpack the data would lead to a total overhead of and $280 + 64 = 344$ (i.e. 172 per block) and $120 + 56 = 176$ cycles for GIFT-64 and GIFT-128, respectively. Overall, this means a penalty of around 40% cycles for GIFT-64 and 15% cycles for GIFT-128. Therefore, fixslicing is still of interest on such platforms compared to the classical representation, although the ARM architecture allows to boost its performance.

13.6. Results

13.6.1. The GIFT block ciphers

Our GIFT implementations, which are written in ARM assembly, are put into the public domain and available at `https://github.com/aadomn/gift`. Results for various lightweight block ciphers including GIFT are provided in Table 13.4.

The implementations of RECTANGLE-64/128, SIMON-64/128 and SPECK-64/128 are the ones from scenario 2 - Best execution time - of the FELICS framework [389]. In this scenario, the key schedule is not taken into account as the round keys are assumed to be precomputed and stored in RAM. The benchmark consists in measuring the time required to encrypt 128-bit data using the CTR mode. We followed

the same approach for our GIFT implementations to ensure a fair comparison. The results for PRESENT-64/128 are taken from [378] and were obtained using the same methodology. Regarding the key schedule, results from the FELICS framework were extracted from the scenario 0, which consists in a simple benchmark of the key schedule and a block encryption/decryption. Except for RECTANGLE, for which implementations are written in ARM assembly, note that the results for the other above mentioned ciphers come from C codes. Therefore, better results can be expected for these algorithms by considering assembly implementations. Table 13.4 also includes results for the current best AES constant-time implementation from [390]. Note that, as in Table 13.3, RAM usage for encryption functions does not take into account the memory required for the round keys to be compliant with the results from the FELICS framework.

As expected, our new GIFT fixsliced representation allows extremely efficient software bitsliced implementations, requiring at best 766 and 838 cycles to encrypt 128-bit data for GIFTb-64 and GIFT-64, respectively. Note that this is about 5 times more efficient than our naive bitsliced implementations written in C reported in Table 13.3. On the other hand, the amount of memory to store the round keys is increased by a factor 2. GIFT-64 outperforms all other 64-bit ciphers listed in Table 13.4, except SPECK-64/128 which is well known for its outstanding performances thanks to its ARX structure. Especially, our implementation of the GIFT-64 key schedule according to the new representation outperforms all the other ones. GIFT-64 key exp. refers to the key schedule including the rearrangement of the encryption key to match the fixsliced representation, while GIFTb-64 key exp. assumes a key already in the right representation as input. Note that rearranging the encryption key can be done only once, when this latter is being derived and/or stored on the device, at the same time that the S-box optimization described in Section 13.5.

Regarding GIFT-128, we observe a factor of 1.6 in terms of performance compared to GIFT-64. Considering that the factor in terms of rounds is about 1.4, it is a remarkable result since its new representation is slightly more complex. However, the cost of the key schedule is more than doubled due to the fact that the optimization for GIFT-64 does not apply to GIFT-128 as stated in Section 13.5. Still, it allows a slightly better performance than the AES key schedule. Note that, unlike for GIFT-64, we do not make a distinction between GIFTb-128 key exp. and GIFT-128 key exp. as our adapted key schedule starts from the key in its classical representation anyway. For encryption routines, it results that our GIFT-128 implementations largely outperforms the current best AES one reported in the literature, with GIFTb-128 saving about 28% cycles on AES-128, for a code size 2.9 times smaller when loops are fully unrolled. It requires 1169 and 1316 cycles for GIFTb-128 and GIFT-128, respectively, which is about 7 times more efficient than

our naive bitsliced implementations reported in Table 13.3. Moreover, note that these AES results benefit from being averaged over 4096 bytes encryption, versus 16 bytes for GIFT-128.

13.6.2. Adding first-order masking

On top of running in constant-time, secure embedded cryptographic implementations should integrate countermeasures against power based side-channel analysis since they are typical targets for these kind of attacks. A well-known approach to overcome such attacks is the masking countermeasure [391], which consists in splitting intermediate values in statistically independent shares by means of random masks. Thereafter, we report implementation results when applying first-order masking (i.e. splitting all intermediate values in two shares).

While linear operations can be simply computed on both shares independently, non-linear operations are more challenging to compute securely. In the case of GIFT, the only non-linear gates are 1 OR and 3 AND during the S-box computation. Our masked implementations rely on the secure AND and OR operations introduced in [392], which run in 6 cycles on ARM Cortex-M processors and do not require any additional randomness. Table 13.5 reports implementation results for GIFT and AES on ARM Cortex-M4 as the STM32F407VG incorporates a random number generator. Because the number of clock cyles required to generate random numbers is platform dependent, it is enclosed in parentheses separately. Our GIFT implementations only require 4 32-bit random words to mask the internal state at the beginning of the algorithm. Regarding the key schedules, the same amount of randomness is required to mask the initial key. For both GIFT-64 and GIFT-128, the internal state fits in 4 registers. Therefore, it is possible to handle the state and the masks in 8 registers, avoiding any additional memory access during the encryption routine.

When taking first-order masking into consideration, the advantage of GIFT-128 over AES-128 is even more significant since the number of non-linear operations to secure is smaller. However, note that the reported results for AES do not take advantage of the optimized AND gate from [392] and therefore bear the cost of additional operations and randomness generation. Compared to our unmasked implementation results reported in Table 13.4, we observe a penalty factor about 2.5 in terms of execution time, showing that GIFT is well suited for software masked implementations thanks to our fixsliced representation.

13.6.3. The GIFT-COFB authenticated cipher

Since GIFTb-128 defines the underlying block cipher of GIFT-COFB, we can easily have a look at the benefits of our fixsliced representation when applied to this authenticated cipher. To do so, our GIFT-COFB implementation computes the COFB mode using C code while calls to the GIFTb-128 primitive are handled by our assembly implementation. Tables 13.6 and 13.7 summarize our implementation results for GIFT-COFB and Ascon [393], another submission to the NIST LWC competition. For both versions of Ascon, namely Ascon-128 and Ascon-128a, we consider the ARM optimized implementations `bi32_arm`, available online at `https://github.com/ascon`. We believe this is a fair comparison since the core function is written in assembly in a fully unrolled manner, while the rest of the algorithm is handled by C code, just like our GIFT-COFB implementation.

According to our benchmark, fixslicing makes GIFT-COFB a very efficient authenticated cipher, running at 79 cycles per byte for long messages, versus 58 and 42 cycles per byte under the same setting for Ascon-128 and Ascon-128a, respectively. However, because the considered Ascon implementation are highly speed-optimized, their code size are bigger than our fully unrolled implementation by a factor 1.2 and 1.5 for Ascon-128 and Ascon-128a, respectively. We observe that our first-order masked implementation of GIFT-COFB requires about thrice as much cycles as Ascon-128, taking into account the randomness generation on the STM32F407VG micro-controller. Although it is unclear how Ascon-128 would perform compared to our fixsliced implementations when taking first-order masking into account, we expect it to be more efficient for messages composed of several blocks since masking can be restricted to the initialization and finalization phases as done in [394].

13.7. Conclusion

In this article, we proposed a new representation for the GIFT family of lightweight block ciphers called fixslicing, and showed how it can be used to obtain extremely fast implementations on micro-controllers, making GIFT a very efficient candidate on these platforms. Especially, our fixsliced representation fits very well the ARM architecture as the inline barrel shifter allows to compute the linear layer for free every 2 and 5 rounds for GIFT-64 and GIFT-128, respectively. Our implementations, available online at `https://github.com/aadomn/gift` to validate our overall strategy, run in constant-time since they are bitsliced in essence. This result directly provides efficient implementations of GIFT-COFB, a submission to the NIST LWC competition, placing it as a very promising candidate on micro-controllers.

We also report implementation results for GIFT and GIFT-COFB when adding first-

order masking and observe a penalty factor about 2.5 and 2.1, respectively. According to our benchmark, GIFT-COFB masked at first-order requires about thrice as much cycles than Ascon-128 without masking. Further work should be conducted to draw a clear picture when comparing both algorithms regarding masked implementations.

More generally, we believe that the approach of not following the classical cipher representation for a few rounds might be applicable to other designs. Especially, bitsliced implementations can take advantage of the fixslicing technique as long as each bit located in a slice remains in the same one through the linear layer, as is the case for GIFT. From a design point of view, considering a permutation with a low order for the linear layer might be of interest, since it allows to define a compact routine to resynchronise the slices. Furthermore, the key schedule should be designed accordingly to avoid any additional calculations due to round keys adjustment.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. The authors are supported by a Temasek Labs grant (DSOCL16194) and a joint WASP/NTU grant.

bib$_s$*hort*

Algorithm	Ref	Parallel Blocks	Speed (cycles)		ROM (bytes)		RAM (bytes)	
			M3	M4	Code	Data	I/O	Stack
64-bit ciphers with 128-bit key								
GIFTb-64 key exp.	Ours	-	**339**	**338**	**972**	**0**	**224**	**52**
			381	*383*	*226*	*0*	*224*	*56*
GIFTb-64 encryption		2	**383**	**383**	**2666**	**0**	**40**	**48**
			415	*415*	*756*	*112*	*40*	*52*
GIFT-64 key exp.		-	**488**	**487**	**1575**	**0**	**224**	**52**
			530	*533*	*828*	*0*	*224*	*56*
GIFT-64 encryption		2	**419**	**419**	**2962**	**0**	**40**	**48**
			458	*456*	*1058*	*112*	*40*	*52*
PRESENT key exp.	[378]	-	**5043**	**3464**	•	•	•	•
PRESENT encryption		2	**1058**	**800**	**2476**	•	•	•
RECTANGLE key exp.	[389]	-	**1106**	•	**157**		**232**	**44**
			1106		*157*		*232*	*44*
RECTANGLE encryption		1	**854**	•	**800**		**76**	**24**
			1185		*440*		*52*	*24*
SIMON-64 key exp.	[389]	-	**1195**	•	**112**		**200**	**8**
			1202		*108*		*200*	*12*
SIMON-64 encryption		1	**650**	•	**456**		**48**	**24**
			1281		*336*		*40*	*24*
SPECK-64 key exp.	[389]	-	**475**	•	**46**		**132**	**12**
			484		*46*		*132*	*12*
SPECK-64 encryption		1	**285**	•	**628**		**36**	**24**
			518		*254*		*36*	*24*
128-bit ciphers with 128-bit key								
AES-128 key exp.	[390]	-	**1028**	**1034**	**3384**	**1036**	**368**	**188**
AES-128 encryption		2	**1617**	**1618**	**12120**	**12**	**48**	**108**
GIFT-128 key exp.	Ours	-	**966**	**969**	**3510**	0	**320**	**48**
			1813	*1812*	*1100*	*0*	*320*	*56*
GIFTb-128 encryption		1	**1169**	**1172**	**4250**	**0**	**48**	**56**
			1297	*1279*	*834*	*160*	*48*	*64*
GIFT-128 encryption		1	**1316**	**1319**	**4868**	**0**	**48**	**56**
			1444	*1427*	*1332*	*160*	*48*	*64*

Table 13.4: Constant-time implementation results on ARM Cortex-M3 and M4 for various versions and representations of GIFT, as well as other lightweight block ciphers. For encryption routines, speed is expressed in cycles per block. Emboldened (resp. italic) results refer to speed (resp. code size) oriented implementations.

Algorithm	Ref	Parallel Blocks	Speed (cycles)	ROM (bytes)		RAM (bytes)	
				Code	Data	I/O	Stack
64-bit ciphers with 128-bit key							
GIFTb-64 key exp.	Ours	-	**641** (+196) *723* (+196)	**1950** *360*	**0** *0*	**448** *448*	**68** *68*
GIFTb-64 encryption	Ours	2	**911** (+98) *1014* (+98)	**6646** *1516*	**0** *112*	**40** *40*	**64** *72*
GIFT-64 key exp.	Ours	-	**965** (+196) *993* (+196)	**3160** *944*	**0** *0*	**448** *448*	**68** *68*
GIFT-64 encryption	Ours	2	**940** (+98) *1051* (+98)	**6942** *1722*	**0** *112*	**40** *40*	**64** *72*
128-bit ciphers with 128-bit key							
AES-128 encryption	[390]	2	**5290** (+2133)	**39916**	**12**	**48**	**1588**
GIFT-128 key exp.	Ours	-	**1994** (+196) *2202* (+196)	**6816** *2168*	**0** *0*	**640** *640*	**64** *72*
GIFTb-128 encryption	Ours	1	**2815** (+196) *3049* (+196)	**10266** *1532*	**0** *160*	**48** *48*	**64** *72*
GIFT-128 encryption	Ours	1	**2972** (+196) *3203* (+196)	**10906** *2172*	**0** *160*	**48** *48*	**64** *72*

Table 13.5: Masked constant-time implementation results of GIFT and AES on ARM Cortex-M4. For encryption routines, speed is expressed in cycles per block. Emboldened (resp. italic) results refer to speed (resp. code size) oriented implementations.

Algorithm	Ref	Speed (cycles)		ROM (bytes)		RAM (bytes)	
		M3	M4	Code	Data	I/O	Stack
Without masking							
GIFT-COFB	Ours	**4827** *6028*	**4893** *6082*	**10092** *4240*	**0** *160*	**428** *428*	**92** *100*
Ascon-128	https://github.com/ascon (Our measurements)	**4203**	**4276**	**12348**	**0**	**124**	**36**
Ascon-128a		**3862**	**3990**	**15200**	**0**	**140**	**36**
With 1st-order masking (including randomness generation)							
GIFT-COFB	Ours	•	**10978** (+579) *11928* (+579)	**19808** *5096*	**0** *160*	**732** *732*	**108** *100*

Table 13.6: Constant-time implementation results on ARM Cortex-M3 and M4 for GIFT-COFB and Ascon to secure 16 bytes of message along with 16 bytes of additional data. Emboldened (resp. italic) results refer to speed (resp. code size) oriented implementations.

Algorithm	Ref	Message size (bytes)					
		16	64	256	1024	4096	16384
Without masking							
GIFT-COFB	Ours	4893	8725	23929	84701	327581	1299101
Ascon-128	https://github.com/ascon (Our measurements)	4276	7073	18246	62886	241446	955686
Ascon-128a		3990	6028	14171	46715	176891	697595
With 1st-order masking (including randomness generation)							
GIFT-COFB	Ours	11557	20824	57773	205572	796733	3161412

Table 13.7: Running time (cycles) of constant-time speed-oriented implementations of GIFT-COFB and Ascon on ARM Cortex-M4 for different message sizes along with 16 bytes of additional data.

VII

Conclusion

14

Conclusion

14.1. Achievements

In summary, here are the questions addressed in this Ph.D. thesis with their answers:

*RQ*1

What is the actual cost of a differential cryptanalysis attack when the cipher's security is near or below the 80-bit security threshold?

We showed that ASICs present a significant threat to primitives with 64-bit security and even pose a risk to those with 80-bit security. The chosen-prefix collision on SHA-1, achievable with an ASIC cluster costing a few million, expands the attack surface to TLS and SSH. The first chapter of this thesis underscores the need for regular security evaluations and updates. We also showed how minor inaccuracies in attack parameters can significantly impact overall cost estimates. However, this research has limitations as it cannot be easily applied to other cryptographic primitives like AES. Further study of the cost of other real-world attacks like the biclique attack would be beneficial.

*RQ*2

How can we practically prevent side channel attacks?

To explore this inquiry, we began our analysis by examining the practical advantages of using boolean S-boxes, like those employed in AES, compared to other cipher structures to implement resistance to side-channel attacks.

In Chapter 4, we underscore the importance of selecting the appropriate cipher by comparing AES and ChaCha20's resistance to side-channel attacks. While ChaCha20 exhibits natural resistance to timing attacks and may be suitable for specific applications, it is susceptible to power side-channel attacks. AES, on the other hand, is preferred for defense against power side-channel attacks, but ChaCha20 may prove advantageous for applications where power side-channel attacks are not a concern due to its superior software performance and innate defense against timing attacks. Our evaluation of the primary non-linear components of these two ciphers indicates that the addition in ChaCha20 is marginally more challenging to attack than AES's S-box. However, it also has a more significant overhead for protection.

Chapter 5 shows the difference between theory and practice when using memory-contained provable secure software masked high-order masked implementations. The chapter aims to showcase the effect of choosing an appropriate security model while creating security proof and evaluating the practical security level of implementation.

Our mathematical analysis showed that a first-order multivariate attack exploits several leakages connected to one mask during table re-computation and can surpass the efficacy of a classical high-order differential power analysis (HODPA) attack on the same masked implementation. We emphasized the importance of considering the number of exploitable variables and noise variance as security metrics since these factors can significantly impact the attack's effectiveness. It is worth noting that implementing a higher-order masking scheme does not necessarily guarantee a higher security level. The cost of implementing a high-order masking scheme can be too high, and any trade-off in how it is implemented can affect the security advantage it provides.

Our findings were further confirmed on a real smartcard. Future research will address the challenge of protecting table re-computation from such attacks and examining the potential application of multivariate attacks on other masking schemes and defensive techniques, such as Threshold masking implementation schemes. However, one of this attack's limitations is identifying Points of Interest (PoI) in a black box or non-profiled attack setting. In order to identify PoIs, a white-box setting

with profiling is required. Another issue is how to avoid unintended indirect leakages that occur due to the uncontrollable underlying architecture of the processor executing the code.

The problem of PoI selection in a non-profiled SCA setting is investigated in the following chapter. The context of the study is the security evaluation of Elliptic curve cryptography (ECC) against horizontal side-channel attacks. However, the same methodology could be used for SCA evaluations of masked symmetric cryptographic implementations. To improve the precision of non-profiled attacks, several feature selection methods are examined and validated using actual measurements acquired from FPGA and microcontroller targets.

We tested several non-profiled feature selection techniques and discovered that some are nearly as precise as profiled attacks. This is a crucial finding as it suggests that the black-box setting constraint does not offer a substantial security advantage.

Preventing side-channel attacks requires careful design, rigorous evaluation, and effective countermeasures.

Based on our research, here are some possible measures that can be taken to prevent side-channel attacks:

(a) Choose the appropriate primitive with natural resistance to the specific type of side-channel attacks the system is vulnerable to.

(b) Select a suitable security model during the creation of a security proof and when evaluating the practical security level of an implementation.

(c) Consider the number of exploitable variables and noise variance as security metrics when evaluating the effectiveness of a security measure.

(d) Implement an appropriate masking scheme that balances security and performance.

(e) Preferably use constant-time bit sliced cryptographic implementations in software

(f) When evaluating the implementation, make sure to identify Points of Interest (PoI) in a white-box setting with profiling to avoid unintended indirect leakages that occur due to the underlying architecture of the processor executing the code.

(g) use feature selection techniques to enhance accuracy in non-profiled attacks.

In the future, one promising research direction for AI and neural network-based attacks could be further enhancing feature selection. Additionally, the research could focus on the explainability of neural networks to identify the source of a successful neural network-based attack. AI-based attacks have the potential to factor in indirect leakages created by the underlying architecture, which can be challenging to predict to designers and may be invisible to evaluators who use traditional side-channel evaluation methodologies.

How can we protect a system against attacks exploiting leakages we cannot even predict? We could imagine trainable countermeasures that would learn about themselves. Is this even achievable?

3.

*RQ*3

What methods or techniques can be employed to detect and prevent practical fault attacks such as EMFI?

The primary step to address this question is to identify a generic approach to compare the resistance of various existing ciphers against fault attacks. Since this task can be challenging, we narrowed our study to the DFA attack model on symmetric block ciphers.

In Chapter 7, we present a comprehensive overview of existing DFA attacks on Substitution-Permutation Network (SPN) ciphers. Then, we introduce a generic DFA attack method for this family of ciphers. Our proposed method involves the development of a novel approach to identify appropriate fault masks and the Joint Difference Distribution Table (JDDT) tool to pre-compute solutions for fault equations. This technique successfully recovers the last round key with minimal faulty and non-faulty cipher text pairs. We demonstrate the effectiveness of our approach on several block ciphers, including AES-128, PRESENT-80 and 128, GIFT-64 and 128, and LED-64 and 128. The outcomes of this research could lay the foundation for future efforts to design fault-resistant block ciphers and explore the application of our approach to other block cipher designs, possibly leading to the creation of a fully automated and integrated tool with cryptographic design and analysis software.

As a potential future research direction, it would be valuable to expand this work to other families of ciphers, symmetric cryptographic

schemes, and fault models.

We then investigated the efficacy of using the private circuits II as a potential defense mechanism against fault injection attacks on symmetric cryptography. Although this approach may be too expensive to implement on asymmetric cryptographic circuits, it is still feasible for less complex symmetric cryptographic circuits in specific scenarios.

Private Circuit II offers security proof of resistance against probing attacks, considered the most potent form of passive and active side channels and attacks.

This thesis presents the first implementation of private circuits II in an FPGA. A security analysis assesses its effectiveness against classical Side Channel and Fault Injection attacks. The results showed that while Private Circuits II protected against read and reset attacks, it was still susceptible to correlated faults that could result in exploitable ciphertexts.

The cause of this issue is that the fault models observed in real-world scenarios are intricate and go beyond what can be adequately addressed by the active and passive bit probing attacker model employed in the security proof of Private Circuit-II. As a result, it needs to be sufficiently robust to thwart the occurrence of complex fault models in reality, leading to correlated outputs.

A key challenge is to accurately model the faults occurring in reality to bridge the gap between theory and practice. This task is challenging due to process fabrication variations, which can result in chip-specific fault models. Furthermore, in experimental testing setups, numerous sources of variability further complicate the task.

Our experiments demonstrate that employing an on-chip monitor can enhance the comprehension of the types of faults that occur in reality. Moreover, an on-chip monitor can be utilized to study the impact of faults on circuits, obviating the need for experimental post-silicon fault injection characterization setups.

Then, in chapter 9, the use of ring extension as a fault detection method for asymmetric cryptography is explored and proposed. The ring extension is applied on twisted Edwards Curves and was shown to be a simple and effective solution. However, it involved some technicalities related to the curve parameters and base point. Various variants are investigated a new "test-free" variant that is proven secure and efficient is proposed. Implementing the elliptic curve scalar multiplication (ECSM) algorithm protected with the test-free modular extension is evaluated

on Edwards and twisted Edwards curves. The results demonstrate the effectiveness of the proposed method in detecting faults and maintaining the integrity of the computation result. Despite some technicalities in applying the ring extension method, the provable fault detection method for (twisted) Edwards curves is novel, elegant, and effective in securing cryptographic systems from fault injection attacks.

In chapter 10, we showcase a solution in the form of a PLL-based sensor circuit and a comprehensive automated design flow on an FPGA platform able to detect Electromagnetic Fault Injections. This solution provides a high fault detection rate and low hardware impact. Although other methods of avoiding fault injection attacks have been studied, the proposed solution serves as a positive step in enhancing the security of ICs by safeguarding sensitive information. However, it must be noted that this approach is limited to FPGAs and ASICs and does not apply to Microcontrollers. Additionally, while the physical countermeasure presents a high detection rate, it is not foolproof and should be used with other protection mechanisms for maximum security. Further research is necessary to understand unexpected fault behavior and to find more effective solutions.

We wish to caution designers about the potential dangers of utilizing a technology feature for a single purpose. In Chapter 11, we illustrate an instance in which an FPGA feature that designers often consider appealing for deterring Side Channel Attacks can prove to be a double-edged sword as attackers can exploit it to introduce a Hardware Trojan surreptitiously. This highlights the significance of conducting a thorough risk analysis that accounts for all possible attack vectors while designing a connected system.

In summary, based on our research, the following list of factors can help ensure the prevention of fault attacks:

(a) Selection of an appropriate primitive for the cryptographic algorithm

(b) Adoption of a suitable security model that can provide comprehensive protection against possible attacks

(c) Use of the right fault model that can accurately reflect the types of faults that occur in real-world scenarios

(d) Accurate modeling of faults that occur in reality, which can help in designing more robust cryptographic systems

(e) Employment of RTL, post-synthesis simulations, and an on-chip monitor to enhance the understanding of the types of faults that occur in reality

(f) Implementation of suitable data-level countermeasures such as data path time-space redundancy or ring extensions according to the identified occurring fault models

(g) Study of the processor architecture to identify if any assembly-level macro can be used to prevent fault propagation

(h) Consideration of the usage of physical-level countermeasures, when feasible, in conjunction with other techniques for maximum protection level

(i) Conducting a thorough risk analysis that accounts for all possible attack vectors while designing a connected system

4.

*RQ*4

What potential guidelines or principles could designers utilize to enhance the rigor in the design process of a security-critical application?

One can systematize the acquired knowledge by comprehending distinct attack sources and constraints on countermeasures. By examining the literature, we discuss in chapter 12 the definition of "security" from a designer's standpoint and show that security is a design constraint. The importance of considering security as a primary architectural design constraint, time, space, and power are discussed. The conflict between security and efficiency objectives in multiple design layers has been highlighted, and numerous examples have been presented to support this idea. The need for a security-aware design flow has been emphasized, starting from the choice of cryptographic primitives to system design, in order to avoid severe security breaches. A security-aware design flow is proposed.

Some potential guidelines include the following elements:

(a) Threat analysis: Understand the distinct attack sources and constraints on countermeasures in order to systematize the acquired knowledge.

(b) Consider security as a primary architectural design constraint: Along with time, space, and power, designers must prioritize security as a critical consideration in the design process.

(c) Address the conflict between security and efficiency objectives: Designers must balance security and efficiency objectives in multiple design layers.

(d) Adopt a security-aware design flow: from the choice of cryptographic primitives to system design.

(e) Test the resulting implementation on actual targets: Despite the methodology used, testing the implementation on actual targets is still necessary, as the underlying architecture may result in unpredictable behavior.

In the final chapter, the principles presented in the previous chapters are put into practice in a real-world scenario involving selecting and implementing a lightweight cipher for IoT applications within the NIST Lightweight Cipher competition. To achieve this, a novel cipher representation, known as fix-slicing, is proposed, facilitating highly efficient software bit-sliced implementations of block ciphers on microcontrollers. The proposed methodology is applied to the GIFT family of block ciphers. It is demonstrated that by using this approach, our constant-time implementations of GIFT-64 and GIFT-128 outperform the fastest current constant-time AES implementation in terms of speed. This approach of not following the traditional cipher representation for a few rounds can also be applied to other designs and has been independently applied to the AES family of block ciphers in a subsequent study, highlighting the generality and efficacy of our approach.

However, testing the resulting implementation on actual targets is still necessary, as the underlying architecture may result in unpredictable behavior. This methodology enables a just comparison of various block ciphers for IoT mass-market application scenarios, where general-purpose microcontrollers are employed to decrease costs while delivering the flexibility of post-production secure firmware updates. Moreover, this methodology can be extended to other block cipher families, such as the ASCON family of lightweight block ciphers based on SHA-3 S-boxes.

14.2. Reflection and future work

When evaluating the security of a system, it is essential to identify the factors of risk and exposure. Moreover, it is necessary to determine whether these factors are worth the value and benefits of the assets being protected, while considering the trade-offs between security, availability, and usability. However, these factors are often challenging to formalize and quantify, and there is no straightforward approach to measuring security.

One way to approach the challenge of measuring security is to think in four dimensions or explore alternative approaches. For instance, assessing the security level can involve determining the minimum expenditure of both time and money required to traverse any of the potential routes of attack in a directed graph that has been weighted. Each node in the graph represents an attack vector, which an attack model, a target, and a distinguisher define. However, assessing the level of security can be particularly challenging in situations where soft parameters, such as the signal-to-noise ratio, are involved.

Machine learning can play a crucial role in bridging the gap between theory and practice by fitting attack data, but the explainability of machine learning-based attacks poses a challenge. In addition, side-channel and fault injection attacks are particularly challenging to evaluate. For example, the metrics used to evaluate side-channel attacks lack adherence, making it challenging to compare the security of different algorithm implementations against these attacks. Meanwhile, formal security proof under a given fault model does not guarantee practical security due to the difference between fault models used in the proof and those occurring in practice.

Safeguarding a circuit from poorly understood attacks is another challenge that designers face. However, there are potential solutions available, such as designing an analog-digital circuit that can adapt its power consumption based on the data obtained from power consumption analysis or developing a circuit that introduces artificial power data points, acting as a generic adversarial defense. Machine learning techniques can also help detect sensitive points in a circuit and identify optimal locations to introduce fault injection sensors.

In conclusion, there is no straightforward approach to evaluating security, and many unanswered questions remain. Machine learning may help solve some of the challenges associated with measuring security, but the explainability of machine learning-based attacks is still a concern. Overall, a multidisciplinary approach is necessary to tackle the complexities of cybersecurity and protect valuable data. Therefore, future research is needed to address the questions raised in this thesis and further advance the field of cybersecurity.

The following table summarizes Zakaria Najm's contributions to the publications constituting the chapters of this thesis.

Table 14.1: Publications summary and contributions

Chapter	Reference	Contributions
chapter 2 (Part II)	"Mustafa Khairallah, Zakaria Najm, Anupam Chattopadhyay, Thomas Peyrin: Crack me if you can: hardware acceleration bridging the gap between practical and theoretical cryptanalysis?: a Survey. SAMOS 2018: 167-172 "	Wrote state-of-the-art, and presented paper at SAMOS
chapter 3 (Part II)	"Anupam Chattopadhyay, Mustafa Khairallah, Gaëtan Leurent, Zakaria Najm, Thomas Peyrin, Vesselin Velichkov: On the Cost of ASIC Hardware Crackers: A SHA-1 Case Study. CT-RSA 2021: 657-681"	Wrote results section and ASIC vs GPU comparision, designed ASIC (RTL+backend)
chapter 4 (Part II)	"Zakaria Najm, Dirmanto Jap, Bernhard Jungk, Stjepan Picek, Shivam Bhasin: On Comparing Side-channel Properties of AES and ChaCha20 on Microcontrollers. APCCAS 2018: 552-5558"	Implemented masked CHACHA, worked on results section and presented paper at DSP
chapter 5 (Part III)	"Multivariate High-Order Attacks of Shuffled Tables Recomputation." by Nicolas Bruneau, Sylvain Guilley, Zakaria Najm, Yannick Teglia in Journal of Cryptology 2018 p351-393."	Initiated idea, Implemented TR masking on AVR + setup and sca measurement and wrote result section
chapter 6 (Part III)	"Feature selection methods for non-profiled side-channel attacks on ecc Bernhard Jungk, Dirmanto Jap, Zakaria Najm, Shivam Bhasin,Prasanna Ravi: Feature Selection Methods for Non-Profiled Side-Channel Attacks on ECC. DSP 2018: 1-5"	Initiated idea, wrote background and result sections, presented paper at DSP
chapter 7 (Part IV)	"SoK: On DFA Vulnerabilities of Substitution-Permutation Networks by Mustafa Khairallah, Xiaolu Hou, Zakaria Najm, Jakub Breier, Shivam Bhasin, Thomas Peyrin in ACM AsiaCCS 2019 p403-414."	Participated to brainstorming and shared experience on Private circuit II and EMI, Participated to the experiments and building the methodology
chapter 8 (Part IV)	"Henitsoa Rakotomalala, Xuan Thuy Ngo, Zakaria Najm, Jean-Luc Danger, Sylvain Guilley: Private circuits II versus fault injection attacks. ReConFig 2015: 1-9"	Helped intern on the implementation and simulations on Xilinx FPGA and setup Fault injection experiments + Chip-scope dynamic analysis
chapter 9 (Part IV)	"Using modular extension to provably protect Edwards curves against fault attacks. J. Cryptogr. Eng. 7(4): 321-330 (2017) "	Conducted experiments and generated results (Implementation on AVR platform) + EMI + comparision theory vs practice
chapter 10 (Part IV)	" by Noriyuki Miura, Zakaria Najm, Wei He, Shivam Bhasin, Xuan Thuy Ngo, Makoto Nagata, Jean-Luc Danger which is published in ACM/IEEE Design Automation Conference (DAC'2016)."	Initiated idea and discovered the behaviour of the PLL during experiments on EMI + conducted all the experiments + wrote concept description section + results
chapter 11 (Part IV)	"Debapriya Basu Roy, Shivam Bhasin, Sylvain Guilley, Jean-Luc Danger, Debdeep Mukhopadhyay, Xuan Thuy Ngo, Zakaria Najm: Reconfigurable LUT: A Double Edged Sword for Security-Critical Applications. SPACE 2015: 248-268"	Initiated idea of RLUT based hardware trojan exploiting Xilinx scan-chain, and worked on POC
chapter 12 (Part V)	Prasanna Ravi, Zakaria Najm, Shivam Bhasin, Mustafa Khairallah, Sourav Sen Gupta, Anupam Chattopadhyay: Security is an architectural design constraint. Microprocess. Microsystems 68: 17-27 (2019)	Main paper contributor and initiator and wrote section on Fault, SCA and Cryptanalysis + contributed to security aware design flow section
chapter 13 (Part VI)	"Fix-slicing: A New GIFT Representation Fast Constant-Time Implementations of GIFT and GIFT-COFB on ARM Cortex-M" by Alexandre Adomnicai, Zakaria Najm, Thomas Peyrin in IACR TCHES 2020(3) p402-420	Implemented different versions of GIFT64 and 128 and worked on optimizations + main initiator of the fix-slicing concept and comparison table with others ciphers

Acknowledgments

I wish to express my gratitude to my promotors Pieter Hartel and Stjepan Picek for their patience and invaluable guidance throughout the redaction of this thesis. I would also like to extend my thanks to Sylvain Guilley for his interesting discussions and insightful recommendations. Furthermore, I am grateful to Shivam Bhasin, Anupam Chattopadhyhai, Jean-Luc Danger, Thomas Peyrin, Mustafa Khairallah, Prassana Ravi, Xuan Thuy Ngo, Nicolas Bruneau, Pablo Rauzy, and Margaux Dugardin and all the researcher I had the chance to collaborate with for their brilliant ideas, collaborative efforts, and valuable recommendations. Lastly, a special thanks goes to my sister Yasmine Najm for her feedback and assistance in improving my work and my mom for her support.

Biography

Zakaria Najm

Zakaria Najm is a seasoned cybersecurity professional with over 13 years of experience in cryptography and security for embedded systems. He specializes in automotive product and government sector security and has a proven track record in secure hardware architectures, threat analysis, system-level security analysis, embedded firmware, secure OS, protocols, and high-performance computing. He is a curious, autonomous, and strong analytical thinker who enjoys sharing new ideas and working as part of a team.

Currently, Zakaria is pursuing his Ph.D. at Delft University of Technology, focusing on hardware security. He holds a Master of Science degree from University of Grenoble in Security Cryptology and Coding of Information, a Postgraduate Master degree from the University of Dundee in Microelectronics, and a French "Grandes Ecoles" Graduate Engineer degree from INSA, Blois in Electrical, Electronics, and Computer Science Engineering.

Zakaria has an extensive list of publications, which include over 40 papers published during his time as a Research Engineer at Telecom ParisTech-CNRS and Research Associate at NTU Singapore. Some of his notable works include "High precision fault injections on the instruction cache of ARMv7-M architectures," "NICV: normalized inter-class variance for detection of side-channel leakage," and "Linear complementary dual code improvement to strengthen encoded circuit against hardware Trojan horses."

Throughout his career, Zakaria has held various positions, including Senior Cyber Security Architect and Security Lead at Continental AG, Cyber-Edge-to-Cloud Program Manager at SECURE-IC PTE LTD, and Research Associate at TL@NTU/SYLLAB@SPMS. He has also worked as a Security Design Architect at STMicroelectronics and conducted research in cryptography and security, leading to multiple awards and recognitions.

Zakaria is fluent in English, French, and Arabic, and has basic knowledge of German. In his free time, he enjoys running, cycling, karate, rock climbing, music, and traveling accross the world and especially in Asia.

List of Publications

1. High precision fault injections on the instruction cache of ARMv7-M architectures (2015 IEEE International Symposium on Hardware Oriented Security and Trust ..., 2015) (Cited by 101)
2. NICV: normalized inter-class variance for detection of side-channel leakage (2014 International Symposium on Electromagnetic Compatibility, Tokyo, 310-313, 2014) (Cited by 101)
3. Linear complementary dual code improvement to strengthen encoded circuit against hardware Trojan horses (2015 IEEE International Symposium on Hardware Oriented Security and Trust ..., 2015) (Cited by 78)
4. Hardware Trojan detection by delay and electromagnetic measurements (2015 Design, Automation Test in Europe Conference Exhibition (DATE), 782-787, 2015) (Cited by 57)
5. Analysis and improvements of the DPA contest v4 implementation (Security, Privacy, and Applied Cryptography Engineering: 4th International ..., 2014) (Cited by 55)
6. Hardware property checker for run-time hardware trojan detection (2015 European Conference on Circuit Theory and Design (ECCTD), 1-4, 2015) (Cited by 46)
7. Side-channel leakage and trace compression using normalized inter-class variance (Proceedings of the Third Workshop on Hardware and Architectural Support for ..., 2014) (Cited by 46)
8. Pll to the rescue: a novel em fault countermeasure (Proceedings of the 53rd Annual Design Automation Conference, 1-6, 2016) (Cited by 43)
9. Dismantling real-world ECC with horizontal and vertical template attacks (Constructive Side-Channel Analysis and Secure Design: 7th International ..., 2016) (Cited by 42)
10. Method taking into account process dispersion to detect hardware Trojan Horse by side-channel analysis (Journal of Cryptographic Engineering 6, 239-247, 2016) (Cited by 38)

11. Fixslicing: A new gift representation (Cryptology ePrint Archive, 2020) (Cited by 37)
12. Cryptographically secure shield for security IPs protection (IEEE Transactions on Computers 66 (2), 354-360, 2016) (Cited by 37)
13. A look into SIMON from a side-channel perspective (2014 IEEE International Symposium on Hardware-Oriented Security and Trust ..., 2014) (Cited by 35)
14. Time-frequency analysis for second-order attacks (Smart Card Research and Advanced Applications: 12th International Conference ..., 2014) (Cited by 34)
15. Multivariate high-order attacks of shuffled tables recomputation (Journal of Cryptology 31, 351-393, 2018) (Cited by 31)
16. Formally proved security of assembly code against power analysis: A case study on balanced logic (Journal of Cryptographic Engineering 6, 201-216, 2016) (Cited by 23)
17. A low-entropy first-degree secure provable masking scheme for resource-constrained devices (Proceedings of the Workshop on Embedded Systems Security, 1-10, 2013) (Cited by 22)
18. Integrated sensor: a backdoor for hardware Trojan insertions? (2015 Euromicro Conference on Digital System Design, 415-422, 2015) (Cited by 19)
19. Encoding the state of integrated circuits: a proactive and reactive protection against hardware trojans horses (Proceedings of the 9th Workshop on Embedded Systems Security, 1-10, 2014) (Cited by 19)
20. Formally Proved Security of Assembly Code Against Leakage. (IACR Cryptol. ePrint Arch. 2013, 554, 2013) (Cited by 18)
21. Optimized linear complementary codes implementation for hardware trojan prevention (2015 European Conference on Circuit Theory and Design (ECCTD), 1-4, 2015) (Cited by 14)
22. Correlated extra-reductions defeat blinded regular exponentiation (Cryptographic Hardware and Embedded Systems–CHES 2016: 18th International ..., 2016) (Cited by 13)
23. On comparing side-channel properties of AES and ChaCha20 on microcontrollers (2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 552-555, 2018) (Cited by 12)

24. Feature selection methods for non-profiled side-channel attacks on ecc (2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), 1-5, 2018) (Cited by 11)

25. Reconfigurable LUT: A double edged sword for security-critical applications (Security, Privacy, and Applied Cryptography Engineering: 5th International . . . , 2015) (Cited by 11)

26. Security is an architectural design constraint (Microprocessors and microsystems 68, 17-27, 2019) (Cited by 9)

27. Using modular extension to provably protect ECC against fault attacks (Cryptology ePrint Archive, Report 2015/882, 2015) (Cited by 6)

28. Using modular extension to provably protect Edwards curves against fault attacks (Journal of Cryptographic Engineering 7, 321-330, 2017) (Cited by 5)

29. Time-frequency analysis for second-order attacks (Cryptology ePrint Archive, 2016) (Cited by 5)

30. Private circuits II versus fault injection attacks (2015 International Conference on ReConFigurable Computing and FPGAs . . . , 2015) (Cited by 5)

31. Method taking into account process dispersions to detect hardware trojan horse by side-channel (June, 2015) (Cited by 5)

32. SoK: on DFA vulnerabilities of substitution-permutation networks (Proceedings of the 2019 ACM Asia Conference on Computer and Communications . . . , 2019) (Cited by 4)

33. Correlated extra-reductions defeat blinded regular exponentiation-extended version (Cryptology ePrint Archive, 2016) (Cited by 4)

34. The conflicted usage of RLUTs for security-critical applications on FPGA (Journal of Hardware and Systems Security 2, 162-178, 2018) (Cited by 3)

35. On the cost of asic hardware crackers: A sha-1 case study (Topics in Cryptology–CT-RSA 2021: Cryptographers' Track at the RSA . . . , 2021) (Cited by 2)

36. Crack me if you can: hardware acceleration bridging the gap between practical and theoretical cryptanalysis? a Survey (Proceedings of the 18th International Conference on Embedded Computer . . . , 2018) (Cited by 2)

37. A Generic Countermeasure Against Fault Injection Attacks on Asymmetric Cryptography. (IACR Cryptol. ePrint Arch. 2015, 882, 2015) (Cited by 2)
38. Security, Privacy, and Applied Cryptography Engineering (Springer, Berlin, 2014) (Cited by 1)
39. Software Camouflage (Foundations and Practice of Security: 6th International Symposium, FPS 2013 . . . , 2014) (Cited by 1)
40. Session details: Security threats caused by novel technologies (Proceedings of the 18th International Conference on Embedded Computer . . . , 2018) (Cited by 0)
41. Power and Electromagnetic Analysis for Template Attacks (TRUDEVICE, 2015) (Cited by 0)
42. Lecture Note 10 () (Cited by 0)
43. Using Modular Extension to Provably Protect Edwards Curves Against Fault Attacks () (Cited by 0)
44. Correlated Extra-Reductions Defeat Blinded Regular Exponentiation () (Cited by 0)

References

[1] C. Arthur, "Tech giants may be huge, but nothing matches big data", The Guardian (2013).

[2] Allied Market Research, *World Encryption Software Market - Opportunities and Forecasts, 2014-2022*, https://www.alliedmarketresearch.com/world-encryption-software-market, Accessed on March 5, 2023, 2016.

[3] S. Brannigan, *Secure Gaussian sampling for lattice-based signaturesi, New directions for reaching high standard deviation*, https://pureadmin.qub.ac.uk/ws/portalfiles/portal/258063826/thesis.pdf, Accessed on March 5, 2023, 2021.

[4] M. Rosulek, *Introduction to Modern Cryptography: Cryptography Engineering*, https://web.engr.oregonstate.edu/~rosulekm/crypto/chap6.pdf, Accessed on March 5, 2023, 2016.

[5] D. J. Bernstein, A. Hülsing, and T. Lange, *POST-QUANTUM CRYPTOGRAPHY Integration study*, Technical Report (European Union Agency for Cybersecurity (ENISA), Heraklion, Greece, Oct. 2022).

[6] G. Leurent and C. Pernot, *New Representations of the AES Key Schedule*, Cryptology ePrint Archive, Paper 2020/1253, https://eprint.iacr.org/2020/1253, 2020.

[7] N. Mouha, *Review of the Advanced Encryption Standard*, NIST Interagency/Internal Report (NISTIR) 8319 (National Institute of Standards and Technology (NIST), Gaithersburg, MD, 2022).

[8] A. Adomnicai and T. Peyrin, "Fixslicing AES-like Ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V", IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**, 402–425 (2021).

[9] F. Carter, *The Turing Bombe* (Bletchley Park Trust, 2008).

[10] B. Randell, "Colossus: Godfather of the computer", in *The Origins of Digital Computers* (Springer, 1982), pp. 349–354.

[11] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking ciphers with COPACOBANA–a cost-optimized parallel code breaker", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2006), pp. 101–118.

[12] C. Swenson, *Modern cryptanalysis: techniques for advanced code breaking* (John Wiley & Sons, 2008).

[13] A. Joux, *Algorithmic cryptanalysis* (CRC Press, 2009).

[14]M. Stamp and R. M. Low, *Applied cryptanalysis: breaking ciphers in the real world* (John Wiley & Sons, 2007).
[15].
[16]W. Diffie and M. E. Hellman, "Special feature exhaustive cryptanalysis of the NBS data encryption standard", Computer **10**, 74–84 (1977).
[17]M. Hellman, "A cryptanalytic time-memory trade-off", IEEE transactions on Information Theory **26**, 401–406 (1980).
[18]P. C. Van Oorschot and M. J. Wiener, "Parallel collision search with cryptanalytic applications", Journal of cryptology **12**, 1–28 (1999).
[19]J. M. Pollard, "Monte Carlo methods for index computation", Mathematics of computation **32**, 918–924 (1978).
[20]J. Gilmore, *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, 1998.
[21]T. Güneysu, T. Kasper, M. Novotnỳ, C. Paar, and A. Rupp, "Cryptanalysis with COPACOBANA", IEEE Transactions on Computers **57**, 1498–1513 (2008).
[22]J Keller and B Seitz, "A Hardware-Based Attack on the A5/1 Stream Cipher (2001)", URL http://pv. fernuni-hagen. de/docs/apc2001-final. pdf. Accessed April (2012).
[23]S. Biddle, "NYU ACCIDENTALLY EXPOSED MILITARY CODE-BREAKING COMPUTER PROJECT TO ENTIRE INTERNET", URL https://theintercept.com/2017/05/accidentally-exposed-military-code-breaking-computer-project-to-entire-internet/ (2017).
[24]X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1", in Annual international cryptology conference (Springer, 2005), pp. 17–36.
[25]M. Hassan, A. Khalid, A. Chattopadhyay, C. Rechberger, T. Güneysu, and C. Paar, "New asic/fpga cost estimates for sha-1 collisions", in Digital System Design (DSD), 2015 Euromicro Conference on (IEEE, 2015), pp. 669–676.
[26]M. Stevens, "New collision attacks on SHA-1 based on optimal joint local-collision analysis", in Annual International Conference on the Theory and Applications of Cryptographic Techniques (Springer, 2013), pp. 245–261.
[27]F. Chabaud and A. Joux, "Differential collisions in SHA-0", in Annual International Cryptology Conference (Springer, 1998), pp. 56–71.
[28]E. Biham and R. Chen, "Near-collisions of SHA-0", in Annual International Cryptology Conference (Springer, 2004), pp. 290–305.
[29]N. Pramstaller, C. Rechberger, and V. Rijmen, "Exploiting coding theory for collision attacks on SHA-1", in IMA International Conference on Cryptography and Coding (Springer, 2005), pp. 78–95.
[30]V. Rijmen and E. Oswald, "Update on SHA-1", in Cryptographers' Track at the RSA Conference (Springer, 2005), pp. 58–71.

[31]K. Matusiewicz and J. Pieprzyk, "Finding good differential patterns for attacks on SHA-1", in *Coding and Cryptography* (Springer, 2006), pp. 164–177.

[32]C. S. Jutla and A. C. Patthak, "A Matching Lower Bound on the Minimum Weight of SHA-1 Expansion Code.", IACR Cryptology ePrint Archive **2005**, 266 (2005).

[33]X. Wang, A. C. Yao, and F. Yao, "Cryptanalysis on SHA-1", in Cryptographic Hash Workshop hosted by NIST (2005).

[34]X. Wang, "Cryptanalysis of hash functions and potential dangers", in Invited Talk at the Cryptographer's Track at RSA Conference 2006 (2006).

[35]M. Cochran et al., "Notes on the Wang et al. 263 SHA-1 Differential Path.", IACR Cryptology ePrint Archive **2007**, 474 (2007).

[36]E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby, "Collisions of SHA-0 and Reduced SHA-1", in Annual International Conference on the Theory and Applications of Cryptographic Techniques (Springer, 2005), pp. 36–57.

[37]X. Wang, H. Yu, and Y. L. Yin, "Efficient collision search attacks on SHA-0", in Annual International Cryptology Conference (Springer, 2005), pp. 1–16.

[38]Y. Naito, Y. Sasaki, T. Shimoyama, J. Yajima, N. Kunihiro, and K. Ohta, "Improved collision search for SHA-0", in International Conference on the Theory and Application of Cryptology and Information Security (Springer, 2006), pp. 21–36.

[39]S. Manuel and T. Peyrin, "Collisions on SHA-0 in one hour", in International Workshop on Fast Software Encryption (Springer, 2008), pp. 16–35.

[40]F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen, "The impact of carries on the complexity of collision attacks on SHA-1", in International Workshop on Fast Software Encryption (Springer, 2006), pp. 278–292.

[41]J. Yajima, T. Iwasaki, Y. Naito, Y. Sasaki, T. Shimoyama, N. Kunihiro, and K. Ohta, "A strict evaluation method on the number of conditions for the SHA-1 collision search", in Proceedings of the 2008 ACM symposium on Information, computer and communications security (ACM, 2008), pp. 10–20.

[42]C. De Canniere and C. Rechberger, "Finding SHA-1 characteristics: General results and applications", in International Conference on the Theory and Application of Cryptology and Information Security (Springer, 2006), pp. 1–20.

[43]C. De Canniere, F. Mendel, and C. Rechberger, "Collisions for 70-Step SHA-1: on the full cost of collision search", in International Workshop on Selected Areas in Cryptography (Springer, 2007), pp. 56–73.

[44]S. Manuel, "Classification and generation of disturbance vectors for collision attacks against SHA-1", (2008).
[45]C. McDonald, P. Hawkes, and J. Pieprzyk, "Differential Path for SHA-1 with complexity O (252).", IACR Cryptology ePrint Archive **2009**, 259 (2009).
[46]S. Manuel, "Classification and generation of disturbance vectors for collision attacks against SHA-1", Designs, Codes and Cryptography **59**, 247–263 (2011).
[47]E. A. Grechnikov, "Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics.", IACR Cryptology ePrint Archive **2010**, 413 (2010).
[48]F. Mendel, C. Rechberger, and V. Rijmen, "Update on SHA-1", rump session of CRYPTO **2007** (2007).
[49]R. Chen and E. Biham, "New Techniques for Cryptanalysis of Cryptographic Hash Functions", PhD thesis (Computer Science Department, Technion, 2011).
[50]E. Biham, R. Chen, and A. Joux, "Cryptanalysis of SHA-0 and Reduced SHA-1", Journal of Cryptology **28**, 110–160 (2015).
[51]M. M. J. Stevens et al., *Attacks on hash functions and applications* (Mathematical Institute, Faculty of Science, Leiden University, 2012).
[52]A. Joux and T. Peyrin, "Hash functions and the (amplified) boomerang attack", in Annual International Cryptology Conference (Springer, 2007), pp. 244–263.
[53]T. Peyrin, "Analyse de fonctions de hachage cryptographiques", PhD thesis (PhD thesis, University of Versailles, 2008).
[54]J. Yajima, Y. Sasaki, Y. Naito, T. Iwasaki, T. Shimoyama, N. Kunihiro, and K. Ohta, "A new strategy for finding a differential path of SHA-1", in Australasian Conference on Information Security and Privacy (Springer, 2007), pp. 45–58.
[55]P. Karpman, T. Peyrin, and M. Stevens, "Practical free-start collision attacks on 76-step SHA-1", in Annual Cryptology Conference (Springer, 2015), pp. 623–642.
[56]M. Stevens, P. Karpman, and T. Peyrin, "Freestart collision for full SHA-1", in Annual International Conference on the Theory and Applications of Cryptographic Techniques (Springer, 2016), pp. 459–483.
[57]P. Karpman, "Analyse de primitives symétriques", PhD thesis (Université Paris-Saclay, 2016).
[58]M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1", in Annual International Cryptology Conference (Springer, 2017), pp. 570–596.
[59]E. Tromer and, *Hardware-based cryptanalysis* (2007).

[60]D. Boneh, C. Dunworth, and R. J. Lipton, "Breaking DES Using a Molecular Computer", Proceedings of DIMACS workshop on DNA computing (1995).

[61]V. Suresh, S. Satpathy, and S. Mathew, *Bitcoin mining hardware accelerator with optimized message digest and message scheduler datapath*, US Patent App. 15/274,200, 2018.

[62]N. Marinoff, "Samsung Is Building ASIC Chips for Halong Mining", URL https://bitcoinmagazine.com/articles/samsung-building-asic-chips-halong-mining/ (2018).

[63]M. Santha, "Quantum cryptanalysis: How to break some classical cryptosystems with quantum computers?", in FWS01 Quantum Cryptography (Centre for Quantum Technologies, NUS Singapore and CNRS IRIF, Universite Paris Diderot France, 2018).

[64]B. Lekitsch, S. Weidt, A. G. Fowler, K. Mølmer, S. J. Devitt, C. Wunderlich, and W. K. Hensinger, "Blueprint for a microwave trapped ion quantum computer", in Science Advances, Vol. 3, 2 (American Association for the Advancement of Science, 2017), e1601540.

[65]M. Khairallah, Z. Najm, A. Chattopadhyay, and T. Peyrin, "Crack Me if You Can: Hardware Acceleration Bridging the Gap Between Practical and Theoretical Cryptanalysis?: A Survey", in Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS '18 (2018), pp. 167–172.

[66]E. Wiki, "Ethash", GitHub Ethereum Wiki. https://github. com/ethereum/wiki/wiki/Ethash (2017).

[67]X16R, https://en.bitcoinwiki.org/wiki/X16R.

[68]G. Leurent and T. Peyrin, *SHA-1 is a Shambles - First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust*, Cryptology ePrint Archive, Report 2020/014, https://eprint.iacr.org/2020/014, 2020.

[69]K. Bhargavan and G. Leurent, "Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH", in NDSS 2016 (Feb. 2016).

[70]A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique Cryptanalysis of the Full AES", in Advances in Cryptology – ASIACRYPT 2011, edited by D. H. Lee and X. Wang (2011), pp. 344–371.

[71]R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK Lightweight Block Ciphers", in Proceedings of the 52nd Annual Design Automation Conference, DAC '15 (2015).

[72]Tom Brostöm, *Lightweight Trusted Computing*, https://www.nist.gov/news-events/events/2019/11/lightweight-cryptography-workshop-2019, 2019.

[73]G. Leurent and T. Peyrin, "From Collisions to Chosen-Prefix Collisions Application to Full SHA-1", in Annual International Conference on the

Theory and Applications of Cryptographic Techniques (Springer, 2019), pp. 527–555.

[74]A. Joux and T. Peyrin, "Hash Functions and the (Amplified) Boomerang Attack", in CRYPTO, Vol. 4622, edited by A. Menezes, Lecture Notes in Computer Science (2007), pp. 244–263.

[75]Y.-M. Tu and C.-W. Lu, "The Influence of Lot Size on Production Performance in Wafer Fabrication Based on Simulation", in Procedia Engineering, Vol. 174, 13th Global Congress on Manufacturing and Management Zhengzhou, China 28-30 November, 2016 (2017), pp. 135 –144.

[76]H. Jones, "FINFET AND FD SOI:MARKET AND COST ANALYSIS", FD-SOI Forum 2018. http://soiconsortium.eu/wp-content/uploads/2018/08/MS-FDSOI9.1818-cr.pdf (2018).

[77]globalpetrolprices.com, https://www.globalpetrolprices.com.

[78]D. Vivek, S. Narendra, M. Haycock, V. Govindarajulu, V. Erraguntla, H. Wilson, S. Vangal, A. Pangal, E. Seligman, R. Nair, et al., "1. 1 V 1 GHz communications router with on-chip body bias in 150 nm CMOS", in DIG TECH PAP IEEE INT SOLID STATE CIRCUITS CONF. pp. 270-271+ 466+ 263. 2002 (2002).

[79]V. Rijmen and J. Daemen, "Advanced Encryption Standard", Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology, 19–22 (2001).

[80]Y. Nir and A. Langley, *ChaCha20 and Poly1305 for IETF Protocols*, tech. rep. (2018).

[81]P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis", in Annual International Cryptology Conference (Springer, 1999), pp. 388–397.

[82]N.-F. Standard, "Announcing the Advanced Encryption Standard (AES)", Federal Information Processing Standards Publication **197**, 1–51 (2001).

[83]S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2002), pp. 13–28.

[84]S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens, "Side-channel analysis and machine learning: A practical perspective", in 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017 (2017), pp. 4095–4102.

[85]L. Breiman, "Random Forests", Machine Learning **45**, 5–32 (2001).

[86]F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python", Journal of Machine Learning Research **12**, 2825–2830 (2011).

[87] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES", in Cryptographers' Track at the RSA Conference (Springer, 2006), pp. 1–20.

[88] E. Trichina, "Combinational Logic Design for AES SubByte Transformation on Masked Data", IACR Cryptology ePrint Archive **2003**, 236 (2003).

[89] A. Biryukov, D. Dinu, Y. Le Corre, and A. Udovenko, "Optimal First-Order Boolean Masking for Embedded IoT Devices", in International Conference on Smart Card Research and Advanced Applications (Springer, 2017), pp. 22–41.

[90] J.-S. Coron and A. Tchulkine, "A new algorithm for switching from arithmetic to boolean masking", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2003), pp. 89–97.

[91] J.-S. Coron, J. Großschädl, M. Tibouchi, and P. K. Vadnala, "Conversion from arithmetic to boolean masking with logarithmic complexity", in International Workshop on Fast Software Encryption (Springer, 2015), pp. 130–149.

[92] B. Jungk, R. Petri, and M. Stöttinger, *Efficient Side-Channel Protections of ARX Ciphers*, Cryptology ePrint Archive, Report 2018/693, https://eprint.iacr.org/2018/693, 2018.

[93] P. Schwabe and K. Stoffelen, *All the AES You Need on Cortex-M3 and M4*, Cryptology ePrint Archive, Report 2016/714, https://eprint.iacr.org/2016/714, 2016.

[94] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis", in CRYPTO, Vol. 1666, edited by M. J. Wiener, Lecture Notes in Computer Science (1999), pp. 388–397.

[95] L. Goubin and J. Patarin, "DES and Differential Power Analysis. The "Duplication" Method", in CHES, LNCS, Worcester, MA, USA (1999), pp. 158–172.

[96] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks", in CRYPTO, Vol. 1666, LNCS, Santa Barbara, CA, USA. ISBN: 3-540-66347-9 (1999).

[97] T. S. Messerges, "Using second-Order Power Analysis to Attack DPA resistant Software", in CHES, Vol. 1965, LNCS, Worcester, MA, USA (2000), pp. 71–77.

[98] J. Waddle and D. Wagner, "Towards Efficient Second-Order Power Analysis", in CHES, Vol. 3156, LNCS, Cambridge, MA, USA (2004), pp. 1–15.

[99] E. Prouff, M. Rivain, and R. Bevan, "Statistical Analysis of Second Order Differential Power Analysis", IEEE Trans. Computers **58**, 799–811 (2009).

[100] H. Maghrebi, E. Prouff, S. Guilley, and J.-L. Danger, *A First-Order Leak-Free Masking Countermeasure*, Cryptology ePrint Archive, Report 2012/028, http://eprint.iacr.org/2012/028, 2012.

[101] J. Blömer, J. Guajardo, and V. Krummel, "Provably Secure Masking of AES", in Selected Areas in Cryptography, Vol. 3357, edited by H. Handschuh and M. A. Hasan, Lecture Notes in Computer Science (2004), pp. 69–83.

[102] M. Rivain and E. Prouff, "Provably Secure Higher-Order Masking of AES", in CHES, Vol. 6225, edited by S. Mangard and F.-X. Standaert, LNCS (2010), pp. 413–427.

[103] E. Prouff and M. Rivain, "A Generic Method for Secure SBox Implementation", in WISA, Vol. 4867, edited by S. Kim, M. Yung, and H.-W. Lee, Lecture Notes in Computer Science (2007), pp. 227–244.

[104] F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard, "The World is Not Enough: Another Look on Second-Order DPA", in ASIACRYPT, Vol. 6477, LNCS, Singapore. http://www.dice.ucl.ac.be/~fstandae/PUBLIS/88.pdf (2010), pp. 112–129.

[105] T. S. Messerges, "Securing the AES Finalists Against Power Analysis Attacks", in Fast Software Encryption'00, New York (2000), pp. 150–164.

[106] M.-L. Akkar and C. Giraud, "An Implementation of DES and AES Secure against Some Attacks", in Proceedings of CHES'01, Vol. 2162, edited by LNCS, LNCS, Paris, France (2001), pp. 309–318.

[107] J. Coron, "Higher Order Masking of Look-Up Tables", in Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings, Vol. 8441, edited by P. Q. Nguyen and E. Oswald, Lecture Notes in Computer Science (2014), pp. 441–458.

[108] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis", in Proceedings of CRYPTO'99, Vol. 1666, LNCS (1999), pp. 388–397.

[109] J. Pan, J. I. den Hartog, and J. Lu, "You Cannot Hide behind the Mask: Power Analysis on a Provably Secure *S*-Box Implementation", in WISA, Vol. 5932, edited by H. Y. Youm and M. Yung, Lecture Notes in Computer Science (2009), pp. 178–192.

[110] M. Tunstall, C. Whitnall, and E. Oswald, "Masking Tables - An Underestimated Security Risk", in Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers, Vol. 8424, edited by S. Moriai, Lecture Notes in Computer Science (2013), pp. 425–444.

[111] É. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model", in CHES, Vol. 3156, LNCS, Cambridge, MA, USA (2004), pp. 16–29.

[112] A. DeTrano, S. Guilley, X. Guo, N. Karimi, and R. Karri, "Exploiting Small Leakages in Masks to Turn a Second-order Attack into a First-order At-

tack", in Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy, HASP '15 (2015), 7:1–7:5.

[113]T. S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software", in CHES, Vol. 1965, LNCS, Worcester, MA, USA (2000), pp. 238–251.

[114]E. Oswald and S. Mangard, "Template Attacks on Masking — Resistance Is Futile", in CT-RSA, Vol. 4377, edited by M. Abe, Lecture Notes in Computer Science (2007), pp. 243–256.

[115]C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil, "Improved Collision-Correlation Power Analysis on First Order Protected AES", in CHES, Vol. 6917, edited by B. Preneel and T. Takagi, LNCS (2011), pp. 49–62.

[116]N. Bruneau, S. Guilley, A. Heuser, and O. Rioul, "Masks Will Fall Off – Higher-Order Optimal Distinguishers", in Advances in Cryptology – ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II, Vol. 8874, edited by P. Sarkar and T. Iwata, Lecture Notes in Computer Science (2014), pp. 344–365.

[117]A. A. Ding, L. Zhang, Y. Fei, and P. Luo, "A Statistical Model for Higher Order DPA on Masked Devices", in Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, Vol. 8731, edited by L. Batina and M. Robshaw, Lecture Notes in Computer Science (2014), pp. 147–169.

[118]Y. Fei, Q. Luo, and A. A. Ding, "A Statistical Model for DPA with Novel Algorithmic Confusion Analysis", in CHES, Vol. 7428, edited by E. Prouff and P. Schaumont, LNCS (2012), pp. 233–250.

[119]Y. Fei, A. A. Ding, J. Lao, and L. Zhang, "A statistics-based success rate model for DPA and CPA", J. Cryptographic Engineering **5**, 227–243 (2015).

[120]S. Guilley, A. Heuser, and O. Rioul, "A Key to Success - Success Exponents for Side-Channel Distinguishers", in Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings, Vol. 9462, edited by A. Biryukov and V. Goyal, Lecture Notes in Computer Science (2015), pp. 270–290.

[121]S. Guilley, A. Heuser, and O. Rioul, *A Key to Success – Success Exponents for Side-Channel Distinguishers (extended version of [120])*, Cryptology ePrint Archive, Report 2016/987, `http://eprint.iacr.org/2016/987`, 2016.

[122]C. Whitnall and E. Oswald, "A Fair Evaluation Framework for Comparing Side-Channel Distinguishers", J. Cryptographic Engineering **1**, 145–160 (2011).

[123]K. Schramm and C. Paar, "Higher Order Masking of the AES", in CT-RSA, Vol. 3860, edited by D. Pointcheval, LNCS (2006), pp. 208–225.

[124]J.-S. Coron, E. Prouff, and M. Rivain, "Side Channel Cryptanalysis of a Higher Order Masking Scheme", in CHES, Vol. 4727, edited by P. Paillier and I. Verbauwhede, LNCS (2007), pp. 28–44.

[125]M. Rivain, E. Prouff, and J. Doget, "Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers", in CHES, Vol. 5747, Lecture Notes in Computer Science, Lausanne, Switzerland (2009), pp. 171–188.

[126]U. Datta and A. Muktibodh, *Algebra And Trigonometry* (Prentice-Hall Of India Pvt. Limited, 2006).

[127]P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", in Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings, Vol. 1109, edited by N. Koblitz, Lecture Notes in Computer Science (1996), pp. 104–113.

[128]J.-L. Danger, S. Guilley, P. Hoogvorst, C. Murdica, and D. Naccache, "A synthesis of side-channel attacks on elliptic curve cryptography in smartcards", J. Cryptographic Engineering **3**, 241–265 (2013).

[129]J. Heyszl, A. Ibing, S. Mangard, F. D. Santis, and G. Sigl, "Clustering Algorithms for Non-profiled Single-Execution Attacks on Exponentiations", in Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers, Vol. 8419, edited by A. Francillon and P. Rohatgi, Lecture Notes in Computer Science (2013), pp. 79–93.

[130]R. Specht, J. Heyszl, M. Kleinsteuber, and G. Sigl, "Improving Non-profiled Attacks on Exponentiations Based on Clustering and Extracting Leakage from Multi-channel High-Resolution EM Measurements", in Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers, Vol. 9064, edited by S. Mangard and A. Y. Poschmann, Lecture Notes in Computer Science (2015), pp. 3–19.

[131]E. Nascimento and L. Chmielewski, "Applying Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations", in Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers, Vol. 10728, edited by T. Eisenbarth and Y. Teglia, Lecture Notes in Computer Science (2017), pp. 213–231.

[132]G. Perin and L. Chmielewski, "A Semi-Parametric Approach for Side-Channel Attacks on Protected RSA Implementations", in Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015,

Bochum, Germany, November 4-6, 2015. Revised Selected Papers, Vol. 9514, edited by N. Homma and M. Medwed, Lecture Notes in Computer Science (2015), pp. 34–53.

[133]G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, *A testing methodology for side-channel resistance validation*, 2011.

[134]S. P. Lloyd, "Least squares quantization in PCM", IEEE Trans. Information Theory **28**, 129–136 (1982).

[135]B. L. Welch, "The Generalization of 'Student's' Problem when Several Different Population Variances are Involved", Biometrika **34**, 28–35 (1947).

[136]S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, "NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage", IACR Cryptology ePrint Archive **2013**, 717 (2013).

[137]C. Clavier and L. Reynaud, "Improved Blind Side-Channel Analysis by Exploitation of Joint Distributions of Leakages", in Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, Vol. 10529, edited by W. Fischer and N. Homma, Lecture Notes in Computer Science (2017), pp. 24–44.

[138]D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-speed high-security signatures", J. Cryptographic Engineering **2**, 77–89 (2012).

[139]M. Joye, "Highly Regular Right-to-Left Algorithms for Scalar Multiplication", in Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings, Vol. 4727, edited by P. Paillier and I. Verbauwhede, LNCS (2007), pp. 135–147.

[140]H. Hisil, K. K. Wong, G. Carter, and E. Dawson, "Twisted Edwards Curves Revisited", in Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings, Vol. 5350, edited by J. Pieprzyk, Lecture Notes in Computer Science (2008), pp. 326–343.

[141]M. Hutter and P. Schwabe, "NaCl on 8-Bit AVR Microcontrollers", in Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings, Vol. 7918, edited by A. Youssef, A. Nitaj, and A. E. Hassanien, Lecture Notes in Computer Science (2013), pp. 156–172.

[142]P. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis", J. Comput. Appl. Math. **20**, 53–65 (1987).

[143]D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure", IEEE Trans. Pattern Anal. Mach. Intell. **1**, 224–227 (1979).

[144]E. Cagli, C. Dumas, and E. Prouff, "Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing", in Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, Vol. 10529, edited by W. Fischer and N. Homma, Lecture Notes in Computer Science (2017), pp. 45–68.

[145]J. Schmidhuber, "Deep learning in neural networks: An overview", Neural Networks **61**, 85–117 (2015).

[146]M. Khairallah, R. Sadhukhan, R. Samanta, J. Breier, S. Bhasin, R. S. Chakraborty, A. Chattopadhyay, and D. Mukhopadhyay, "DFARPA: Differential fault attack resistant physical design automation", in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018 (IEEE, 2018), pp. 1171–1174.

[147]W. He, J. Breier, and S. Bhasin, "Cheap and cheerful: A low-cost digital sensor for detecting laser fault injection attacks", in International Conference on Security, Privacy, and Applied Cryptography Engineering (Springer, 2016), pp. 27–46.

[148]C. Patrick, B. Yuce, N. F. Ghalaty, and P. Schaumont, "Lightweight fault attack resistance in software using intra-instruction redundancy", in International Conference on Selected Areas in Cryptography (Springer, 2016), pp. 231–244.

[149]A. Baksi, S. Bhasin, J. Breier, M. Khairallah, and T. Peyrin, "Protecting block ciphers against differential fault attacks without re-keying", in 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) (IEEE, 2018), pp. 191–194.

[150]M. Medwed, F.-X. Standaert, J. Großschädl, and F. Regazzoni, "Fresh rekeying: Security against side-channel and fault attacks for low-cost devices", in International Conference on Cryptology in Africa (Springer, 2010), pp. 279–296.

[151]C. Dobraunig, F. Koeune, S. Mangard, F. Mendel, and F.-X. Standaert, "Towards fresh and hybrid re-keying schemes with beyond birthday security", in International Conference on Smart Card Research and Advanced Applications (Springer, 2015), pp. 225–241.

[152]C. Dobraunig, M. Eichlseder, S. Mangard, and F. Mendel, "On the security of fresh re-keying to counteract side-channel and fault attacks", in International Conference on Smart Card Research and Advanced Applications (Springer, 2014), pp. 233–244.

[153]G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD", in International workshop on cryptographic hardware and embedded systems (Springer, 2003), pp. 77–88.

[154]K. Sakiyama, Y. Li, M. Iwamoto, and K. Ohta, "Information-theoretic approach to optimal differential fault analysis", IEEE Transactions on Information Forensics and Security **7**, 109–120 (2012).

[155]M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault", in IFIP international workshop on information security theory and practices (Springer, 2011), pp. 224–233.

[156]K. Jeong and C. Lee, "Differential fault analysis on block cipher LED-64", in *Future Information Technology, Application, and Service* (Springer, 2012), pp. 747–755.

[157]N. Vafaei, N. Bagheri, S. Saha, and D. Mukhopadhyay, "Differential Fault Attack on SKINNY Block Cipher", in International Conference on Security, Privacy, and Applied Cryptography Engineering (Springer, 2018), pp. 177–197.

[158]L. Song and L. Hu, "Differential fault attack on the PRINCE block cipher", in International Workshop on Lightweight Cryptography for Security and Privacy (Springer, 2013), pp. 43–54.

[159]J. Breier and W. He, "Multiple fault attack on present with a hardware trojan implementation in FPGA", in Secure Internet of Things (SIoT), 2015 International Workshop on (IEEE, 2015), pp. 58–64.

[160]S. Patranabis, J. Breier, D. Mukhopadhyay, and S. Bhasin, "One plus one is more than two: a practical combination of power and fault analysis attacks on PRESENT and PRESENT-like block ciphers", in Fault Diagnosis and Tolerance in Cryptography (FDTC), 2017 Workshop on (IEEE, 2017), pp. 25–32.

[161]B. Lac, M. Beunardeau, A. Canteaut, J. J. Fournier, and R. Sirdey, "A First DFA on PRIDE: from Theory to Practice", in International Conference on Risks and Security of Internet and Systems (Springer, 2016), pp. 214–238.

[162]E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems", in Annual international cryptology conference (Springer, 1997), pp. 513–525.

[163]P. Dusart, G. Letourneux, and O. Vivolo, "Differential fault analysis on AES", in International Conference on Applied Cryptography and Network Security (Springer, 2003), pp. 293–306.

[164]C. Giraud, "DFA on AES", in International Conference on Advanced Encryption Standard (Springer, 2004), pp. 27–41.

[165]G. Wang and S. Wang, "Differential Fault Analysis on PRESENT Key Schedule", in Computational Intelligence and Security (CIS), 2010 International Conference on (2010), pp. 362–366.

[166]F. De Santis, O. M. Guillen, E. Sakic, and G. Sigl, "Ciphertext-only fault attacks on PRESENT", in International Workshop on Lightweight Cryptography for Security and Privacy (Springer, 2014), pp. 85–108.

[167]B. Lac, A. Canteaut, J. Fournier, and R. Sirdey, "DFA on LS-Designs with a Practical Implementation on SCREAM", in International Workshop on Constructive Side-Channel Analysis and Secure Design (Springer, 2017), pp. 223–247.

[168]L. Hemme, "A differential fault attack against early rounds of (triple-) DES", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2004), pp. 254–267.

[169]H. Tupsamudre, S. Bisht, and D. Mukhopadhyay, "Differential fault analysis on the families of SIMON and SPECK ciphers", in 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC) (IEEE, 2014), pp. 40–48.

[170]N. Bagheri, R. Ebrahimpour, and N. Ghaedi, "New differential fault analysis on PRESENT", EURASIP Journal on Advances in Signal Processing **2013**, 145 (2013).

[171]M. Agoyan, J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, and A. Tria, "How to flip a bit?", in On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International (IEEE, 2010), pp. 235–239.

[172]D. Saha, D. Mukhopadhyay, and D. R. Chowdhury, "A Diagonal Fault Attack on the Advanced Encryption Standard", IACR Cryptology ePrint Archive **2009** (2009).

[173]V. Grosso, G. Leurent, F.-X. Standaert, and K. Varıcı, "LS-designs: Bitslice encryption for efficient masked software implementations", in International Workshop on Fast Software Encryption (Springer, 2014), pp. 18–37.

[174]S. Halevi, D. Coppersmith, and C. Jutla, "SCREAM: A software-efficient stream cipher", in International Workshop on Fast Software Encryption (Springer, 2002), pp. 195–209.

[175]C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS", in Annual Cryptology Conference (Springer, 2016), pp. 123–153.

[176]J. Jean, I. Nikolić, and T. Peyrin, "Tweaks and keys for block ciphers: the TWEAKEY framework", in International Conference on the Theory and Application of Cryptology and Information Security (Springer, 2014), pp. 274–288.

[177]A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2007), pp. 450–466.

[178]M. Wang, "Differential cryptanalysis of reduced-round PRESENT", in International Conference on Cryptology in Africa (Springer, 2008), pp. 40–49.

[179]T. Reis, D. Aranha, and J. López, "PRESENT Runs Fast: Efficient and Secure Implementation in Software", in Conference on Cryptographic Hardware and Embedded Systems (CHES'17) (2017).

[180]N. F. Pub, "197: Advanced encryption standard (AES)", Federal information processing standards publication **197**, 0311 (2001).

[181]J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard* (Springer Science & Business Media, 2013).

[182]G. Liu, M. Ghosh, and L. Song, "Security analysis of SKINNY under related-tweakey settings", IACR Transactions on Symmetric Cryptology **2017**, 37–72 (2017).

[183]C. Chen, T. Eisenbarth, A. Shahverdi, and X. Ye, "Balanced Encoding to Mitigate Power Analysis: A Case Study", in CARDIS, Vol. 8968, Lecture Notes in Computer Science, Paris, France (2014).

[184]P. Rauzy, S. Guilley, and Z. Najm, "Formally proved security of assembly code against power analysis", English, Journal of Cryptographic Engineering, 1–16 (2015).

[185]"The Other Side of The Coin: Analyzing Software Encoding Schemes Against Fault Injection Attacks", in CARDIS (2015).

[186]Y. Ishai, A. Sahai, and D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", in CRYPTO, Vol. 2729, Lecture Notes in Computer Science, Santa Barbara, California, USA (2003), pp. 463–481.

[187]Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner, "Private Circuits II: Keeping Secrets in Tamperable Circuits", in EUROCRYPT, Vol. 4004, Lecture Notes in Computer Science, St. Petersburg, Russia (2006), pp. 308–327.

[188]S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, ISBN 0-387-30857-1, http://www.dpabook.org/ (Springer, 2006), p. 338.

[189]C. Helfmeier, D. Nedospasov, C. Tarnovsky, J. S. Krissler, C. Boit, and J.-P. Seifert, "Breaking and entering through the silicon", in ACM Conference on Computer and Communications Security, edited by A.-R. Sadeghi, V. D. Gligor, and M. Yung (2013), pp. 733–744.

[190]M. Gomathisankaran and A. Tyagi, "Glitch Resistant Private Circuits Design Using HORNS", in IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2014, Tampa, FL, USA, July 9-11, 2014 (2014), pp. 522–527.

[191]Y. Yu, J. Leiwo, and B. Premkumar, "Private stateful circuits secure against probing attacks", in ASIACCS, Singapore (2007), pp. 63–69.

[192]J. Park and A. Tyagi, "*t*-Private logic synthesis on FPGAs", in Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on (2012), pp. 63–68.

[193]J. Park and A. Tyagi, "*t*-Private Systems: Unified Private Memories and Computation", English, in Security, Privacy, and Applied Cryptography Engineering, Vol. 8804, edited by R. Chakraborty, V. Matyas, and P. Schaumont, Lecture Notes in Computer Science (2014), pp. 285–302.

[194]J. Park and A. Tyagi, "Towards Making Private Circuits Practical: DPA Resistant Private Circuits", in IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2014, Tampa, FL, USA, July 9-11, 2014 (2014), pp. 528–533.

[195]S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches", in ICICS, Vol. 4307, LNCS, Raleigh, NC, USA (2006), pp. 529–545.

[196]O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating Masking Schemes", in Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I, Vol. 9215, edited by R. Gennaro and M. Robshaw, Lecture Notes in Computer Science (2015), pp. 764–783.

[197]H. Kim, S. Hong, and J. Lim, "A Fast and Provably Secure Higher-Order Masking of AES S-Box", in CHES, Vol. 6917, edited by B. Preneel and T. Takagi, LNCS (2011), pp. 95–107.

[198]J. Coron, J. Großschädl, and P. K. Vadnala, "Secure Conversion between Boolean and Arithmetic Masking of Any Order", in Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, Vol. 8731, edited by L. Batina and M. Robshaw, Lecture Notes in Computer Science (2014), pp. 188–205.

[199]J. Danger, S. Guilley, P. Hoogvorst, C. Murdica, and D. Naccache, "Low-Cost Countermeasure against RPA", in Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers, Vol. 7771, edited by S. Mangard, Lecture Notes in Computer Science (2012), pp. 106–122.

[200]Xilinx, *Virtex-6 Libraries Guide for HDL Designs (UG623, v 13.1)*, Primitive: Six-input, 2-output, Look-Up Table, `http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/virtex6_hdl.pdf`, 2011.

[201]S. Bhasin, S. Guilley, F. Flament, N. Selmane, and J.-L. Danger, "Countering Early Evaluation: An Approach Towards Robust Dual-Rail Precharge Logic", in WESS, Scottsdale, Arizona, USA. DOI: 10.1145/1873548.1873554 (2010), 6:1–6:8.

[202] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, *The SIMON and SPECK Families of Lightweight Block Ciphers*, Cryptology ePrint Archive, Report 2013/404, http://eprint.iacr.org/2013/404, 2013.

[203] S. Bhasin, T. Graba, J. Danger, and Z. Najm, "A look into SIMON from a side-channel perspective", in 2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014, Arlington, VA, USA, May 6-7, 2014 (2014), pp. 56–59.

[204] L. Sauvage, M. Nassar, S. Guilley, F. Flament, J.-L. Danger, and Y. Mathieu, "Exploiting Dual-Output Programmable Blocks to Balance Secure Dual-Rail Logics", International Journal of Reconfigurable Computing (IJRC), edited by Hindawi, DOI: 10.1155/2010/375245, 12 (2010).

[205] W. He, A. Otero, E. de la Torre, and T. Riesgo, "Customized and Automated Routing Repair Toolset Towards Side-Channel Analysis Resistant Dual Rail Logic", Microprocessors and Microsystems, edited by Elsevier, DOI: 10.1016/j.micpro.2014.02.005 (2014).

[206] M. G. Karpovsky, K. J. Kulikowski, and A. Taubin, "Robust Protection against Fault Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard", in DSN, Florence, Italy (2004), pp. 93–101.

[207] A. Moradi and V. Immler, "Early Propagation and Imbalanced Routing, How to Diminish in FPGAs", IACR Cryptol. ePrint Arch., 454 (2014).

[208] G. Piret and J.-J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad", in CHES, Vol. 2779, LNCS, Cologne, Germany (2003), pp. 77–88.

[209] N. Selmane, S. Guilley, and J.-L. Danger, "Setup Time Violation Attacks on AES", in EDCC, The seventh European Dependable Computing Conference, ISBN: 978-0-7695-3138-0, DOI: 10.1109/EDCC-7.2008.11 (2008), pp. 91–96.

[210] S. Guilley, S. Chaudhuri, L. Sauvage, T. Graba, J.-L. Danger, P. Hoogvorst, V.-N. Vong, M. Nassar, and F. Flament, "Shall we trust WDDL?", in Future of Trust in Computing, Vol. 2, edited by Vieweg+Teubner, DOI: 10.1007/978-3-8348-9324-6_22. Berlin, Germany (2008), pp. 208–215.

[211] A. Boscher and H. Handschuh, "Masking Does Not Protect Against Differential Fault Attacks", in FDTC, 5th Workshop on Fault Detection and Tolerance in Cryptography, IEEE-CS, DOI: 10.1109/FDTC.2008.12, Washington, DC, USA (2008), pp. 35–40.

[212] Y. Monnet, M. Renaudin, R. Leveugle, C. Clavier, and P. Moitrel, "Case Study of a Fault Attack on Asynchronous DES Crypto-Processors", in FDTC, Vol. 4236, Lecture Notes in Computer Science, Yokohama, Japan (2006), pp. 88–97.

[213] A. Dehbaoui, A. Mirbaha, N. Moro, J. Dutertre, and A. Tria, "Electromagnetic Glitch on the AES Round Counter", in Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers, Vol. 7864, edited by E. Prouff, Lecture Notes in Computer Science (2013), pp. 17–31.

[214] S. Ordas, L. Guillaume-Sage, K. Tobich, J. Dutertre, and P. Maurine, "Evidence of a Larger EM-Induced Fault Model", in Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers, Vol. 8968, edited by M. Joye and A. Moradi, Lecture Notes in Computer Science (2014), pp. 245–259.

[215] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta, "Fault Sensitivity Analysis", in CHES, Vol. 6225, Lecture Notes in Computer Science, Santa Barbara, CA, USA (2010), pp. 320–334.

[216] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)", English, International Journal of Information Security **1**, 36–63 (2001).

[217] M. Naehrig, R. Niederhagen, and P. Schwabe, "New software speed records for cryptographic pairings", in Progress in Cryptology – LATINCRYPT 2010, Vol. 6212, edited by M. Abdalla and P. S. Barreto, Lecture Notes in Computer Science, Updated version: `http://cryptojedi.org/papers/#dclxvi` (2010), pp. 109–123.

[218] A. Guillevic and D. Vergnaud, "Genus 2 Hyperelliptic Curve Families with Explicit Jacobian Order Evaluation and Pairing-Friendly Constructions", English, in *Pairing-Based Cryptography — Pairing 2012*, Vol. 7708, edited by M. Abdalla and T. Lange, Lecture Notes in Computer Science (Springer Berlin Heidelberg, 2013), pp. 234–253.

[219] M. Joye and M. Tunstall, eds., *Fault Analysis in Cryptography*, Information Security and Cryptography (Springer, 2012).

[220] I. Biehl, B. Meyer, and V. Müller, "Differential Fault Attacks on Elliptic Curve Cryptosystems", in CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology (2000), pp. 131–146.

[221] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)", in EUROCRYPT, Vol. 1233, edited by W. Fumy, Lecture Notes in Computer Science (1997), pp. 37–51.

[222] J. Blömer, M. Otto, and J.-P. Seifert, "Sign Change Fault Attacks on Elliptic Curve Cryptosystems", English, in *Fault Diagnosis and Tolerance in Cryptography*, Vol. 4236, edited by L. Breveglieri, I. Koren, D. Naccache, and

J.-P. Seifert, Lecture Notes in Computer Science (Springer Berlin Heidelberg, 2006), pp. 36–52.

[223]G. Barthe, F. Dupressoir, P. Fouque, B. Grégoire, and J. Zapalowicz, "Synthesis of Fault Attacks on Cryptographic Implementations", in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, edited by G. Ahn, M. Yung, and N. Li (2014), pp. 1016–1027.

[224]R. Lashermes, M. Paindavoine, N. El Mrabet, J. J. Fournier, and L. Goubin, "Practical Validation of Several Fault Attacks against the Miller Algorithm", in Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on, Busan, Korea (2014), pp. 115–122.

[225]J. Blömer, R. Gomes Da Silva, P. Gunther, J. Krämer, and J.-P. Seifert, "A Practical Second-Order Fault Attack against a Real-World Pairing Implementation", in Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on, Busan, Korea (2014), pp. 123–136.

[226]J. Blömer, P. Günther, and G. Liske, "Tampering Attacks in Pairing-Based Cryptography", in Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on, Busan, Korea (2014), pp. 1–7.

[227]N. El Mrabet, J. J. Fournier, L. Goubin, and R. Lashermes, "A survey of fault attacks in pairing based cryptography", English, Cryptography and Communications, 1–21 (2014).

[228]D. Wagner, "Cryptanalysis of a provably secure CRT-RSA algorithm", in ACM Conference on Computer and Communications Security, edited by V. Atluri, B. Pfitzmann, and P. D. McDaniel (2004), pp. 92–97.

[229]C. Giraud, "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis", IEEE Trans. Computers **55**, 1116–1120 (2006).

[230]A. Boscher, R. Naciri, and E. Prouff, "CRT RSA Algorithm Protected Against Fault Attacks", in WISTP, Vol. 4462, edited by D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, Lecture Notes in Computer Science (2007), pp. 229–243.

[231]D.-P. Le, M. Rivain, and C. H. Tan, "On Double Exponentiation for Securing RSA against Fault Analysis", in CT-RSA, Vol. 8366, edited by J. Benaloh, Lecture Notes in Computer Science (2014), pp. 152–168.

[232]S.-K. Kim, T. H. Kim, D.-G. Han, and S. Hong, "An efficient CRT-RSA algorithm secure against power and fault attacks", J. Syst. Softw. **84**, 1660–1669 (2011).

[233]D. Karaklajic, J. Fan, J. Schmidt, and I. Verbauwhede, "Low-cost fault detection method for ECC using Montgomery powering ladder", in Design, Automation and Test in Europe, DATE 2011, Grenoble, France, March 14-18, 2011 (2011), pp. 1016–1021.

[234]A. Shamir, *Method and apparatus for protecting public key schemes from timing and fault attacks*, Patent Number 5,991,415; also presented at the rump session of EUROCRYPT '97, 1999.

[235]C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures", in CHES, Vol. 2523, edited by B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, Lecture Notes in Computer Science (2002), pp. 260–275.

[236]D. Vigilant, "RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks", in CHES, Vol. 5154, edited by E. Oswald and P. Rohatgi, Lecture Notes in Computer Science (2008), pp. 130–145.

[237]M. Joye, P. Paillier, and S.-M. Yen, "Secure evaluation of modular functions", in International Workshop on Cryptology and Network Security, edited by R. Hwang and C. Wu, http://joye.site88.net/papers/JPY01dfa.pdf, Taipei, Taiwan (2001), pp. 227–229.

[238]J. Blömer, M. Otto, and J.-P. Seifert, "A new CRT-RSA algorithm secure against bellcore attacks", in ACM Conference on Computer and Communications Security, edited by S. Jajodia, V. Atluri, and T. Jaeger (2003), pp. 311–320.

[239]M. Ciet and M. Joye, "Practical Fault Countermeasures for Chinese Remaindering Based RSA", in Fault Diagnosis and Tolerance in Cryptography (2005).

[240]E. Dottax, C. Giraud, M. Rivain, and Y. Sierra, "On Second-Order Fault Analysis Resistance for CRT-RSA Implementations", in WISTP, Vol. 5746, edited by O. Markowitch, A. Bilas, J.-H. Hoepman, C. J. Mitchell, and J.-J. Quisquater, Lecture Notes in Computer Science (2009), pp. 68–83.

[241]Y.-J. Baek and I. Vasyltsov, "How to Prevent DPA and Fault Attack in a Unified Way for ECC Scalar Multiplication - Ring Extension Method", English, in *Information Security Practice and Experience*, Vol. 4464, edited by E. Dawson and D. S. Wong, Lecture Notes in Computer Science (Springer Berlin Heidelberg, 2007), pp. 225–237.

[242]M. Joye, *Fault-resistant calculations on elliptic curves*, EP Patent App. EP20,100,155,001 ; http://www.google.com/patents/EP2228716A1?cl=en, 2010.

[243]D. J. Bernstein, T. Lange, and P. Schwabe, "The Security Impact of a New Cryptographic Library", in Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings, Vol. 7533, edited by A. Hevia and G. Neven, Lecture Notes in Computer Science (2012), pp. 159–176.

[244]C. Clavier, "Secret External Encodings Do Not Prevent Transient Fault Analysis", in CHES, Vol. 4727, Lecture Notes in Computer Science, Vienna, Austria (2007), pp. 181–194.

[245]N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson, "Formal verification of a software countermeasure against instruction skip attacks", J. Cryptographic Engineering **4**, 145–156 (2014).

[246]V. Leont'ev, "Roots of random polynomials over a finite field", English, Mathematical Notes **80**, 300–304 (2006).

[247]M. Dugardin, S. Guilley, M. Moreau, Z. Najm, and P. Rauzy, *Using Modular Extension to Provably Protect Edwards Curves Against Fault Attacks*, Cryptology ePrint Archive, Report 2015/882, http://eprint.iacr.org/2015/882, 2015.

[248]H. M. Edwards, "A normal form for elliptic curves", Bulletin of the American Mathematical Society **44**, 393–422 (2007).

[249]D. J. Bernstein and T. Lange, "Faster Addition and Doubling on Elliptic Curves", in Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings, Vol. 4833, edited by K. Kurosawa, Lecture Notes in Computer Science (2007), pp. 29–50.

[250]D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards Curves", in Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings, Vol. 5023, edited by S. Vaudenay, Lecture Notes in Computer Science (2008), pp. 389–405.

[251]D. J. Bernstein and T. Lange, *Explicit-Formulas Database*, http://hyperelliptic.org/EFD/, 2015.

[252]A. Battistello, "Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers", in (Springer International Publishing, Cham, 2014) Chap. Common Points on Elliptic Curves: The Achilles' Heel of Fault Attack Countermeasures, pp. 69–81.

[253]D. Moody and D. Shumow, "Isogenies on Edwards and Huff curves", Computer Security Division, National Institute of Standards and Technology (NIST) (2011).

[254]University of Sydney, *Magma Computational Algebra System*, http://magma.maths.usyd.edu.au/magma/.

[255]P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis", in, CRYPTO '99 (1999), pp. 388–397.

[256]F. Poucheret, K. Tobich, M. Lisart, L. Chusseau, B. Robisson, and P. Maurine, "Local and Direct EM Injection of Power Into CMOS Integrated Circuits", in 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011 (2011), pp. 100–104.

[257]L. Zussa, A. Dehbaoui, K. Tobich, J. Dutertre, P. Maurine, L. Guillaume-Sage, J. Clédière, and A. Tria, "Efficiency of a glitch detector against electromagnetic fault injection", in Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014 (2014), pp. 1–6.

[258]N. Miura, D. Fujimoto, M. Nagata, N. Homma, Y. Hayashi, and T. Aoki, "EM attack sensor: concept, circuit, and design-automation methodology", in Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015 (2015), 176:1–176:6.

[259]C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapid prototyping tools for FPGA designs: RapidSmith", in Field-Programmable Technology (FPT), 2010 International Conference on (IEEE, 2010), pp. 353–356.

[260]W. He, A. Otero, E. de la Torre, and T. Riesgo, "Customized and automated routing repair toolset towards side-channel analysis resistant dual rail logic", Microprocessors and Microsystems **38**, 899–910 (2014).

[261]S. Bhasin, J. Danger, S. Guilley, Z. Najm, and X. T. Ngo, "Linear Complementary Dual Code Improvement to Strengthen Encoded Circuit Against Hardware Trojan Horses", in 2015 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2015, McLean, VA, USA (2015).

[262]S. Trimberger and J. Moore, "FPGA Security: Motivations, Features, and Applications", Proceedings of the IEEE **102**, 1248–1265 (2014).

[263]T. Güneysu and A. Moradi, "Generic Side-Channel Countermeasures for Reconfigurable Devices", in CHES, Vol. 6917, edited by B. Preneel and T. Takagi, LNCS (2011), pp. 33–48.

[264]S. Bhasin, W. He, S. Guilley, and J.-L. Danger, "Exploiting FPGA block memories for protected cryptographic implementations", in ReCoSoC (2013), pp. 1–8.

[265]T. Güneysu and C. Paar, "Ultra High Performance ECC over NIST Primes on Commercial FPGAs", in CHES (2008), pp. 62–78.

[266]D. B. Roy, D. Mukhopadhyay, M. Izumi, and J. Takahashi, "Tile Before Multiplication: An Efficient Strategy to Optimize DSP Multiplier for Accelerating Prime Field ECC for NIST Curves", in The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014 (2014), pp. 1–6.

[267]T. Güneysu, *Getting Post-Quantum Crypto Algorithms Ready for Deployment*.

[268]W. He, A. Otero, E. de la Torre, and T. Riesgo, "Automatic generation of identical routing pairs for FPGA implemented DPL logic", in ReConFig (2012), pp. 1–6.

[269] M. Kumm, K. Möller, and P. Zipf, "Reconfigurable FIR filter using distributed arithmetic on FPGAs", in 2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), Beijing, China, May 19-23, 2013 (2013), pp. 2058–2061.

[270] P. Sasdrich, A. Moradi, O. Mischke, and T. Güneysu, "Achieving Side-Channel Protection with Dynamic Logic Reconfiguration on Modern FPGAs", IACR Cryptology ePrint Archive **2015**, 203 (2015).

[271] F. Madlener, M. Stöttinger, and S. Huss, "Novel hardening techniques against differential power analysis for multiplication in GF(2^n)", in Field-Programmable Technology, 2009. FPT 2009. International Conference on (2009), pp. 328–334.

[272] S. Ali, R. S. Chakraborty, D. Mukhopadhyay, and S. Bhunia, "Multi-level attacks: An emerging security concern for cryptographic hardware", in Design, Automation and Test in Europe, DATE 2011, Grenoble, France, March 14-18, 2011 (2011), pp. 1176–1179.

[273] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware Trojan: Threats and Emerging solutions", in IEEE International High Level Design Validation and Test Workshop, HLDVT 2009, San Francisco, CA, USA, 4-6 November 2009 (2009), pp. 166–171.

[274] M. Tehranipoor and D. Forte, "Tutorial T4: All You Need to Know about Hardware Trojans and Counterfeit ICs", in 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, Mumbai, India, January 5-9, 2014 (2014), pp. 9–10.

[275] Z. Chen, X. Guo, R. Nagesh, A. Reddy, M. Gora, and A. Maiti, *Hardware Trojan Designs on BASYS FPGA Board*.

[276] A. P. Johnson, S. Saha, R. S. Chakraborty, D. Mukhopadhyay, and S. Gören, "Fault Attack on AES via Hardware Trojan Insertion by Dynamic Partial Reconfiguration of FPGA over Ethernet", in Proceedings of the 9th Workshop on Embedded Systems Security, WESS '14 (2014), 1:1–1:8.

[277] S. Bhasin, J.-L. Danger, S. Guilley, X. T. Ngo, and L. Sauvage, "Hardware Trojan Horses in Cryptographic IP Cores", in FDTC, edited by W. Fischer and J.-M. Schmidt (2013), pp. 15–29.

[278] *Benchmarks*, `https://www.trust-hub.org/resources/benchmarks`, Accessed: 2015-01-30.

[279] N. Homma, Y. Hayashi, N. Miura, D. Fujimoto, D. Tanaka, M. Nagata, and T. Aoki, "EM Attack Is Non-invasive? - Design Methodology and Validity Verification of EM Attack Sensor", in Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings (2014), pp. 1–16.

[280]S. Ali, D. Mukhopadhyay, and M. Tunstall, "Differential fault analysis of AES: towards reaching its limits", J. Cryptographic Engineering **3**, 73–97 (2013).

[281]A. Poschmann, S. Ling, and H. Wang, "256 Bit Standardized Crypto for 650 GE — GOST Revisited", English, in *Cryptographic Hardware and Embedded Systems, CHES 2010*, Vol. 6225, edited by S. Mangard and F.-X. Standaert, Lecture Notes in Computer Science (Springer Berlin Heidelberg, 2010), pp. 219–233.

[282]S. Hajra, C. Rebeiro, S. Bhasin, G. Bajaj, S. Sharma, S. Guilley, and D. Mukhopadhyay, "DRECON: DPA Resistant Encryption by Construction", in AFRICACRYPT, Vol. 8469, edited by D. Pointcheval and D. Vergnaud, Lecture Notes in Computer Science (2014), pp. 420–439.

[283]A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher", in CHES, Vol. 4727, LNCS, Vienna, Austria (2007), pp. 450–466.

[284]V. Izosimov, A. Asvestopoulos, O. Blomkvist, and M. Törngren, "Security-aware development of cyber-physical systems illustrated with automotive case study", in Proceedings of the 2016 Conference on Design, Automation & Test in Europe (EDA Consortium, 2016), pp. 818–821.

[285]J. Wan, A. Canedo, and M. A. Al Faruque, "Security-aware functional modeling of cyber-physical systems", in Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on (IEEE, 2015), pp. 1–4.

[286]D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, et al., "Imperfect forward secrecy: How Diffie-Hellman fails in practice", in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (ACM, 2015), pp. 5–17.

[287]N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices.", in USENIX Security Symposium, Vol. 8 (2012), p. 1.

[288]P. Švenda, M. Nemec, P. Sekan, R. Kvašňovskỳ, D. Formánek, D. Komárek, and V. Matyáš, "The Million-Key Question–Investigating the Origins of RSA Public Keys", in 25th USENIX Security Symposium. Proceedings (2016).

[289]M. J. Wiener, "Cryptanalysis of short RSA secret exponents", IEEE Transactions on Information theory **36**, 553–558 (1990).

[290]D. Boneh, G. Durfee, and Y. Frankel, "An attack on RSA given a small fraction of the private key bits", in International Conference on the Theory and Application of Cryptology and Information Security (Springer, 1998), pp. 25–34.

[291] V. M. Sidelnikov, "A public-key cryptosystem based on binary Reed-Muller codes", Discrete Mathematics and Applications **4**, 191–208 (1994).

[292] D. Boneh and G. Durfee, "Cryptanalysis of RSA with private key d less than N/sup 0.292", IEEE transactions on Information Theory **46**, 1339–1349 (2000).

[293] F. Weimer, *Factoring RSA Keys With TLS Perfect Forward Secrecy*, 2015.

[294] J. Hastad, "Solving simultaneous modular equations of low degree", siam Journal on Computing **17**, 336–341 (1988).

[295] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults", in International conference on the theory and applications of cryptographic techniques (Springer, 1997), pp. 37–51.

[296] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions", Computational Complexity **16**, 365–411 (2007).

[297] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings", J. ACM **60**, 43 (2013).

[298] L. K. Grover, "A fast quantum mechanical algorithm for database search", in Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (ACM, 1996), pp. 212–219.

[299] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography* (US Department of Commerce, National Institute of Standards and Technology, 2016).

[300] T. P. Berger and P. Loidreau, "How to mask the structure of codes for a cryptographic use", Designs, Codes and Cryptography **35**, 63–79 (2005).

[301] R. Misoczki and P. S. Barreto, "Compact McEliece keys from Goppa codes", in International Workshop on Selected Areas in Cryptography (Springer, 2009), pp. 376–392.

[302] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory", JPL DSN Progress Report **4244**, 114–116 (1978).

[303] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto, "MDPC-McEliece: New McEliece variants from moderate density parity-check codes", in Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on (IEEE, 2013), pp. 2069–2073.

[304] Q. Guo, T. Johansson, and P. Stankovski, "A key recovery attack on MDPC with CCA security using decoding errors", in International Conference on the Theory and Application of Cryptology and Information Security (Springer, 2016), pp. 789–815.

[305] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2007), pp. 450–466.

[306]J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, "The LED Block Cipher", in Cryptographic Hardware and Embedded Systems–CHES 2011: 13th International Workshop, Nara, Japan, September 28–October 1, 2011, Proceedings, Vol. 6917 (Springer, 2011), p. 326.

[307]R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The SIMON and SPECK lightweight block ciphers", in Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE (IEEE, 2015), pp. 1–6.

[308]C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS", in Annual Cryptology Conference (Springer, 2016), pp. 123–153.

[309]S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: a small PRESENT", in International Conference on Cryptographic Hardware and Embedded Systems (Springer, 2017), pp. 321–345.

[310]M. Hell, T. Johansson, and W. Meier, "Grain: a stream cipher for constrained environments", International Journal of Wireless and Mobile Computing **2**, 86–93 (2007).

[311]V. Mikhalev, F. Armknecht, and C. Müller, "On ciphers that continuously access the non-volatile key", IACR Transactions on Symmetric Cryptology **2016**, 52–79 (2017).

[312]V. A. Ghafari, H. Hu, and Y. Chen, "Fruit-v2: ultra-lightweight stream cipher with shorter internal state", IACR Cryptology ePrint Archive **2016**, 355 (2016).

[313]M. Hamann, M. Krause, and W. Meier, "LIZARD–a lightweight stream cipher for power-constrained devices", IACR Transactions on Symmetric Cryptology **2017**, 45–79 (2017).

[314]J. Guo, T. Peyrin, and A. Poschmann, "The PHOTON family of lightweight hash functions", in Annual Cryptology Conference (Springer, 2011), pp. 222–239.

[315]S. Kumar, J. Haj-Yihia, M. Khairallah, M. A. Elmohr, and A. Chattopadhyay, "A Comprehensive Performance Analysis of Hardware Implementations of CAESAR Candidates", IACR Cryptology ePrint Archive **2017**, 1261 (2017).

[316]S. Vaudenay, "Security Flaws Induced by CBC Padding-Applications to SSL, IPSEC, WTLS...", in International Conference on the Theory and Applications of Cryptographic Techniques (Springer, 2002), pp. 534–545.

[317]T Duong and J Rizzo, "BEAST: Surprising crypto attack against HTTPS", Blog, September **42**, 45–47 (2011).

[318]N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols", in Security and Privacy (SP), 2013 IEEE Symposium on (IEEE, 2013), pp. 526–540.

[319]T. Duong and J. Rizzo, *The CRIME attack*, https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-1Ca2GizeuOfaLU2HOU/edit#slide=id.g1d134dff_1_222.

[320]Y. Gluck, N. Harris, and A. Prado, "BREACH: reviving the CRIME attack", Unpublished manuscript (2013).

[321]Microsoft, *SecureZeroMemory function*, https://msdn.microsoft.com/en-us/library/windows/desktop/aa366877(v=vs.85).aspx.

[322]FreeBSD, *explicit_bzero - FreeBSD Library Functions Manual*, https://www.freebsd.org/cgi/man.cgi?query=explicit_bzero&sektion=3.

[323]ISO/IEC, *ISO/IEC 9899:201x for C Programming Language*, http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf.

[324]Z. Yang, B. Johannesmeyer, A. T. Olesen, S. Lerner, and K. Levchenko, "Dead store elimination (still) considered harmful", in 26th USENIX Security Symposium. USENIX Association (2017).

[325]A. Moradi, S. Guilley, and A. Heuser, "Detecting Hidden Leakages", in Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings (2014), pp. 324–342.

[326]P. Rauzy, S. Guilley, and Z. Najm, "Formally proved security of assembly code against power analysis - A case study on balanced logic", J. Cryptographic Engineering **6**, 201–216 (2016).

[327]S. Bhasin, N. Bruneau, J. Danger, S. Guilley, and Z. Najm, "Analysis and Improvements of the DPA Contest v4 Implementation", in Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings (2014), pp. 201–218.

[328]J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, "State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures", in Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on (IEEE, 2010), pp. 76–87.

[329]J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 1999), pp. 292–302.

[330]K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation", in Proceedings of the conference on Design, automation and test in Europe-Volume 1 (IEEE Computer Society, 2004), p. 10246.

[331]J.-L. Danger, S. Guilley, S. Bhasin, and M. Nassar, "Overview of dual rail with precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors", in Signals, Circuits and Systems (SCS), 2009 3rd International Conference on (IEEE, 2009), pp. 1–8.

[332]S. Bhasin, S. Guilley, F. Flament, N. Selmane, and J.-L. Danger, "Countering early evaluation: an approach towards robust dual-rail precharge logic", in Proceedings of the 5th Workshop on Embedded Systems Security (ACM, 2010), p. 6.

[333]M. Ender, A. Wild, and A. Moradi, "SafeDRP: Yet Another Way Toward Power-Equalized Designs in FPGA", in International Workshop on Constructive Side-Channel Analysis and Secure Design (Springer, 2017), pp. 83–101.

[334]O. Acıiçmez and W. Schindler, "A Vulnerability in RSA Implementations Due to Instruction Cache Analysis and Its Demonstration on OpenSSL", in Topics in Cryptology – CT-RSA 2008, edited by T. Malkin (2008), pp. 256–273.

[335]M. Dugardin, S. Guilley, J.-L. Danger, Z. Najm, and O. Rioul, "Correlated Extra-Reductions Defeat Blinded Regular Exponentiation", in Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings (2016), pp. 3–22.

[336]Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware", Journal of Cryptographic Engineering, 1–27 (2016).

[337]R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against kernel space ASLR", in Security and Privacy (SP), 2013 IEEE Symposium on (IEEE, 2013), pp. 191–205.

[338]O. Acıiçmez and J.-P. Seifert, "Cheap hardware parallelism implies cheap security", in Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on (IEEE, 2007), pp. 80–91.

[339]O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert, "Predicting secret keys via branch prediction", in Cryptographers Track at the RSA Conference (Springer, 2007), pp. 225–242.

[340]M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham, "On subnormal floating point and abnormal timing", in Security and Privacy (SP), 2015 IEEE Symposium on (IEEE, 2015), pp. 623–639.

[341]D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Jump over ASLR: Attacking branch predictors to bypass ASLR", in Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on (IEEE, 2016), pp. 1–13.

[342]Y. Bulygin, "CPU side-channels vs. virtualization malware: The good, the bad, or the ugly", Proceedings of the ToorCon (2008).
[343]D. J. Bernstein, "Cache-timing attacks on AES", (2005).
[344]G. I. Apecechea, M. S. Inci, T. Eisenbarth, and B. Sunar, "Fine grain Cross-VM Attacks on Xen and VMware are possible!", IACR Cryptology ePrint Archive **2014**, 248 (2014).
[345]Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack.", in USENIX Security Symposium (2014), pp. 719–732.
[346]D. Gruss, R. Spreitzer, and S. Mangard, "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches.", in USENIX Security Symposium (2015), pp. 897–912.
[347]D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ Flush: a fast and stealthy cache attack", in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (Springer, 2016), pp. 279–299.
[348]Z. Wu, Z. Xu, and H. Wang, "Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud.", in USENIX Security symposium (2012), pp. 159–173.
[349]A. Richter, C. Herber, H. Rauchfuss, T. Wild, and A. Herkersdorf, "Performance isolation exposure in virtualized platforms with pci passthrough i/o sharing", in International Conference on Architecture of Computing Systems (Springer, 2014), pp. 171–182.
[350]W. Song, J. Kim, J.-W. Lee, and D. Abts, "Security vulnerability in processor-interconnect router design", in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (ACM, 2014), pp. 358–368.
[351]X. Wang, N. Zeldovich, M. F. Kaashoek, and A. Solar-Lezama, "Towards optimization-safe systems: Analyzing the impact of undefined behavior", in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (ACM, 2013), pp. 260–275.
[352]Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks", in Annual International Cryptology Conference (Springer, 2003), pp. 463–481.
[353]D. B. Roy, S. Bhasin, S. Guilley, J.-L. Danger, and D. Mukhopadhyay, "From theory to practice of private circuit: A cautionary note", in Computer Design (ICCD), 2015 33rd IEEE International Conference on (IEEE, 2015), pp. 296–303.
[354]A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: exposing the perils of security-oblivious energy management", in 26th USENIX Security Symposium (2017).

[355]T. Pöppelmann, T. Oder, and T. Güneysu, "High-Performance Ideal Lattice-Based Cryptography on 8-Bit ATxmega Microcontrollers", in Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings (2015), pp. 346–365.

[356]Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede, "Efficient Ring-LWE encryption on 8-bit AVR processors", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2015), pp. 663–682.

[357]J. Howe, C. Moore, M. O'Neill, F. Regazzoni, T. Güneysu, and K. Beeden, "Lattice-based Encryption over Standard Lattices in Hardware", in Proceedings of the 53rd Annual Design Automation Conference (ACM, 2016), p. 162.

[358]T. Pöppelmann, L. Ducas, and T. Güneysu, "Enhanced lattice-based signatures on reconfigurable hardware", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2014), pp. 353–370.

[359]H. Chen, K. Lauter, and K. E. Stange, *Attacks on search RLWE*, https://www.microsoft.com/e
us/research/publication/attacks-on-search-rlwe/, 2015.

[360]R. Cramer, L. Ducas, C. Peikert, and O. Regev, "Recovering short generators of principal ideals in cyclotomic rings", in Annual International Conference on the Theory and Applications of Cryptographic Techniques (Springer, 2016), pp. 559–585.

[361]E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-Quantum Key Exchange-A New Hope.", in USENIX Security Symposium (2016), pp. 327–343.

[362]J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé, *CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM*, tech. rep. (2017).

[363]P. FIPS, "180-4", Secure hash standard (SHS), March (2012).

[364]M. J. Dworkin, *SHA-3 standard: Permutation-based hash and extendable-output functions*, tech. rep. (2015).

[365]*CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness*, See `http://competitions.cr.yp.to/caesar.html`.

[366]T. Tal Be'ery and A. Shulman, "A perfect crime? only time will tell", Black Hat Europe **2013** (2013).

[367]C. Clavier, J.-S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures", in International Workshop on Cryptographic Hardware and Embedded Systems (Springer, 2000), pp. 252–263.

[368]J. McCarthy and J. Painter, "Correctness of a compiler for arithmetic expressions", Mathematical aspects of computer science **1** (1967).

[369]J. A. Painter, *Semantic correctness of a compiler for an Algol-like language*, tech. rep. (STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE, 1967).

[370]V. D'Silva, M. Payer, and D. Song, "The correctness-security gap in compiler optimization", in Security and Privacy Workshops (SPW), 2015 IEEE (IEEE, 2015), pp. 73–87.

[371]C. Deng and K. S. Namjoshi, "Securing a compiler transformation", in International Static Analysis Symposium (Springer, 2016), pp. 170–188.

[372]GIZMODO, *The Trillion Fold Increase In Computing Power, Visualized*, https://gizmodo.com/the-trillion-fold-increase-in-computing-power-visualiz-1706676799.

[373]C. Carvalho, "The gap between processor and memory speeds", in Proc. of IEEE International Conference on Control and Automation (2002).

[374]Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks", in Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on (IEEE, 2012), pp. 142–151.

[375]R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK lightweight block ciphers", in Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015 (2015), 175:1–175:6.

[376]C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS", in CRYPTO (2), Vol. 9815, Lecture Notes in Computer Science (2016), pp. 123–153.

[377]S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption", in Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, Vol. 10529, Lecture Notes in Computer Science (2017), pp. 321–345.

[378]T. B. S. Reis, D. F. Aranha, and J. López, "PRESENT Runs Fast - Efficient and Secure Implementation in Software", in Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings (2017), pp. 644–664.

[379]K. Bhargavan and G. Leurent, "On the Practical (In-)Security of 64-bit Block Ciphers", in ACM CCS 2016 - 23rd ACM Conference on Computer and Communications Security (2016), pp. 456–467.

[380]A. Chakraborti, N. Datta, A. Jha, C. M. Lopez, M. Nandi, and Y. Sasaki, *ESTATE*, Submission to the NIST Lightweight Cryptography project, 2019.

[381] S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, *GIFT-COFB v1.0*, Submission to the NIST Lightweight Cryptography project, 2019.

[382] A. Chakraborti, N. Datta, A. Jha, and M. Nandi, *HYENA*, Submission to the NIST Lightweight Cryptography project, 2019.

[383] A. Chakraborti, N. Datta, A. Jha, C. M. Lopez, M. Nandi, and Y. Sasaki, *LOTUS-AEAD and LOCUS-AEAD*, Submission to the NIST Lightweight Cryptography project, 2019.

[384] S. Gueron and Y. Lindell, *Simple: a simple AEAD scheme*, Submission to the NIST Lightweight Cryptography project, 2019.

[385] S. Banik, A. Bogdanov, T. Peyrin, Y. Sasaki, S. M. Sim, E. Tischhauser, and Y. Todo, *SUNDAE-GIFT v1.0*, Submission to the NIST Lightweight Cryptography project, 2019.

[386] T. Iwata, M. Khairallah, K. Minematsu, T. Peyrin, Y. Sasaki, S. M. Sim, and L. Sun, *Thank Goodness It's Friday (TGIF)*, Submission to the NIST Lightweight Cryptography project, 2019.

[387] L. May, L. Penna, and A. Clark, "An Implementation of Bitsliced DES on the Pentium MM^{XTM} Processor", in Information Security and Privacy, edited by E. P. Dawson, A. Clark, and C. Boyd (2000), pp. 112–122.

[388] C. Wolf, *RISC-V Bitmanip Extension*, 2020.

[389] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of lightweight block ciphers for the Internet of things", J. Cryptographic Engineering **9**, 283–302 (2019).

[390] P. Schwabe and K. Stoffelen, "All the AES You Need on Cortex-M3 and M4", in Selected Areas in Cryptography - SAC 2016 (2016), pp. 180–194.

[391] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks", in Advances in Cryptology — CRYPTO' 99, edited by M. Wiener (1999), pp. 398–412.

[392] A. Biryukov, D. Dinu, Y. Le Corre, and A. Udovenko, "Optimal First-Order Boolean Masking for Embedded IoT Devices", in Smart Card Research and Advanced Applications, edited by T. Eisenbarth and Y. Teglia (2018), pp. 22–41.

[393] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, *Ascon v1.2*, Submission to Round 1 of the NIST Lightweight Cryptography project, 2019.

[394] A. Adomnicai, J. J. Fournier, and L. Masson, *Masking the Lightweight Authenticated Ciphers ACORN and Ascon in Software*, Cryptology ePrint Archive, Report 2018/708, 2018.

[395] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: a small PRESENT", in International Conference on Cryptographic Hardware and Embedded Systems (Springer, 2017), pp. 321–345.

[396] M. R. Albrecht, B. Driessen, E. B. Kavun, G. Leander, C. Paar, and T. Yalçın, "Block ciphers–focus on the linear layer (feat. PRIDE)", in International Cryptology Conference (Springer, 2014), pp. 57–76.

[397] Xilinx, *Virtex-5 FPGA System Monitor*, http://www-inst.eecs.berkeley.edu/~cs150/fa13/resources/ug192.pdf.

[398] B. Preneel and T. Takagi, eds., *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 – October 1, 2011. Proceedings*, Vol. 6917, LNCS (Springer, 2011).

[399] P. Paillier and I. Verbauwhede, eds., *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, Vol. 4727, LNCS (Springer, 2007).

[400] W. Fischer and N. Homma, eds., *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, Vol. 10529, Lecture Notes in Computer Science (Springer, 2017).

Appendix

15
CHAPTER 5

.1. Proof of Theorem 5.4.3

In order to prove the Theorem 5.4.3 let us first introduce some lemmas. By Remark 2 the only random parts of are the noise and the mask. As a consequence the random variable $(|M = m)$ depends only on the noise, and is equal to:

$$(|M = m) = -2 \times \frac{1}{2^n} \sum_{\omega=0}^{2^n-1} \left[\left(\Phi(\omega) \oplus m + N_\omega^{(1)} - \frac{n}{2} \right) \times \left(\Phi(\omega) + N_\omega^{(2)} - \frac{n}{2} \right) \right] . \tag{1}$$

$$\begin{aligned} (|M = m) = m - \frac{n}{2} & \\ & - 2 \times N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) \\ & - 2 \times N_\omega^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) \\ & - 2 \times N_\omega^{(1)} \times N_\omega^{(2)} \ . \end{aligned} \tag{2}$$

$(|M = m)$ can be split into a deterministic part and a random part:

$$(|M = m) = -2 \times (S_d + S_r) \ ,$$

where

$$\begin{aligned} S_d &= \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) \times \left(\Phi(\omega) - \frac{n}{2} \right) \ , \\ S_r &= N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) \\ &\quad + N_\omega^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) \\ &\quad + N_\omega^{(1)} \times N_\omega^{(2)} \ . \end{aligned}$$

$$\begin{aligned} S_d &= (U \oplus M - U \oplus M) \times (U - U \oplus M) \mid M = m \\ &= -\frac{1}{2} m + \frac{n}{4} \qquad \text{by [99]} , \end{aligned}$$

where U denotes a random variable drawn uniformly over $\mathbb{F}_2^n$.

$$(|M = m) = 4 \times \left(\frac{\sigma^2}{2^n} \times \frac{n}{2} + \frac{\sigma^4}{2^n} \right) . \tag{3}$$

Recall that the random variable $(|M = m)$ can be write as in Lemma .1; thus $(|M = m) = 4 \times (V_1 + V_2 + V_3 + C_1 + C_2 + C_3)$, where

$$V_1 = N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) ,$$

$$V_2 = N_\omega^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) ,$$

$$V_3 = N_\omega^{(1)} \times N_\omega^{(2)} ,$$

$$C_1 = 2 \times N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right), N_\omega^{(1)} \times N_\omega^{(2)} ,$$

$$C_2 = 2 \times N_\omega^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right), N_\omega^{(1)} \times N_\omega^{(2)} ,$$

$$C_3 = 2 \times \mathrm{Cov} \left[N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) , \right.$$
$$\left. N_\omega^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) \right] .$$

Let us now prove that $C_1 = C_2 = 0$. First we have:

$$N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right), N_\omega^{(1)} \times N_\omega^{(2)} = C_1^{(1)} - C_1^{(2)},$$

with

$$\begin{aligned} C_1^{(1)} &= \Phi(\omega) \times N_\omega^{(1)}, N_\omega^{(1)} \times N_\omega^{(2)} \\ &= \frac{1}{2^n} \sum_{\omega'=0}^{2^n-1} \Phi(\omega) \times N_\omega^{(1)}, N_{\omega'}^{(1)} \times N_{\omega'}^{(2)} . \end{aligned}$$

The random variables $N_\omega^{(i)}$, where $i \in \{1, 2\}$ and $\omega \in \mathbb{F}_2^n$ are mutually independent and independent with all the $\Phi(\omega)$. Thus we have:

$$\begin{aligned} & \forall \omega, \omega', \Phi(\omega) \times N_\omega^{(1)}, N_{\omega'}^{(1)} \times N_{\omega'}^{(2)} = 0 \\ \iff & \frac{1}{2^n} \sum_{\omega'=0}^{2^n-1} \Phi(\omega) \times N_\omega^{(1)}, N_{\omega'}^{(1)} \times N_{\omega'}^{(2)} = 0 \\ \iff & C_1^{(1)} = 0 . \end{aligned}$$

Besides

$$\begin{aligned} C_1^{(2)} &= \frac{n}{2} \times N_\omega^{(1)}, N_\omega^{(1)} \times N_\omega^{(2)} \\ &= \frac{n}{2} \times \frac{1}{2^n} \sum_{\omega'=0}^{2^n-1} N_\omega^{(1)}, N_{\omega'}^{(1)} \times N_{\omega'}^{(2)} \ . \end{aligned}$$

As $N_\omega^{(i)}$, where $i \in \{1,2\}$ and $\omega \in \mathbb{F}_2^n$, are mutually independent, we have:

$$\begin{aligned} & N_\omega^{(1)}, N_{\omega'}^{(1)} \times N_{\omega'}^{(2)} = 0, \ \forall (\omega, \omega') \in \mathbb{F}_2^n \times \mathbb{F}_2^n \\ \iff & C_1^{(2)} = \frac{n}{2} \times N_\omega^{(1)}, N_\omega^{(1)} \times N_\omega^{(2)} = 0 \\ \iff & \frac{1}{2^n} \sum_{\omega=0}^{2^n-1} N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right), \frac{1}{2^n} \sum_{\omega=0}^{n-1} N_\omega^{(1)} \times N_\omega^{(2)} = 0 \ . \end{aligned}$$

Identically we prove that:

$$N_\omega^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right), N_\omega^{(1)} \times N_\omega^{(2)} = 0 \ .$$

As a consequence $C_1 = C_2 = 0$. Let us now study C_3. By the bi-linearity of the covariance C_3 can be rewritten such that:

$$C_3 = \frac{2}{2^{2n}} \sum_{\omega=0}^{2^n-1} \sum_{\omega'=0}^{2^n-1} N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right), N_{\omega'}^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right).$$

But

$$\begin{aligned} & N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right), N_{\omega'}^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) \\ & = N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) \times N_{\omega'}^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) \\ & - N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) \times N_{\omega'}^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) \ . \end{aligned}$$

By definition, $N_\omega^{(1)}$ is independent from $\Phi(\omega)$. Thus:

$$\begin{aligned} & N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) = N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) = 0 \text{ and} \\ & N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) \times N_{\omega'}^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right) = 0 \ . \end{aligned}$$

$N_\omega^{(1)}$ is independent from $\left(\Phi(\omega) - \frac{n}{2} \right) \times N_{\omega'}^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2} \right)$. Thus $N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2} \right) \times N_{\omega'}^{(2)} \times \left(\Phi(\omega) \oplus m - \right.$ 0, which implies that $C_3 = 0$. As a consequence $(|M = m) = 4 \times (V_1 + V_2 + V_3)$.

$$
\begin{aligned}
V_1 &= N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2}\right) \\
&= \frac{1}{2^{2n}} \sum_{\omega=0}^{2^n-1} N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2}\right) \\
&\quad + \frac{2}{2^{2n}} \sum_{0 \leqslant \omega < \omega' \leqslant 2^n-1} N_\omega^{(1)} \times (\Phi(\omega)), N_{\omega'}^{(1)} \times (\Phi(\omega')) \ . \qquad (4)
\end{aligned}
$$

As $N_\omega^{(1)} \times (\Phi(\omega)), N_{\omega'}^{(1)} \times (\Phi(\omega')) = 0$, the terms in Eq. (4) are all null. It can be noticed that $\Phi(\omega) - \frac{n}{2} = 0$ as $\Phi(\omega)$ is uniformly distributed over S_{2^n} and $N_\omega^{(1)} = 0$. As a consequence:

$$
\begin{aligned}
N_\omega^{(1)} \times \left(\Phi(\omega) - \frac{n}{2}\right) &= \Phi(\omega) - \frac{n}{2} \times N_\omega^{(1)} \text{ , hence} \\
V_1 &= \frac{1}{2^{2n}} \sum_{\omega=0}^{2^n-1} \sigma^2 \times \frac{n}{4} = \frac{1}{2^n} \times \sigma^2 \times \frac{n}{4} \ .
\end{aligned}
$$

Identically, we have

$$
\begin{aligned}
V_2 &= N_\omega^{(2)} \times \left(\Phi(\omega) \oplus m - \frac{n}{2}\right) = \frac{\sigma^2}{2^n} \times \frac{n}{4} = V_1 \text{ , and} \\
V_3 &= \frac{1}{2^n} \sum_{\omega=0}^{2^n-1} N_\omega^{(1)} \times N_\omega^{(2)} = \frac{\sigma^4}{2^n} \ .
\end{aligned}
$$

Finally

$$
(|M = m) = 4 \times \left(\frac{\sigma^2}{2^n} \times \frac{n}{2} + \frac{\sigma^4}{2^n}\right) \ .
$$

Then let us prove the Theorem 5.4.3.

Lemma .1 gives us the value of the variance of the noise. Then by the definition of the , we have:

$$
\begin{aligned}
, M \geqslant X^{(3)}, M &\iff \frac{M}{(|M = m)} \geqslant \frac{M}{N^3} \\
&\iff 4 \times \left(\frac{\sigma^2}{2^n} \times \frac{n}{2} + \frac{\sigma^4}{2^n}\right) \leqslant \sigma^2 \\
&\iff \frac{2^{n-1} - n}{2} \geqslant \sigma^2
\end{aligned}
$$

.2. Proof of the propositions of Sect. 5.4.5

A. Proof of Prop. 5.4.5

By Theorem 2 in [121, Appendix A.2] (extended version of [120]) we have that the of is given by

$$= \min_{k \neq k^\star} \frac{\kappa\left(k^\star, k\right)}{2\left(\frac{\kappa'(k^\star,k)}{\kappa(k^\star,k)} - \kappa\left(k^\star, k\right)\right) + 2\sum_{\substack{i \in \{0,2\}^d \\ i \neq (0,\ldots,0)}} \prod_{1 \leq \delta \leq d} \left(\alpha_\delta^{-i_\delta} \cdot \sigma_\delta^{i_\delta}\right)}$$

$$= \min_{k \neq k^\star} \frac{\kappa\left(k^\star, k\right)}{2\left(\frac{\kappa'(k^\star,k)}{\kappa(k^\star,k)} - \kappa\left(k^\star, k\right)\right) + 2\left(\alpha_1^{-2}\sigma_1^2 + \alpha_2^{-2}\sigma_2^2 + \alpha_1^{-2}\sigma_1^2\alpha_2^{-2}\sigma_2^2\right)}.$$

B. Proof of Prop. 5.4.5

$$\frac{m^{()} - m^{()}}{m^{()}} = \left(\frac{\log(1-)}{} - \frac{\log(1-)}{}\right) \times \frac{}{\log(1-)}$$
$$= \cdot - 1 \ ,$$

which indeed does not depend on .

C. Proof of Prop. 5.4.5

Let us now compute the difference of traces needed to reach any .

$$m^{()} - m^{()} = \frac{\log(1-)}{} - \frac{\log(1-)}{}$$

Let us rewrite using $\alpha = \alpha_1 = \alpha_2$. In such case:

$$m^{()} - m^{()} = \frac{\log(1-)}{} - \frac{\log(1-)}{}$$

$$= \frac{\log(1-)}{} - \frac{\log(1-)}{}$$

$$= \left(2\alpha^{-2}\frac{\log(1-)}{\kappa\left(k^\star, k\right)}\right)\left(1 + \alpha^{-2}\sigma^2\right)\left(\sigma^2 - 4\left(\frac{\sigma^2}{2^n}\frac{n}{2} + \frac{\sigma^4}{2^n}\right)\right)$$

The attacks perform similarly when $m - m = 0$ which implies $(\sigma^2 - 4 \times \left(\frac{\sigma^2}{2^n} \times \frac{n}{2} + \frac{\sigma^4}{2^n}\right) = 0$. Notice that we recover here the results of the Subsect. 5.4.3.

In order to find the noise when the maximum occurs let us compute the derivative in σ^2:

$$\frac{\mathrm{d}\left(m^{()}-m^{()}\right)}{\mathrm{d}\sigma^2} = \left(\left(\alpha^{-2}-\frac{4\alpha^{-2}}{2^n}\times\frac{n}{2}\right)+\left(\frac{8\alpha^{-2}}{2^n}+2\alpha^{-4}-\frac{8\alpha^{-4}}{2^n}\times\frac{n}{2}\right)\sigma^2-\frac{12\alpha^{-4}\sigma^4}{2^n}\right) \times\left(2\frac{\log(1-)}{\kappa(k^\star,k)}\right)$$

The maximum occurs when $\frac{\mathrm{d}\left(m^{()}-m^{()}\right)}{\mathrm{d}\sigma^2}=0$ which not depends on the .

.3. Proof of Theorem 5.5.3

Similarly to the Remark 2 we have $\forall i<d$:

$$(d|M_i=m)=\frac{-2}{d\,2^n}\sum_{\substack{\omega\in\mathbb{F}_{2^n}\\ j\in[\![1,d]\!]}}\left[\left(\Phi(\omega)\oplus m+N^{(1)}_{(\omega,j)}-\frac{n}{2}\right)\times\left(\Phi(\omega)+N^{(2)}_{(\omega,j)}-\frac{n}{2}\right)\right].$$

As the i is fixed for each share we have removed it from the index position.

$$(d|M_i=m)=4\times\left(\frac{\sigma^2}{d\times 2^n}\times\frac{n}{2}+\frac{\sigma^4}{d\times 2^n}\right), \tag{5}$$

where d is the number of share of the high-order masking scheme and $i<d$.

Lemma .3 is a straightforward extension of Lemma .1.

Exploiting Lemma .3 let us prove Theorem 5.5.3.

As Lemma .3 gives us the variance of the noise of the second-order leakage we have

$\forall i < d$

$$\begin{aligned} & d, M_i \geqslant X_i^{(3)}, M_i \\ & \iff \frac{M}{(d|M_i = m)} \geqslant \frac{M}{N_i^{(3)}} \\ & \iff 4 \times \left(\frac{\sigma^2}{d \times 2^n} \times \frac{n}{2} + \frac{\sigma^4}{d \times 2^n} \right) \leqslant \sigma^2 \\ & \iff \left(n - d \times 2^{n-1} \right) \frac{\sigma^2}{d \times 2^{n-1}} + \frac{\sigma^4}{d \times 2^{n-2}} \leqslant 0. \end{aligned}$$

The upper bound of the interval are the σ^2 where $\sigma^2 \neq 0$ and:

$$\begin{aligned} \frac{\sigma^4}{d \times 2^{n-2}} &= \left(d \times 2^{n-1} - n \right) \frac{\sigma^2}{d \times 2^{n-1}} \\ \iff \sigma^2 &= \frac{\left(d \times 2^{n-1} - n \right)}{2} \\ \iff \sigma^2 &= d \times 2^{n-2} - \frac{n}{2}. \end{aligned}$$

It implies that the size of Useful Interval of Variance is given by $d \times 2^{n-2} - \frac{n}{2}$.

.4. Affine model

A. Proof of Lemma 5.6.1

$$\begin{aligned} & (\Psi_\alpha(U) - \Psi_\alpha(U)) \times \left(\Psi_\beta(U \oplus z) - \Psi_\beta(U \oplus z) \right) \\ & = \left(\alpha \cdot U - \alpha \cdot \left(\frac{1}{2}\mathbf{1} \right) \right) \times \left(\beta \cdot (U \oplus z) - \beta \cdot \left(\frac{1}{2}\mathbf{1} \right) \right) \\ & = \left(\alpha \cdot \left(U - \frac{1}{2}\mathbf{1} \right) \right) \times \left(\beta \cdot \left((U \oplus z) - \frac{1}{2}\mathbf{1} \right) \right) \\ & = \left(\frac{1}{2}\alpha \cdot \overline{U} \right) \times \left(\frac{1}{2}\beta \cdot \left(\overline{U \oplus z} \right) \right) \\ & = \frac{1}{4} \left(\alpha \overline{U} \left(\overline{U \oplus z} \right) \beta \right) \ , \end{aligned}$$

where $\overline{U}$ denotes $2\left(U - \frac{1}{2}\mathbf{1}\right)$.

It can also be noticed that: $U = -\left((-1)^{U_1}, \ldots, (-1)^{U_n} \right)$, and thus $\overline{(U \oplus z)} = -\left((-1)^{U_1 + z_1}, \ldots, (-1)^{U_n + z_n} \right)$.

Moreover

$$\begin{aligned}
&\overline{U}\left(\overline{U\oplus z}\right)=\overline{U},\left(\overline{U\oplus z}\right)\\
\Longrightarrow&\left(\overline{U}\left(\overline{U\oplus z}\right)\right)_{i,j}=(-1)^{U_i},(-1)^{\left(U_j+z_j\right)}\\
\Longrightarrow&\left(\overline{U}\left(\overline{U\oplus z}\right)\right)_{i,j}=0 \text{ if } i\neq j \text{ or } \left(\overline{U}\left(\overline{U\oplus z}\right)\right)_{i,j}=(-1)^{z_j} \text{ if } i=j\ .
\end{aligned}$$

Eventually, we have:

$$\begin{aligned}
&\left(\Psi_\alpha(U)-\Psi_\alpha(U)\right)\times\left(\Psi_\beta(U\oplus z)-\Psi_\beta(U\oplus z)\right)\\
&\quad=-\frac{1}{4}(\alpha\odot\beta)\cdot\overline{z}=-\frac{1}{4}(\alpha\odot\beta)\cdot 2\left(z-\frac{1}{2}\mathbf{1}\right)\\
&\quad=-\frac{1}{2}(\alpha\odot\beta)\cdot z+\frac{1}{4}(\alpha\odot\beta)\cdot\mathbf{1}=-\frac{1}{2}(\alpha\odot\beta)\cdot z+\frac{1}{4}\alpha\cdot\beta\ .
\end{aligned}$$

B. Proof of the Theorem 5.6.3

Similarly to Eq. (1) we have:

$$\begin{aligned}
(|M=m)=-2\times\frac{1}{2^n}\sum_{\omega=0}^{2^n-1}\Bigg[&\left(\alpha\cdot(\Phi(\omega)\oplus m)+N_\omega^{(1)}-\frac{1}{2}(\alpha\cdot\mathbf{1})\right)\\
&\times\left(\alpha\cdot(\Phi(\omega))+N_\omega^{(2)}-\frac{1}{2}(\alpha\cdot\mathbf{1})\right)\Bigg]\ .
\end{aligned}$$

$$(|M=m)=4\times\left(\frac{n}{2^{n+1}}\times\sigma^2+\frac{\sigma^4}{2^n}\right)\ .$$

Similar to proof of Lemma .1 (see Appendix .1) using the affine model instead of the Hamming Weight and Assumption 1.

Then we can prove the Theorem 5.6.3.

Lemma B gives us the value of the variance of the noise. Then by the definition of the , we have:

$$
\begin{aligned}
&, M \geqslant X^{(3)}, M \\
&\iff \frac{\alpha^2 \cdot M}{(|M = m)} \geqslant \frac{\alpha \cdot M}{N^{(3)}} \\
&\iff \frac{\frac{1}{4} 4\alpha^4}{4 \times \left(\frac{\sigma^2}{2^{n+1}} \times n + \frac{\sigma^4}{2^n} \right)} \geqslant \frac{\frac{1}{4} n}{\sigma^2} \\
&\iff \frac{\sigma^2}{4} \times 4\alpha^4 - \frac{\sigma^2}{2^{n+1}} \times n^2 - \frac{\sigma^4}{2^n} \times n \geqslant 0 \\
&\iff \sigma^2 \times \left(\frac{1}{4} \times 4\alpha^4 - \frac{1}{2^{n+1}} \times n^2 - \frac{\sigma^2}{2^n} \times n \right) \geqslant 0 \\
&\iff \frac{1}{4} \times 4\alpha^4 - \frac{1}{2^{n+1}} \times n^2 - \frac{\sigma^2}{2^n} \times 2\alpha^2 \geqslant 0 \\
&\iff \frac{2^n}{4} \times \frac{4\alpha^4}{n} - \frac{2^n}{2^{n+1}} \times \frac{n^2}{n} \geqslant \sigma^2 \\
&\iff 4\alpha^4 \times \frac{2^{n-2}}{n} - \frac{n}{2} \geqslant \sigma^2
\end{aligned}
$$

C. Proof of Corollary 5.6.3

Let us first prove the following result: Let $x \in \mathbb{R}^n$ and let p, q two integers such that $p > q > 0$. Then:

$$n^{\frac{1}{p} - \frac{1}{q}} qx \leqslant px \leqslant qx \ . \tag{6}$$

These two bounds are tight. Indeed,

$$
\begin{aligned}
&\forall i, j, x_i = x_j && \implies n^{\frac{1}{p} - \frac{1}{q}} qx = px \\
&\exists i / x_i \neq 0 \text{ and } \forall i \neq j, x_j = 0 && \implies px = qx \ .
\end{aligned}
$$

Let us first prove the lower bound of Eq. (6). By the Hölder inequality we have:

$$\sum_i |x_i|^q \leqslant \left(\sum_i (|x_i|^q)^P \right)^{\frac{1}{P}} \left(\sum_i (1)^Q \right)^{\frac{1}{Q}} \text{ where } P = \frac{p}{q} \text{ and } Q = \frac{p}{p-q} \ .$$

So, we have, $\sum_i |x_i|^q \leqslant px^q n^{1 - \frac{q}{p}}$, i.e., $pxn^{\frac{1}{p} - \frac{1}{q}} \leqslant px$.

Then let us prove the upper bound. We have $\sum_i \frac{|x_i|^q}{qx^q} = 1$. Hence, for all $1 \leqslant i \leqslant n$, $\frac{|x_i|^q}{qx^q} \leqslant 1$. Therefore, for all i, $\frac{|x_i|^p}{qx^p} \leqslant \frac{|x_i|^q}{qx^q}$, hence $\sum_i \frac{|x_i|^p}{qx^p} \leqslant \sum_i \frac{|x_i|^q}{qx^q} = 1$, which yields the

announced inequality: $px^p \leqslant qx^p$.

Let us prove the sufficient conditions when the inequalities become equalities:

$$\forall i, j, x_i = x_j \implies px = |x_i| n^{\frac{1}{p}} \text{ and } qx = |x_i| n^{\frac{1}{q}} \implies px = qx n^{\frac{1}{p}-\frac{1}{q}}$$
$$\exists i, x_i \neq 0 \text{ and } \forall i \neq j, x_j = 0 \implies px = |x_i| \text{ and } qx = |x_i| \implies px = qx.$$

The Corollary 5.6.3 is the application of Lemma C with $p = 4$ and $q = 2$.

Indeed we have by Theorem 5.6.3 that the useful interval of variance is $0 \leqslant \sigma^2 \leqslant 4\alpha^4 \times \frac{2^{n-2}}{n} - \frac{n}{2}$, where $2\alpha^2 = n$ (recall Assumption 1). Then by Lemma C:

$$\left(2\alpha n^{\frac{1}{4}-\frac{1}{2}+\frac{1}{2}}\right)^4 \times \frac{2^{n-2}}{n} - \frac{n}{2} \leqslant 4\alpha^4 \times \frac{2^{n-2}}{n} - \frac{n}{2} \leqslant 2\alpha^4 \times \frac{2^{n-2}}{n} - \frac{n}{2}$$
$$\implies \left(2\alpha n^{\frac{1}{4}-\frac{1}{2}}\right)^4 \times \frac{2^{n-2}}{n} - \frac{n}{2} \leqslant 4\alpha^4 \times \frac{2^{n-2}}{n} - \frac{n}{2} \leqslant n^2 \times \frac{2^{n-2}}{n} - \frac{n}{2}$$
$$\implies \underbrace{2^{n-2} - \frac{n}{2}} \leqslant 4\alpha^4 \times \frac{2^{n-2}}{n} - \frac{n}{2} \leqslant n \times 2^{n-2} - \frac{n}{2}$$

Value of Theorem 5.4.3.

16
CHAPTER 7

.5. SPN vs DFA: Good Design Practices

While we have shown that it is almost impossible to design an SPN that provides the same level of security against DFA as the classical security, in the presence of non-uniform fault models, the results also show that not all the SPNs with the same classical security level provide the same level of security against DFA. For example, GIFT-64 and GIFT-128 required 12 pairs of faulty and non-faulty ciphertexts with very specific fault models, located in different locations, while AES-128 requires only 2 pairs, with the fault injected in the same location every time and can be uniformly random. Here, we list some of the techniques we believe make DFA harder to perform and help GIFT be harder to break than the other ciphers considered in this paper:

1. Using Sboxes with irregular DDTs and higher linearity means that, on average, the number of solutions when injecting faults is higher.
2. Using smaller Sboxes means that the gap between the number of possible solutions for each Sbox before and after applying DFA is smaller, hence, the information leakage is lower.
3. The lower the branching number of the diffusion layer, the harder it is to accelerate the attack and reduce the number of faults required.
4. The smaller the ratio between the round key size and the master key size, the lesser the information leaked during applying DFA on the final rounds.
5. Using a non-symmetric diffusion layer (e.g. GIFT) makes the JDDT more complex, as opposed to a symmetric diffusion layer (e.g. PRESENT).
6. While it is a good practice to design the round key to be smaller than the master key (e.g. LED-128) and the block size (e.g. GIFT), it is not a good practice to skip adding the round key to some of the final Sboxes altogether, as this may leak vital information about other Sboxes in the same round (e.g. SKINNY).

In general, as discussed in Section 7.4.1, the goal of the designer should be to minimize the number of active Sboxes before they start interacting with each other.

Table 2: The probability of having a key space of size k for the last round key K_{31} after 2 fault injections

k	0	1	2	3	4	5	6	7	8
$Pr(x=k)$	0.779	0.195	0.0.024	0.002	1×10^{-4}	6×10^{-6}	2×10^{-7}	9×10^{-9}	2×10^{-10}

This can be achieved by having a very slow diffusion when the number of active Sboxes in a given round is very small (1 or 2), and very fast diffusion, otherwise. For example, SKINNY has very slow diffusion, but even when 7 out of 16 Sboxes are active, the 4-round DFA attack we describe in Section 7.6.3 can be still mounted. However, even if the designer manages to come up with a design that satisfies all of these guidelines, it is still impossible to prevent DFA against SPNs completely. So the ultimate goal should be to increase the number of faults required and/or use other constructions that may prevent DFA.

.6. More Case Studies to our Techniques

A. PRESENT-128 and Practical Implementations of PRESENT: Finding Optimal DFA Attack

PRESENT-128

The key scheduling algorithm for PRESENT-128 is similar to present PRESENT-80 (refer to [177], Appendix II, for full description), where

$$K_{31} = [\kappa_{127}\kappa_{126}...\kappa_{64}] \tag{7}$$

$$K_{30} = [\kappa_{60}\kappa_{59}...\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\tilde{\kappa_{16}}\tilde{\kappa_{15}}...\kappa_0]\mathtt{truncate}(sb^{-1}([\kappa_{127}\kappa_{126}\kappa_{125}\kappa_{124}],3)) \tag{8}$$

We note that only 3 bits overlap between K_{31} and K_{30}: $\kappa127, \kappa126$, and $\kappa125$. However, since none of these bits is active in round 30, for both the fault locations we are considering, we conclude that there is no gain in performing step 2 while attacking PRESENT-128. Hence, the number of key candidates for K_{31} and K_m are 2^{31} and 2^{95}, respectively. An interesting result is that the increased security of PRESENT-128 is not just due to the increased Key size, but due to the good design of the larger key scheduling algorithm, which leads to $P_{ks} = 1$. However, Table 2 shows that by applying the attack twice, the last round key K_{31} can be identified as 1 of 8 candidates with probability almost 1. Afterwards, the attack can be applied to a round-reduced version of PRESENT-128 for each of these candidates, targeting K_{30} as the last round key, resulting in 2^{34} or 2^6 master key candidates after applying one or two more faults in round 28, respectively.

Practical Implementations

The attacks described in this section require the injection of a fault $\delta = [1111]_2$ in the output of the sBoxLayer of round 29, in either nibble 7 or nibble 11. While this

requirement can be challenging, it was shown in [159] that such fault is practically possible. Moreover, there are a couple of tricks that the attacker can use to get around this requirement. First, we notice that if instead of injecting $\delta = [1111]_2$ at the output of the Sbox, we inject $\delta^{'} = [1000]_2$ at the corresponding input, there is a 0.25 chance that the correct fault is triggered at the output. We can observe the occurrence of the correct value by the number of active groups in round 31. Besides, if we inject 4 of such faults, there is a very high probability that the required fault occurs, and the other three pairs can be used to further decrease the key space size. The same applies for $\delta^{'} = [1111]_2$, $\delta^{'} = [0110]_2$, and $\delta^{'} = [0111]_2$. Hence, depending on the precision of the equipment the attacker has, he can target 1-, 2- or 3-bit flips with probability 0.25 of getting the required difference, or 4-bit flips (with probability 1).

Moreover, for some specific implementations, the attack can be performed using even a single uniform random fault. In case of a software, bit-sliced implementation of width w, a uniform random fault is injected in the most significant bit of the input of the Sbox at nibble 7, round 29. Since each instance is triggered with probability 0.5, and the instances that are triggered have probability 0.25 of getting the required difference at the output of the Sbox at round 29, then it is expected that $\frac{w}{8}$ instances will have the required value and $\frac{w}{2}$ will be active in general, potentially allowing to even uniquely identify the key using only a single fault. This expectation can be achieved even using architecture as small as 8-bit micro-controllers.

B. GIFT-64: New Results

GIFT cipher was proposed in CHES 2017 by Banik et al. [395] as a more lightweight version of PRESENT. It has two versions: GIFT-64 and GIFT-128. Both these versions have a 128-bit master key, but they differ in the block size, 64 bits and 128 bits, respectively. In this section we analyze only the first version, noting that the techniques from our paper can bet extended with slight modifications to attack GIFT-128, as well. The differences between GIFT-64 and PRESENT-128 are as follows:

1. It runs for 28 rounds only.
2. The Sbox and mixing layer are different.
3. The Key Scheduling algorithm is also different, where 32 bits are extracted from the key state every round, followed by a linear state update. The property that is of interest to our analysis is that every 4 consecutive round keys are independent and uniquely identify the master key.
4. Every round, half of the bits (32 bits), only, are mixed with key bits.

Table 3: Part of the DDT of the Sbox used in GIFT

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	μ_{HW}
1	0	0	0	0	0	2	2	0	2	2	2	2	2	0	0	2	2.25
2	0	0	0	0	0	4	4	0	0	2	2	0	0	2	2	0	2.25
4	0	0	0	2	0	4	0	6	0	2	0	0	0	2	0	0	2.5
8	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4	3

Table 4: Part of the auxiliary DDT of the Sbox used in GIFT with respect to the active key bits

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	EXP(SOLs)
1	0	0	0	0	0	2	2	0	1	2	2	2	1	0	0	2	$2^{0.75}$
2	0	0	0	0	0	4	4	0	0	2	2	0	0	2	2	0	$2^{1.5}$
4	0	0	0	2	0	4	0	4	0	2	0	0	0	2	0	0	$2^{1.625}$
8	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	2	2^{1}

From the structure of GIFT-64, we conclude that the best single-nibble fault value in round 26 is $[1111]_2$ at the output of one of the Sboxes. Similar to to PRESENT, that fault value can be achieved by injecting a fault $[1000]_2$ at the input of the Sbox, with probability 25% (refer to [395], Appendix C.2, for the DDT of GIFT Sbox). However, due to the properties of the mixing layer, regardless of which nibble the fault is injected in, such fault will always trigger the four groups of round 27 by the same four different input differences $\{[0001]_2, [0010]_2, [0100]_2, [1000]_2\}$, up to a cyclic rotation operation. Table 3 shows the corresponding rows of the DDT of GIFT Sbox. Instead of selecting the difference with the largest average Hamming weight, as in the case of PRESENT, in order to find the average number of active Sboxes in the last round, in the case of GIFT-64 the average number of Sboxes in the last round is always 10. Since only two key bits are added to the output of each Sbox, we need to compute an auxiliary DDT, which corresponds to the number of solutions of the active key bits, shown in Table 4. Since, in round 28, each group is triggered with a different input difference, in Table 5 we calculate the different probabilities for each Sbox in round 28 to be active. Combined with the number of solutions in Table 4, for every possible input difference, the expected number of candidates for each group in the last round is computed, which leads to an overall number of $2^{18.1}$ key candidates for K_{28}. In order to identify K_{28} uniquely, we need 3 repetitions of the whole attack, respectively, which reduces the master key space to 2^{96} keys. The attack then needs to be repeated four times for the 4 last round keys, required 12 fault injections, on average.

Table 5: The probability distribution of different output bit differences when a single input bit is active for GIFT Sbox

	0	1	2	3	EXP(Sols)
1	0.5	0.5	0.5	0.75	$2^{5.452}$
2	0.5	0.5	0.75	0.5	$2^{6.204}$
4	1	0.5	0.75	0.5	$2^{3.384}$
8	1	1	0.5	0.5	$2^{3.059}$

C. GIFT-128: New Results

Any two consecutive rounds of GIFT-128 can be viewed as two parallel independent instances of GIFT-64. We use this representation for rounds 38 and 39, while round 40 is the final round and, as explained in Section 7.5, the shuffling and mixing operations in the last round have no effect on our analysis. Hence, the same analysis used for GIFT-64 can be used for GIFT-128. First, 3 faults are used to recover half the bits of K_{40}, then another three faults are used to recover the other half. The attack is then repeated for K_{39}.

GIFT-128 is very similar to GIFT-64 except for the following differences:

1. The block size is 128 bits, divided into 8 groups of 4 nibbles, each.
2. 64-bit round keys are used.
3. Only every two successive round keys are independent, as opposed to 4 rounds in the case of GIFT-64.
4. The description of the shuffling operation is slightly different. It can be viewed as two steps; matrix transposition, followed by interleaving.

From this description, any two successive rounds of GIFT-128 can be viewed as a pair of parallel and independent two rounds of GIFT-64. Since in the case of GIFT-128, we need only two round keys, the number of faults and complexity of the attack is exactly the same as in the case of GIFT-64.

D. PRIDE: Finding Optimal DFA Attack

PRIDE [396] is an SPN-based cipher that can be considered as from the same family of ciphers as PRESENT and GIFT, but targeted towards low latency application. Hence, it uses a more complicated diffusion layer to achieve faster diffusion. In [161], the authors showed that by flipping 16 adjacent bits at the input of the linear diffusion layer in the second to last round, all the Sboxes in the last round can be activated with known input difference. However, since flipping 16 bits is not an easy task and requires high precision, we analyze the last round limiting the number of bits to be flipped to 4 (single-fault model). The location of the fault is the same as [161], round $r-1$. Injecting faults in earlier rounds does not help due to the fast diffusion of PRIDE which will increase the number of active interacting Sboxes. This limits the possible fault locations to 16 nibbles. We tested all the 16 possibilities and found out that the nibbles that maximize the number of active Sboxes in the last round are nibbles: 1, 5 and 13, with a maximum of 9 active Sboxes in the last round. The active Sboxes are shown in Table 6. We notice that if we use all these three pairs, each Sbox has appeared at least once. Hence, we conclude that 4 pairs of faulty and non-faulty ciphertexts are required on average

Table 6: The active Sboxes in the last round depending on the fault location

Faulty Location	Active Sboxes
1	0,2,3,6,7,8,11,12,15
5	0,1,4,5,6,9,10,13,14
13	0,2,3,6,7,8,11,12,15

to uniquely identify the last round key, and then the attack can be repeated for the previous round. This is double the number of faults required in [161], but using a simpler and less demanding fault model. The exact complexities are given in Section 9.1.

.7. Proofs for Section 7.3

If ΔX is sampled from $\mathcal{S}$, such that $|\mathcal{S}| = z$ and $P(\Delta X) = p_x$.then the expected number of leaked bits of K, when ΔY is observed is

$$n - \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x,\Delta y}}{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}} log(\frac{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}}{p_x}) \tag{9}$$

Since, using Bayes' law, we have

$$\begin{aligned}
&P(\Delta X = \Delta x | \Delta Y = \Delta y) \\
=&\frac{P(\Delta Y = \Delta y | \Delta X = \Delta x) P(\Delta X = \Delta x)}{P(\Delta Y = \Delta y)} \\
=&\frac{P(\Delta Y = \Delta y | \Delta X = \Delta x) P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} P(\Delta Y = \Delta y | \Delta X = \Delta x_j) P(\Delta X = \Delta x_j)} \\
=&\frac{\frac{s_{\Delta x,\Delta y}}{2^n} P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} P(\Delta X = \Delta x_j)}.
\end{aligned}$$

Therefore,

$$
\begin{aligned}
& H(K|Z_1, Z_2) \\
&= \sum_{\Delta X \in \mathcal{S}} P(\Delta X = \Delta x | \Delta Y = \Delta y) \times \\
& (H(X|\Delta X = \Delta x, \Delta Y = \Delta y) - log(P(\Delta X = \Delta x | \Delta Y = \Delta y))) \\
&= \sum_{\Delta X \in \mathcal{S}} \frac{\frac{s_{\Delta x, \Delta y}}{2^n} P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j, \Delta y}}{2^n} P(\Delta X = \Delta x_j)} \times \\
& \left(log(s_{\Delta x, \Delta y}) - log\left(\frac{\frac{s_{\Delta x, \Delta y}}{2^n} P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j, \Delta y}}{2^n} P(\Delta X = \Delta x_j)} \right) \right) \\
&= \sum_{\Delta X \in \mathcal{S}} \frac{\frac{s_{\Delta x, \Delta y}}{2^n} p_x}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j, \Delta y}}{2^n} p_{x_j}} log\left(\frac{2^n \sum_{\Delta X_j \in \mathcal{S}} \frac{s_{\Delta x_j, \Delta y}}{2^n} p_{x_j}}{p_x} \right) \\
&= \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x, \Delta y}}{2^n \sum_{\Delta X_j \in \mathcal{S}} \frac{s_{\Delta x_j, \Delta y}}{2^n} p_{x_j}} \left(n + log\left(\frac{\sum_{\Delta X_j \in \mathcal{S}} \frac{s_{\Delta x_j, \Delta y}}{2^n} p_{x_j}}{p_x} \right) \right) \\
&= \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x, \Delta y}}{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j, \Delta y} p_{x_j}} log\left(\frac{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j, \Delta y} p_{x_j}}{p_x} \right)
\end{aligned}
$$

Given a pair of faulty and correct ciphertexts Z_1 and Z_2, if $\Delta X \in \{0,1\}^n$ is a uniform random variable, then $H(K|Z_1 Z_2) = n$, regardless of the properties of the function $S(x)$. Since

$$
\begin{aligned}
& H(\Delta X | \Delta Y = \Delta y) \\
&= - \sum_{\Delta x \in \{0,1\}^n} Pr(\Delta X = \Delta x | \Delta Y = \Delta y) log(Pr(\Delta X = \Delta x | \Delta Y = \Delta y)) \\
&= - \sum_{\Delta x \in \{0,1\}^n} \frac{s_{\Delta x, \Delta y}}{2^n} log\left(\frac{s_{\Delta x, \Delta y}}{2^n} \right),
\end{aligned}
$$

and

$$
\begin{aligned}
& H(X_1 | \Delta X \Delta Y) \\
&= \sum_{\Delta x \in \{0,1\}^n} H(X_1 | \Delta X = \Delta x, \Delta Y = \Delta y) Pr(\Delta X = \Delta x, \Delta Y = \Delta y) \\
&= \sum_{\Delta x \in \{0,1\}^n} \frac{s_{\Delta x, \Delta y}}{2^n} log(s_{\Delta x, \Delta y}),
\end{aligned}
$$

from Equation (7.1), we have

$$
\begin{aligned}
& H(K|Z_1Z_2) \\
= & \sum_{\Delta x\in\{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} log(s_{\Delta x,\Delta y}) - \frac{s_{\Delta x,\Delta y}}{2^n} log(\frac{s_{\Delta x,\Delta y}}{2^n}) \\
= & \sum_{\Delta x\in\{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} log(\frac{2^n}{s_{\Delta x,\Delta y}}) = n \sum_{\Delta x\in\{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} = n
\end{aligned}
$$

If $\Delta X = \Delta x$ (constant), then using one pair (Z_1, Z_2), the key space can be reduced from 2^n to $s_{\Delta x,\Delta y}$. If ΔX is constant, then $H(\Delta X|\Delta Y) = 0$ and $H(X_1|\Delta X\Delta Y) = H(X_1|\Delta X = \Delta x, \Delta Y = \Delta y) = log(s_{\Delta x,\Delta y})$. Hence, $H(K|Z_1Z_2) = log(s_{\Delta x,\Delta y})$. Only linear (affine) Boolean functions achieves the theoretical security bound $H(K|Z_1Z_2) = n\forall\Delta x$, regardless of the distribution of ΔX. In order for a function $S(x)$ to achieve $H(K|Z_1Z_2) = n\forall\Delta x$ for any distribution, $s_{\Delta x,\Delta y}$ must be equal to 2^n or $0\ \forall\ \Delta x, \Delta y\ \in \{0,1\}^n$, which is achieved only by linear and affine functions.

If $\Delta X = \Delta x$ (constant), then the expected number of leaked bits of K is $n - \sum_{\Delta y\in\{0,1\}^n} log(s_{\Delta x,\Delta y})P(\Delta Y = \Delta y|\Delta X = \Delta x)$. We have

$$
\begin{aligned}
& H(K|\Delta X) \\
= & \sum_{\Delta y\in\{0,1\}^n} H(X|\Delta X, \Delta y)P(\Delta Y = \Delta y|\Delta X = \Delta x) \\
= & \sum_{\Delta y\in\{0,1\}^n} log(s_{\Delta x,\Delta y})Pr(\Delta Y = \Delta y|\Delta X = \Delta x).
\end{aligned}
$$

17

CHAPTER 11

.8. Trigger generation for Hardware Trojans

For the hardware Trojan trigger signal, we exploit directly the temperature sensor measurement to generate the ***trigger*** signal. The device, used for this experiment, is Xilinx Virtex 5 FPGA mounted on SASEBO-GII boards. As described in the documentation [397], the temperature measurement is read directly on 10 bits signal output of system monitor. This output allows a value which varies from 0 to 1023. System monitor measurement allows to sense a temperature in range of $[-273, +230]$ hence the LSB of the 10 bits output is equal to $1/2$. At the normal operating temperature (25), system monitor output is around $605 = b'1001011101$. Thanks to this observation, we decided to use directly the 7^{th} bit of system monitor output as hardware Trojan trigger signal. The hardware Trojan will be activated when 7^{th} bit of monitor output is high, i.e., when the monitor output is superior to $640 = b'1010000000$. This value corresponds to 42. Therefore the trigger signal will be active when FPGA temperature is higher than 42. The trigger temperature can be easily changed according to the design under test. In our case study, a simple hair dryer of cost $5 is enough to heat the FPGA and reach this temperature. We assume that a system monitor is already instantiated in the design, to monitor device working conditions and the alarm is raised at a temperature higher than 42. In such a scenario, the hardware Trojan trigger part does not consume much extra logic and would result in a very low-cost hardware Trojan example.

Whenever we need to trigger the Trojan, we bring the heater circuit close to the FPGA. The FPGA heats up slowly to the temperature of 42 and raises the output bit to '1'. At this point, we switch-off the heater. Now this output bit stays '1' till the FPGA cools down below 42, therefore we cannot precisely control the duration of trigger in terms of cycle count. We further process this output bit of the system monitor to generate a precise duration trigger. This can be done with some extra logic. In other words, we need a small circuit which can generate a precise trigger signal when the output bit of system monitor goes to '1'. For the Trojans in Tab 11.1, we either need a trigger of 1 clock cycle or 12 clock cycles. Both these triggers can be generated by deploying one LUT and one flip-flop to process output bit of system monitor. Thus, we can generate a very small trigger circuit to trigger a zero-overhead hardware Trojan.

18

CHAPTER 13

.9. Key schedule in the fixsliced representation

A. GIFT-64

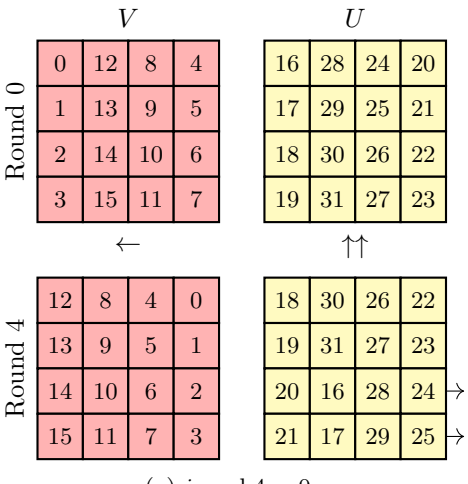
The first step of our proposed key schedule is to rearrange the bits of the first 4 round keys so that they match the fixsliced representation of the internal state for the first 4 rounds. We recall that it can be done only once, when the encryption key is being derived and/or stored on the device. Afterwards, we adjust the key schedule according to the 4 new representations. Because there are 4 different representations depending on the round number, there are 4 different ways to update the key. The new round key representations from rounds 0 to 3 and the corresponding key update functions are depicted in Figure .1. Note that our adjusted key update functions can basically be computed by means of nibble-wise and word-wise rotations.

B. GIFT-128

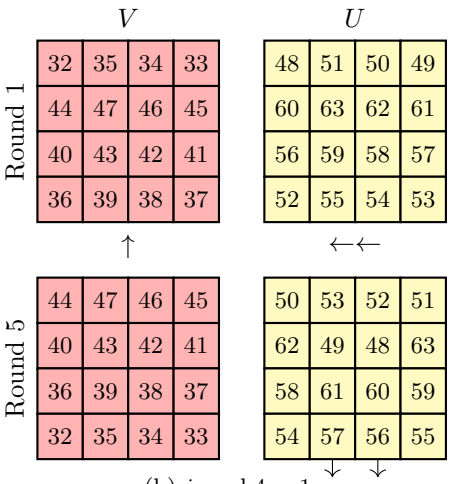
In the case of GIFT-128, adjusting the key schedule according to fixslicing is more tricky since the new and the classical representations of the state are synchronised after 5 rounds, while the key words will return to their original positions after 4 rounds. We suggest to compute the key schedule in the classical representation for the first 10 round before rearranging them in order to match the fixsliced representation of the state. At this stage, all key words will be expressed in each representation, allowing to adapt the key schedule for each of them, without reordering bits. As stated in Section 13.4.2, each key word will be exclusive-ORed to the state in the same representation every 10 rounds. After 10 rounds, 2 out of 4 key words will have been updated thrice while the two others will have been updated twice, as detailed in Table 7. Therefore, our adapted key schedule relies on double and triple update functions for each representation, which are illustrated in Figure .7.

Representation #	Round #	Round key	
		U	V
0	0	$W_2 \| W_3$	$W_6 \| W_7$
1	1	$W_0 \| W_1$	$W_4 \| W_5$
2	2	$(W_6 \| W_7)^1$	$W_2 \| W_3$
3	3	$(W_4 \| W_5)^1$	$W_0 \| W_1$
4	4	$(W_2 \| W_3)^1$	$(W_6 \| W_7)^1$
0	5	$(W_0 \| W_1)^1$	$(W_4 \| W_5)^1$
1	6	$(W_6 \| W_7)^2$	$(W_2 \| W_3)^1$
2	7	$(W_4 \| W_5)^2$	$(W_0 \| W_1)^1$
3	8	$(W_2 \| W_3)^2$	$(W_6 \| W_7)^2$
4	9	$(W_0 \| W_1)^2$	$(W_4 \| W_5)^2$
0	10	$(W_6 \| W_7)^3$	$(W_2 \| W_3)^2$
1	11	$(W_4 \| W_5)^3$	$(W_0 \| W_1)^2$
2	12	$(W_2 \| W_3)^3$	$(W_6 \| W_7)^3$
3	13	$(W_0 \| W_1)^3$	$(W_4 \| W_5)^3$
4	14	$(W_6 \| W_7)^4$	$(W_2 \| W_3)^3$
0	15	$(W_4 \| W_5)^4$	$(W_0 \| W_1)^3$
1	16	$(W_2 \| W_3)^4$	$(W_6 \| W_7)^4$
2	17	$(W_0 \| W_1)^4$	$(W_4 \| W_5)^4$
3	18	$(W_6 \| W_7)^5$	$(W_2 \| W_3)^4$
4	19	$(W_4 \| W_5)^5$	$(W_0 \| W_1)^4$
0	20	$(W_2 \| W_3)^5$	$(W_6 \| W_7)^5$
⋮	⋮	⋮	⋮

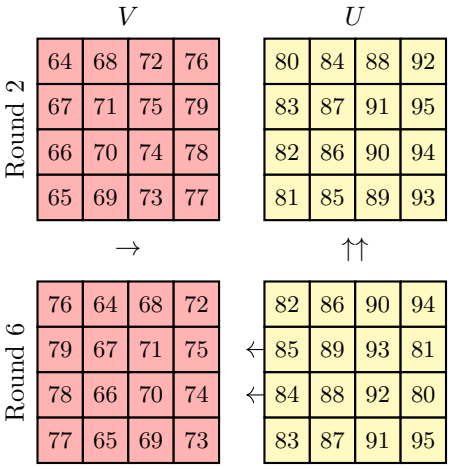
Table 7: Round keys' representations depending on the round number. Exponents refer to the number of times the key words have been updated. Blue and red arrows refer to double and triple key updates, respectively.



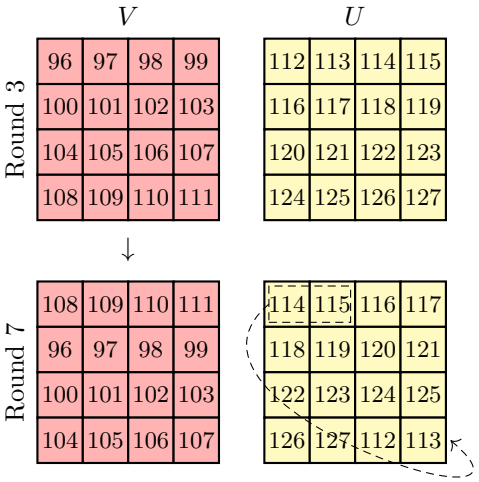
(a) $i \bmod 4 = 0$



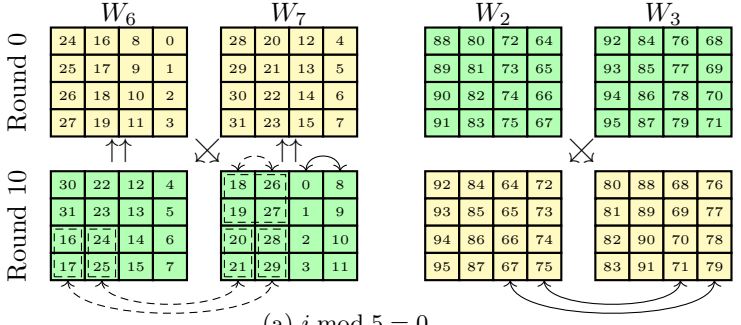
(b) $i \bmod 4 = 1$



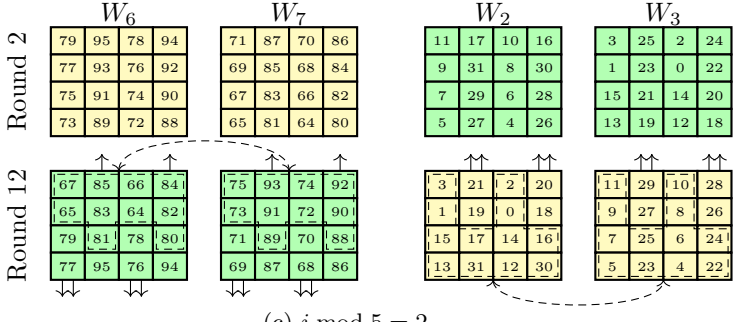
(c) $i \bmod 4 = 2$



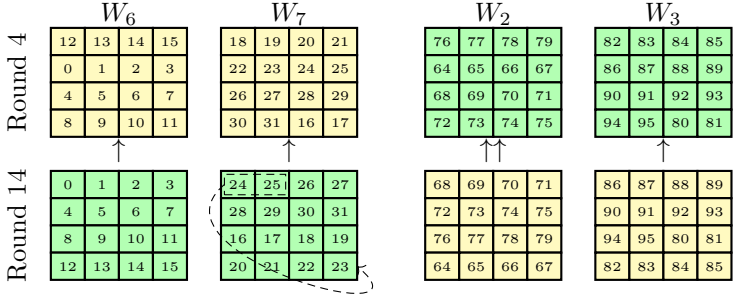
(d) $i \bmod 4 = 3$



(a) $i \bmod 5 = 0$

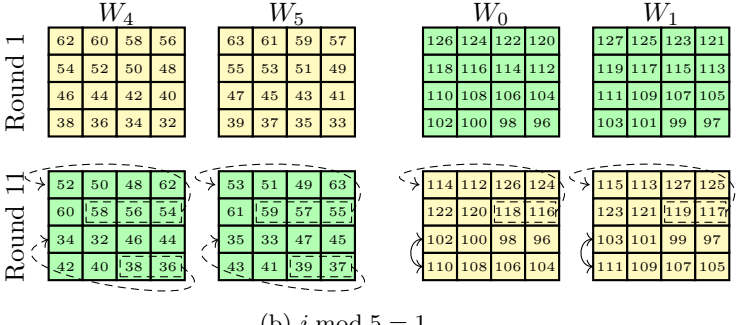


(c) $i \bmod 5 = 2$

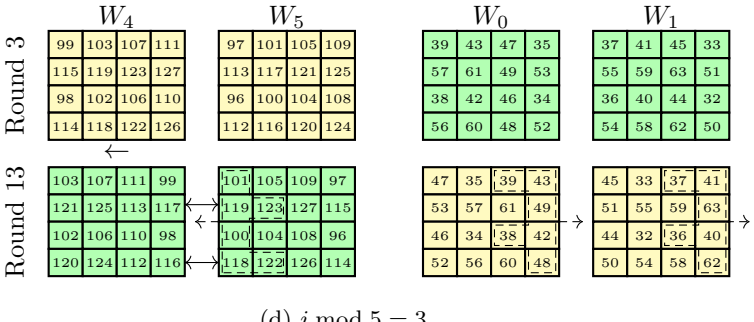


(e) $i \bmod 5 = 4$

Figure .6: $i \bmod 5 = 4$



(b) $i \bmod 5 = 1$



(d) $i \bmod 5 = 3$

Figure 9: GIFT-128 double/triple key update functions from round i to $i + 10$, according to the different fixsliced representations over 5 rounds. Each cell represents a bit, and the numbers in the cells then denote the actual index of that particular bit in the 16-bit key word. Note that $i \bmod 5 = 4$ refers to the classical representation. The cell colors match the corresponding slice for the add round key operation.

.42 Figure 9: GIFT-128 double/triple key update functions from round i to $i + 10$, according to the different fixsliced representations over 5 rounds. Each cell represents a bit, and the numbers in the cells then denote the actual index of that particular bit in the 16-bit key word. Note that i mod 5 = 4 refers to the classical representation. The cell colors match the corresponding slice for the add round key operation.

.10. Additional illustrations

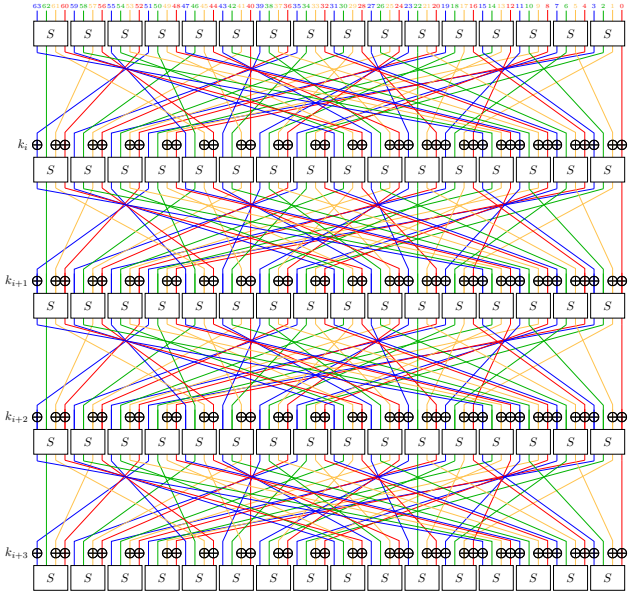


Figure .8: Classical representation of GIFT-64 over 4 rounds. Each color refers to a slice.

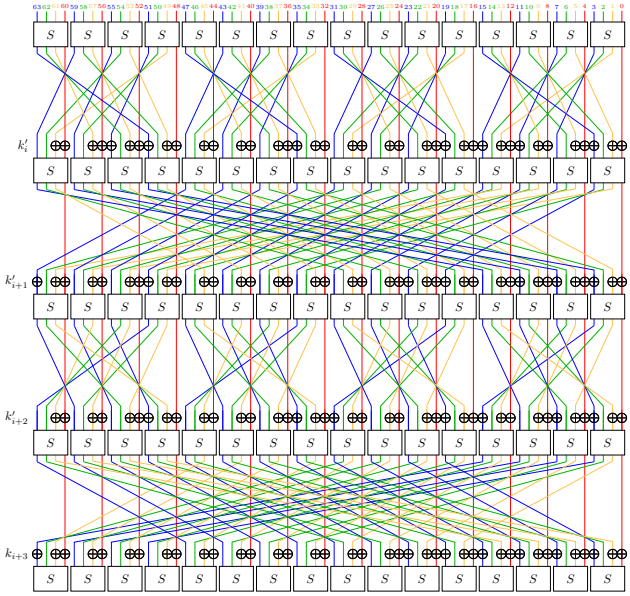


Figure .9: Fixsliced representation of GIFT-64 over 4 rounds. Each color refers to a slice.

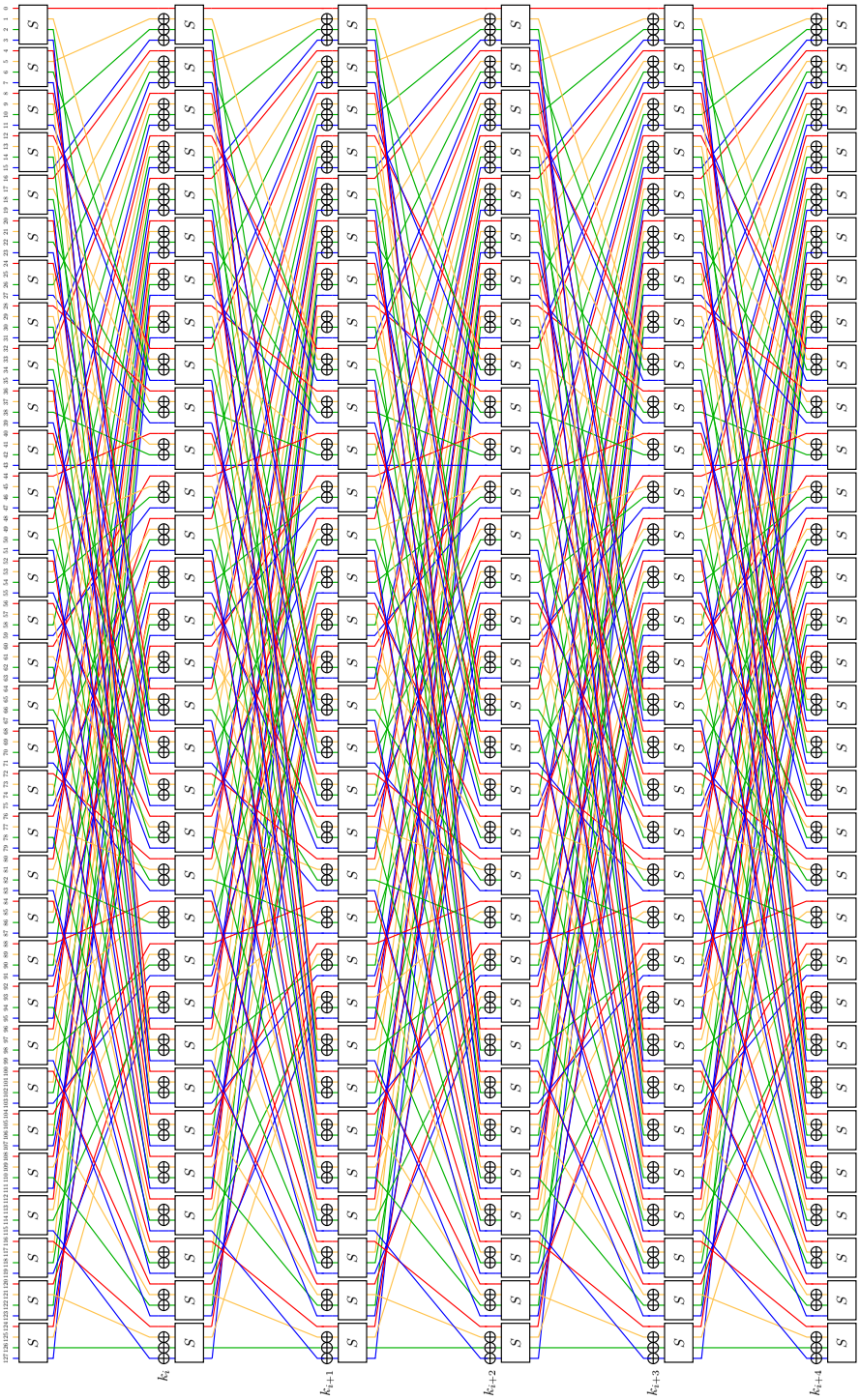


Figure .10: Classical representation of GIFT-128 over 5 rounds. Each color refers to a slice.

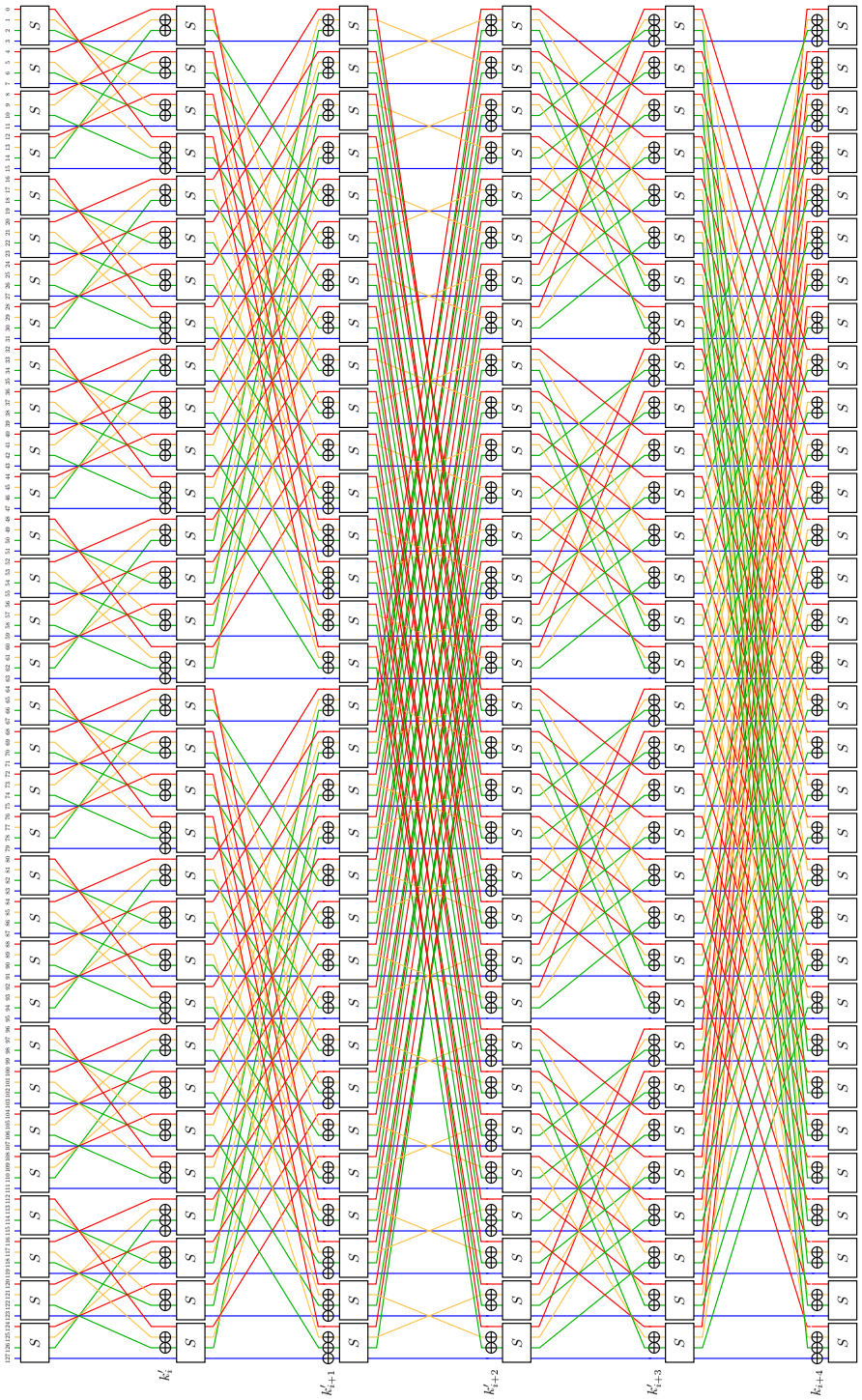


Figure .11: Fixsliced representation of GIFT-128 over 5 rounds. Each color refers to a slice.